Information Processing and Management 63 (2026) 104607

Contents lists available at ScienceDirect

Information Processing and Management

journal homepage: www.elsevier.com/locate/ipm

Q-value guided Text-to-SQL generation: Structured reasoning meets efficient inference exploration

Min Tang [a,b], Lixin Zou [c,*], Shujie Cui [b], Weiqing Wang [b], Zhe Jin [d], Chengliang Li [c], Shiuan-Ni Liang [a,*]

[a] School of Engineering, Monash University, Malaysia Campus, Bandar Sunway, 47500, Malaysia
[b] School of Information Technology, Monash University, Clayton, VIC 3800, Australia
[c] School of Cyber Science and Engineering, Wuhan Univeristy, Wuhan, 430072, Hubei, China
[d] School of Artificial Intelligence, Anhui University, Hefei, 230039, Anhui, China

ARTICLE INFO

Keywords:
Text-to-SQL
Large language model
Reasoning
MCTS

ABSTRACT

Translating natural language queries into executable SQL queries (Text-to-SQL) is essential for human-database interaction but remains a challenging task, especially for complex queries. Existing methods based on large language models (LLMs) often rely on supervised fine-tuning or iterative refinement based on execution success. However, these methods overlook errors that occur during the generation process, leading to error propagation issues. To address this, we propose the *Q*-value guided Text-to-SQL, which trains a *Q*-value network to evaluate SQL potential and avoid generation errors. Specifically, QSQL decomposes the complex SQL generation process into simpler and helpful intermediate steps, such as selection, schema linking, sub-query generation, and combination. Then, we adopt Monte Carlo Tree Search (MCTS) to estimate *Q*-values of each step and incorporate rule-based consensus filtering to eliminate inconsistent/low-quality data, removing the need for extra human annotations. Finally, a *Q*-value probe network approximates the estimated values for efficient generation. Experimental results on the `Spider` and `Bird` benchmarks show that QSQL achieves a superior balance between execution accuracy and inference efficiency. Compared to a baseline MCTS approach, QSQL improves execution accuracy by **9%** while reducing inference time to just **11.9%**.

1. Introduction

Text-to-SQL bridges human language and relational databases by translating natural language queries into executable SQL commands (Guo et al., 2025; Shi et al., 2024). This task has evolved from early handcrafted methods, such as abstract syntax trees (Lin et al., 2019; Yu et al., 2018a) and predefined sketches (Choi et al., 2021; Xu et al., 2017), to more advanced sequence-to-sequence deep learning architectures (Xu et al., 2017; Zhong et al., 2017). More recently, pre-trained language models (PLMs) have been integrated into this process (He et al., 2019; Lyu et al., 2020; Scholak et al., 2021), further enhancing performance. Recent large language models (LLMs) like CodeLlama (Roziere et al., 2023), CodeX (Chen et al., 2021a), and Qwen2.5-Coder (Hui et al., 2024) have shown promise in generating accurate and robust SQL queries from natural language inputs, marking a significant step forward in the field (Rajkumar et al., 2022). However, these models are primarily optimized for general coding tasks (e.g., code completion, bug fixing)

* Corresponding authors.
E-mail addresses: min.tang@monash.edu (M. Tang), zoulixin@whu.edu.cn, lixinzou@wuhan.edu.cn (L. Zou), shujie.cui@monash.edu (S. Cui), teresa.wang@monash.edu (W. Wang), jinzhe@ahu.edu.cn (Z. Jin), cllee@whu.edu.cn (C. Li), liang.shiuan-ni@monash.edu (S.-N. Liang).

https://doi.org/10.1016/j.ipm.2025.104607
Received 1 June 2025; Received in revised form 22 November 2025; Accepted 31 December 2025
Available online 15 January 2026

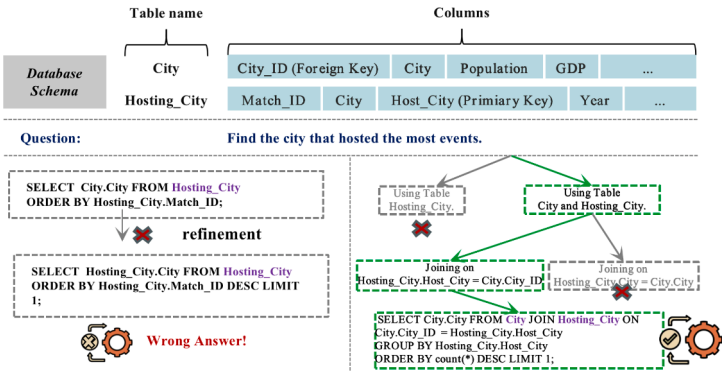


Fig. 1. Examples of text-to-SQL generation: The left side shows results without procedure guidance, while the right includes proper procedure guidance. Relying only on overall refinement may lead to incorrect answers, but using procedure guidance helps avoid issues from the start.

rather than tailoring for SQL generation. This limitation becomes evident in their inability to address domain-specific challenges such as schema linking, and generating filters (Chen et al., 2021a; Xu et al., 2022).

Consequently, pioneering work has explored LLMs for Text-to-SQL tasks in two key directions: **First, training or fine-tuning specialized models** on SQL-related datasets, *e.g.*, Spider (Yu et al., 2018b) or Bird (Li et al., 2024b). For example, studies (Li et al., 2024a; Sun et al., 2023) have adapted models like GPT-3 to generate SQL queries with supervised training. However, these models often rely on greedy decoding, which lacks the reasoning to handle complex queries (e.g., nested subqueries or multi-table joins). **Second, reasoning techniques** have been applied to improve the performance of LLMs, including Chain-of-Thought (CoT) prompting (Liu & Tan, 2023; Tai et al., 2023; Tan et al., 2024; Wei et al., 2022), Self-Correction (Chen et al., 2023b; Pourreza & Rafiei, 2024; Zhang et al., 2024), and Tree Search (Chen et al., 2024). Specifically, C3 (Dong et al., 2023) and DIN-SQL (Pourreza & Rafiei, 2024) decompose SQL generation into sub-tasks using GPT, then aggregate the results to produce the final SQL. Recently, the work in (Chen et al., 2024) treats Text-to-SQL as a multi-step task for language agents and leverages the MCTS planning method to search for optimal SQL.

While these methods achieve promising results, they face two key challenges. First, **these reasoning methods emphasize final performance but neglect procedural correctness in SQL generation**, which we define as the correctness of intermediate steps in SQL generation, such as selecting tables, linking columns to the schema, and generating filters. This oversight can lead to error propagation, which is difficult to correct using simple methods like refinement or an overall discriminant model (Chen et al., 2024; Pourreza & Rafiei, 2024). As shown in Fig. 1, we compare the results of text-to-SQL generation with and without procedure guidance. Without guidance, relying solely on refinement can only fix executable errors or correct obvious mistakes. However, a well-trained guidance model for procedures can carefully evaluate different options and prevent errors from occurring in the first place. Second, **exploration techniques like tree search are theoretically powerful but computationally intensive for real-time applications**. Previous studies have reported that tree search can require 10–20 times the inference time compared to naive methods (Brandfonbrener et al., 2024).

Towards this end, this study explores using procedure guidance to enhance the quality of SQL generation for complex queries. However, it is non-trivial as recent work has highlighted challenges in training components like the value function or Progress Reward Model (PRM) in large language models (Setlur et al., 2024; Wu et al., 2024). These challenges stem from difficulties in accurately estimating correctness, *i.e.*, PRM, or potential outcomes, *i.e.* (*Zhang* et al., *2025*), value functions (Chen et al., 2021b). We encountered similar issues and found they often arise from noise in automatically generated data. This issue may stem from data leakage during LLM pre-training, where models inadvertently memorize SQL results from the training data, rather than learning to generalize to new queries. This phenomenon is analyzed in Section 5.3.

To solve the above challenges, this work introduces a Text-to-SQL generation method, named QSQL. Specifically, our approach comprises three core components. **(1) Step-wise Chain-of-Thought Decomposition**: We decompose SQL generation into simpler intermediate subtasks, *i.e.*, selection, schema linking, generating filters, and final combination. This modular design simplifies multi-step reasoning and schema alignment by breaking complex queries into manageable steps, enabling easier verification of intermediate results. **(2) Consensus Filtered Q-value Estimation**: To accurately estimate the Q-value, which quantifies the expected correctness of intermediate decisions, we employ a modified MCTS to explore high-quality intermediate data efficiently. During the backup phase of MCTS, we integrate consensus filtering to discard noisy data with correct final outputs but flawed intermediate steps, ensuring robust training data. **(3) Q-value Probe Network**: We distill the Q-value function estimated via MCTS into a lightweight probe network at the final hidden states of the last token in each MCTS step. This design minimizes memory usage and latency. During inference, the estimated Q-values are generated alongside standard beam search decoding, with only a slight increase in beam search size (set to 2) in large language models (LLMs). This approach enables efficient generation with minimal computational overhead. Finally, our contributions are validated on benchmark datasets like Spider (Yu et al., 2018b) and Bird (Li et al., 2024b), achieving a superior balance between execution accuracy and inference efficiency.

The key contributions of this work are listed below:

Table 1
The step-wise CoT decomposition framework structures the SQL generation process into four key steps: selection, schema linking, generating filters, and combination.

Step	Aim	Prompt
Selection	`Identify the tables required to answer the question.`	`Please analyze the tables that are relevant for generating the SQL. After completing the analysis, list only the used tables with the format: 'tables': ['table1', 'table2'].`
Schema Linking	`Extract columns from the selected tables relevant to the query.`	`Please analyze columns that will be useful for generating the SQL. After completing the analysis, list only the useful columns in the following format: 'columns': ['column1', 'column2'].`
Generating Filters	`Specify filter conditions (e.g., WHERE, HAVING) needed for the query.`	`Analyze and define the join conditions required for generating the SQL query. After completing the analysis, list only the conditions in the following format: 'conditions': ['condition1', 'condition2'].`
Combination	`Combine tables, columns, and conditions into a valid SQL statement.`	`Based on the provided schema, directly generate the SQL query for the given question by leveraging identified relevant tables, columns, and conditions.`

- This work first introduces a step-wise chain-of-thought (CoT) decomposed generation approach that mimics human step-wise reasoning, significantly improving SQL accuracy.
- For mitigating error propagation arising from overlooking procedural correctness during reasoning, we propose an MCTS framework enhanced with consensus filtering, which suppresses misleading intermediate outputs during Q-value training.
- To address the computational overhead of tree search techniques in real-time applications, we design a memory-efficient Q-value network that integrates seamlessly with LLM decoding, enabling fast inference without compromising performance.

2. Objectives

This study aims to achieve the following objectives:

- To improve the accuracy of LLMs on the Text-to-SQL task by mitigating error propagation caused by neglecting procedural correctness in the reasoning process.
- To accelerate inference efficiency in Text-to-SQL applications that rely on tree search techniques.

3. Preliminary

3.1. Notation and problem statement

Given a natural language question q and a database schema and values $D = \langle T, C, V \rangle$, the goal is to generate a correct SQL query y. Here, $T = \{t_1, t_2, \ldots, t_i\}$ represents a set of tables, $C = \{c_1, c_2, \ldots, c_j\}$ represents a set of columns, and $V = \{v_1, v_2, \ldots, v_k\}$ represents the database values. The task is to learn a function $f : Q \times D \rightarrow Y$ that maps the question $q \in Q$ and D to the SQL query $y \in Y$, ensuring that y accurately retrieves the intended data from the database. In more complex scenarios, such as databases with significant gaps between questions and values (Li et al., 2024b), external knowledge K may be required to resolve ambiguities in the question or database values. The objective is to find $y = f(q, D, K)$ such that y satisfies the semantics of q under the constraints of D and K.

3.2. MDP-based formulation with LLMs

We model text-to-SQL generation as a Markov Decision Process (MDP) (Puterman, 1990) to incorporate Q-value guidance during inference. The MDP is defined by states, actions, transitions, and rewards as follows: The **state** S_t captures the context, including the input database schema D, external knowledge K, user query q, and interaction history $\{s_1, a_1, s_2, a_2 \ldots, s_t\}$, where s_t represents a sub-task used to help generate the final SQL. The **action** a_t is the output generated to address sub-task s_t. Since language models produce a set of tokens A_t, probabilistically, a_t is the extracted code from the token set. The **transition** is deterministic, updating the state as $S_{t+1} = S_t \cup \{a_t, s_{t+1}\}$. The **reward** r_t is a binary function, awarding 1 if the final SQL executes correctly and 0 otherwise. Our objective is to learn a Q-value function that guides action selection (a_t) to reach the target SQL output y by optimizing cumulative rewards. An example of MDP-formatted SQL generation is presented in Fig. 2.

4. Method

Our proposed method is structured into three core components, as depicted in Fig. 3. First, **step-wise CoT decomposition** breaks SQL generation into manageable sub-tasks, enabling systematic verification and simplifying complex query resolution. **Consensus-filtered MCTS** estimates intermediate state-action values $\tilde{Q}(S_t, a_t)$ while leveraging consensus filtering to refine data quality and

```
Please finish the task: Convert Question into an appropriate SQL query based on the detailed database schema
and demo data.

-- Database student_loan
CREATE TABLE bool ( "name" TEXT default '' not null primary key )
DEMOS:
neg
pos

CREATE TABLE person ( "name" TEXT default '' not null primary key )
DEMOS:
student1
student10

CREATE TABLE disabled ( "name" TEXT default '' not null primary key, foreign key ("name") references person
("name") on update cascade on delete cascade )
DEMOS:
student114
student125

CREATE TABLE enlist ( "name" TEXT not null, organ TEXT not null, foreign key ("name") references person
("name") on update cascade on delete cascade )
DEMOS:
student40|fire_department
student51|fire_department

CREATE TABLE filed_for_bankrupcy ( "name" TEXT default '' not null primary key, foreign key ("name")
references person ("name") on update cascade on delete cascade )
DEMOS:
student122
student126

CREATE TABLE longest_absense_from_school ( "name" TEXT default '' not null primary key, "month" INTEGER
default 0 null, foreign key ("name") references person ("name") on update cascade on delete cascade )
DEMOS:
student10|0
student102|0

CREATE TABLE male ( "name" TEXT default '' not null primary key, foreign key ("name") references person
("name") on update cascade on delete cascade )
DEMOS:
student1
student101

CREATE TABLE no_payment_due ( "name" TEXT default '' not null primary key, bool TEXT null, foreign key
("name") references person ("name") on update cascade on delete cascade, foreign key (bool) references bool
("name") on update cascade on delete cascade )
DEMOS:
student10|neg
student101|neg

CREATE TABLE unemployed ( "name" TEXT default '' not null primary key, foreign key ("name") references
person ("name") on update cascade on delete cascade )
DEMOS:
student1000
student102

CREATE TABLE `enrolled` ( `name` TEXT NOT NULL, `school` TEXT NOT NULL, `month` INTEGER NOT NULL DEFAULT 0,
PRIMARY KEY (`name`,`school`), FOREIGN KEY (`name`) REFERENCES `person` (`name`) ON DELETE CASCADE ON UPDATE
CASCADE )
DEMOS:
student10|smc|1
student101|ucb|1

-- Question: How many students belong to the navy department?
-------------------------------------------------------------------------------------------------------

Subtask: Please analyze the tables that are relevant for generating the SQL. After completing the analysis,
list only the used tables with the format: {'tables': ['table1', 'table2']}.

Action: {'tables': ['enlist', 'person']}
-------------------------------------------------------------------------------------------------------

Subtask: Please analyze columns that will be useful for generating the SQL. After completing the analysis,
list only the useful columns in the following format: {'columns': ['column1', 'column2']}, without further
analysis.
Action: {'columns': ['name', 'organ']}
```

Fig. 2. An example state explored by our Consensus Filtered MCTS from the Bird dataset, specifically S_5.

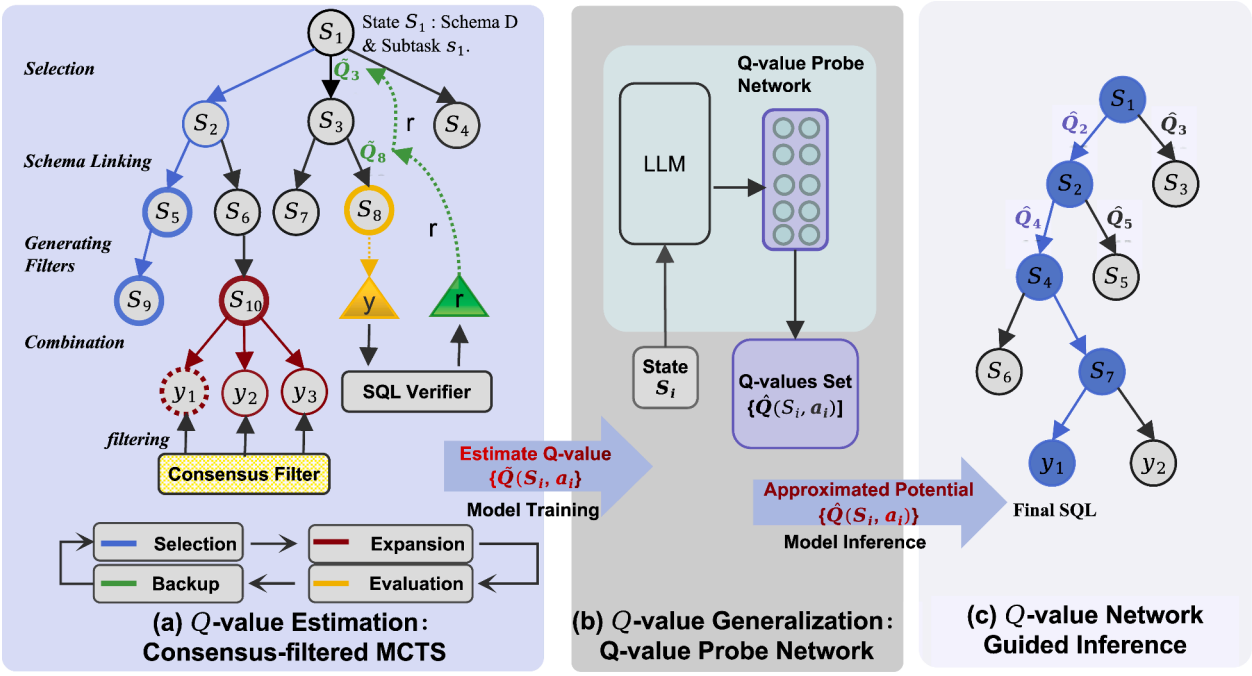


Fig. 3. The overall procedure of proposed QSQL. **(a)** The consensus-filtered MCTS algorithm iteratively performs four steps, selection, expansion, evaluation, and consensus-filtered backup, to explore SQL generation solutions and collect estimated Q-values. **(b)** The Q-value Probe network is subsequently trained using this intermediate data. **(c)** During SQL generation, the tree is traversed by selecting the edge with the highest predicted state-action Q-value, as determined by the Probe network.

improve value estimation accuracy. Second, **Q-value Probe Network Training** (Fig. 3 (b)) distills the MCTS-estimated Q-values into a lightweight network for efficient prediction of intermediate action quality. Third, **Inference under Q-value Guidance** (Fig. 3 (c)) uses the trained Probe Network to guide SQL generation. At each step, beam search selects actions with the highest predicted Q-values.

4.1. Step-wise CoT decompositions

To simplify SQL generation and provide structured guidance, we introduce a step-wise chain-of-thought decomposition framework. This framework breaks the SQL generation task into discrete, manageable sub-tasks, denoted as $\{s_1, s_2, \ldots, s_T\}$. Specifically, we divide the process into four key stages: selection, schema linking, generating filters, and combination. **Selection** identifies the tables required to answer the question. **Schema linking** extracts columns from the selected tables that are relevant to the query. **Generating filters** specifies the filter conditions needed for the query. **Combination** merges the above components to generate the final SQL. Each stage is designed with clear objectives and detailed prompts, as outlined in Table 1.

4.2. Consensus filtered Q-value estimation

To estimate the Q-value, we employ Monte Carlo Tree Search (MCTS) to explore SQL generation solutions. By leveraging learned state values, this approach avoids greedy pitfalls and ensures high-quality, diverse outputs. Additionally, to address noise in the exploration data, we enhance the Q-value update further by introducing a consensus-filtered backup mechanism. This mechanism systematically removes noise and improves overall quality by aggregating only reliable data.

Exploration with MCTS. In the MCTS algorithm, each edge (S_t, a_t) in the search tree tracks an average action value $\tilde{Q}(S_t, a_t)$ and a visit count $N(S_t, a_t)$. The algorithm updates $\tilde{Q}(S_t, a_t)$ through four key steps-selection, expansion, evaluation, and backup-to iteratively refine its estimates, as illustrated in Fig. 3 (a). These steps ensure accurate action-value approximations by balancing exploration and exploitation.

In the **selection** phase, the algorithm starts at the root state and traverses the tree via simulation (i.e., descending the tree by predicting state outcomes without immediate backup). At each time step t during the simulation, an action a_t is chosen from the

current state S_t using the following criterion:

$$a_t = \arg\max_{a \in \mathcal{A}} \tilde{Q}(S_t, a_t) + \underbrace{\frac{c_{\text{puct}}}{|\{a_*\}|} \cdot \frac{\sqrt{\sum_{b \in \{a_*\}} N(S_t, b)}}{1 + N(S_t, a_t)}}_{\text{exploration bonus}} \tag{1}$$

where $\{a_*\}$ is the set of actions. The action with the highest combination of average value and an exploration bonus is selected. The exploration bonus decreases as actions are revisited, encouraging diverse exploration during tree search. c_{puct} is a constant that balances exploration and exploitation.

When the traversal reaches a leaf node S_L at step L, the leaf is **expanded** by initializing each outgoing edge with $\tilde{Q}(S_L, a) = 0$ and $N(S_L, a) = 0$. The leaf node is then **evaluated** by computing the average outcome $V(S_L)$ through rollouts, which proceed until the terminal state T using efficient policies such as random or greedy strategies. Finally, in the **backup** phase, the action values and visit counts of all edges along the traversal path are updated. For each edge (S_j, a_j) in the forward trace (where $j < L$), the updates are performed as follows:

$$\tilde{Q}(S_j, a_j) \leftarrow \frac{N(S_j, a_j) \cdot \tilde{Q}(S_j, a_j) + V(S_j)}{N(S_j, a_j) + 1} \tag{2}$$

$$V(S_j) \leftarrow V(S_{j+1}) + R(S_j, a_j) \tag{3}$$

$$N(S_{j+1}, a_j) \leftarrow N(S_{j+1}, a_j) + 1. \tag{4}$$

The reward function $R(S_j, a_j)$ outputs 1 if a_j is the correct final SQL query and 0 otherwise. These updates ensure that each edge tracks both the number of visits and the average evaluation score from all simulations traversing it.

Consensus Filtered Backup.

Though the theoretical basis of the $\tilde{Q}(S_j, a_j)$ estimator (Eq. 2) is sound, experiments show instability in training a generalizable Q-value network for SQL generation. Subsection 5.3 will analyze this issue. It stems from simple SQL queries that yield correct results via flawed steps, causing unreliable Q-value estimates. We hypothesize this arises from data leakage during pre-training, where models memorize outputs without solving sub-tasks. This leads to procedural inconsistencies, confusing the estimator and degrading performance.

To address these challenges, we introduce a consistency check into the Q-value update process. This mechanism verifies whether the selection, schema linking, and generating filters align with the final SQL query. If inconsistencies are detected, such as cases where correct outputs stem from flawed intermediate steps or *vice versa*, the corresponding data is pruned from the training set. Specifically, the modified backup step is designed as follows:

$$\tilde{Q}(S_j, a_j) \leftarrow \frac{N(S_j, a_j) \cdot \tilde{Q}(S_j, a_j) + \beta \cdot V(S_j)}{N(S_j, a_j) + \beta} \tag{5}$$

$$V(S_j) \leftarrow V(S_{j+1}) + \beta \cdot R(S_j, a_j) \tag{6}$$

$$N(S_{j+1}, a_j) \leftarrow N(S_{j+1}, a_j) + \beta, \tag{7}$$

where $\beta = 1$ only if the selection, schema linking, and generating filters are consistent with the final SQL output. Otherwise, $\beta = 0$, effectively discarding inconsistent data. Specifically, we employ **Abstract Syntax Tree** (AST) analysis to parse the final generated SQL query. Key structural elements-such as tables in the FROM clause and predicates in the WHERE clause-are extracted and compared against the decisions made during the intermediate states. This check reduces noise in the training process, leading to more reliable Q-value estimates. A detailed algorithm of Consensus Filtered MCTS is provided in Algorithm 1.

4.3. Q-value probe network

Model Training. To guide SQL generation during inference, we approximate the Q-value using a probe on the final token of the model's response. Specifically, for each reasoning sub-task, we compute the estimate Q-value $\hat{Q}_\theta(s_j, a_j)$ as:

$$\hat{Q}_\theta(s_j, a_j) = \boldsymbol{w}^\top \boldsymbol{h}_{\text{eos}} + b, \tag{8}$$

where $\boldsymbol{h}_{\text{eos}}$ is the last-layer hidden state of the end-of-sequence token, and b is a learnable bias. $\theta = \{\boldsymbol{h}_{\text{eos}}, b\}$ is the set of trainable parameters. This lightweight layer enables efficient inference by operating directly on the hidden state.

We train the network by minimizing the Mean Squared Error (MSE) between predicted and target Q-values as:

$$\arg\min_\theta \frac{1}{|\{S_j, a_j\}|} \sum_{S_j, a_j} (\hat{Q}_\theta(S_j, a_j) - \tilde{Q}(S_j, a_j))^2. \tag{9}$$

Model Inference. During inference, we employ step-by-step CoT decoding for each subtask, using beam search (beam width = 2), as depicted in Fig. 3(c). Simultaneously, the model also generates Q-values to evaluate candidate potential and prune low-valued branches.

Algorithm 1: Consensus Filtered Monte Carlo Tree Search.

Input: Language model M, maximum trials number N.
Output: States and Estimated Q-values $\mathbb{D} = \{S_t, \tilde{Q}_t\}$

```
# Initialization.
Initialize 𝔻 = { }.
Initialize value function with 0.
Set the number of trials k = 1.
for k <= N do
    Initialize time step t = 1, and state S_t.
    for t <= T do
        # State Generation.
        Generate action a_t according to Eq. 1.
        Execute the a_t and collect S_{t+1}.
        Do rollout get the final reward r.
        Do consensus filtering and get β.
        # Updating estimation.
        Updating the Q̃(S_t, a_t) with Eq. 2.
        Increase the training trial k ← k + 1.
        if k == N then
            Updating 𝔻 ← D ∪ {𝕊_t, Q̃_t}
        end
    end
end
return States and Estimated Q-values 𝔻
```

5. Experiments

To evaluate the effectiveness of QSQL, we carried out a series of experiments on benchmark datasets derived from real-world scenarios. Our investigation primarily focuses on assessing translation performance and inference efficiency.

5.1. Experimental settings

5.1.1. Datasets

To evaluate the performance of various methods and validate our findings, we utilized two widely recognized datasets in the Text-to-SQL research domain.

- **Spider** (Yu et al., 2018b): A cross-domain Text-to-SQL benchmark comprising 10,181 questions, over 200 databases, and 5693 training and 1034 development queries. The dataset includes complex queries involving multi-table joins (37%) and nested queries (23%).
- **Bird** (Li et al., 2024b): This is a newly introduced large-scale dataset specifically designed for Text-to-SQL tasks, with a focus on real-world applications. It comprises 12,751 text-to-SQL pairs and 95 databases with a total size of 33.4 GB, spanning 37 professional domains.

The dataset splits follow the same protocol as in previous work (Li et al., 2024a). Detailed statistics for both datasets are presented in Table 2.

5.1.2. Metrics

We evaluated Text-to-SQL system performance using two widely used metrics. This evaluation approach is consistent with prior works (Chen et al., 2024; Li et al., 2024a) in the field.

- **Execution Accuracy (EX)** (Cai et al., 2017; Zhong et al., 2017): This metric checks if the predicted SQL query produces the same result as the ground truth when executed on the target database, directly measuring correctness. We report EX on both the Spider and Bird datasets.
- **Test-Suite Accuracy (TS)** (Cai et al., 2017; Zhong et al., 2020, 2017): To mitigate EX's false positives (e.g., incorrect queries coincidentally matching outputs), TS assesses robustness by verifying if the query passes EX across multiple database instances generated via automated augmentation. In our work, we reported TS for Spider but not for Bird for the following two main reasons. TS is well-suited for Spider due to its small, clean databases and **pre-built test suites**. In contrast, Bird contains large, complex real-world databases, and no such test suites are currently available. Building them would be highly costly and has not yet been done. Second, EX is a sufficient and practical metric for Bird, as it directly assesses a query's correctness on real databases-aligning

Table 2
Statistics of Experimental Datasets.

Dataset	Part	Easy	Medium	Hard	Extra Hard	Overall
Spider	Train Set	-	-	-	-	8659
	Dev Set	248	446	174	166	1034
	Part	Simple	Moderate	Challenging	Overall	
Bird	Train Set	-	-	-	9428	
	Dev Set	925	465	144	1,534	

Table 3
Statistics of data generated by the proposed Consensus Filtered Monte Carlo Tree Search.

Dataset	Text-to-SQL tasks in Train Set	Successful Tasks	States and Estimate Q-values	Average per Task
Spider	8659	6226	194,639	31
Bird	9428	1228	122,248	99

with Bird's focus on real-world applicability. The values of TS and EX would be very similar on Bird, making it redundant to report both.

5.1.3. Baseline methods

In all experiments, we used three popular code-specific LLMs, **Codes** (Li et al., 2024a), **CodeLlama** (Roziere et al., 2023), and **Qwen2.5-Coder** (Hui et al., 2024), along with one general-purpose LLM, **ChatGLM-4** (GLM et al., 2024), as the foundational models for our evaluation. Then, we compare our proposed method against the intrinsic performance of LLMs, a CoT-based method, and a state-of-the-art tree-search-based approach for Text-to-SQL tasks. The baseline methods and corresponding settings are outlined as follows:

- **Naive LLMs**: These baselines directly leverage the model's built-in Text-to-SQL capabilities without fine-tuning, serving as a lower bound for different methods. For a fair comparison, we used similarly sized LLMs: Codes-7B, CodeLlama-7B, Qwen2.5-Coder-7B, and ChatGLM-4-9B (Li et al., 2024a).
- **CoT-based Method**: Beyond the most straightforward baseline, we implemented an approach based on the CoT method as a comparison (Wei et al., 2022). This method divides the SQL generation into four subtasks, but it does not perform progress checking on each subtask.
- **MCTS-based method** (Chen et al., 2024): This method employs a deterministic model as the guidance and utilizes MCTS to identify the optimal SQL query for a given question. Since our (QSQL) also utilizes MCTS technology, it serves as a strong baseline for evaluating both parsing performance and efficiency. For a fair comparison, we chose the Qwen2.5-Coder-7B model as the generator of MCTS and observation-enhanced fine-tuned Qwen2.5-Coder-7B as the discriminator of the MCTS baseline method.

5.1.4. Implementation details

In this work, we used Qwen2.5-Coder-7B and ChatGLM-4-9B as the backbone models. During MCTS, we set the beam search width to 8, the temperature of the backbone model to 1.0, and the maximum generation length to 4080 tokens. **This process yielded 316,887 intermediate data, including states and corresponding Q-values extracted from the trees (60,115 positives and 256,772 negatives). Detailed statistics are summarized in** Table 3. The Q-value network was trained using the same backbone models, augmented with a regression head to predict Q-values from prior stages. The network was trained using generated samples from a single dataset (either Spider or Bird). Training used 1 epoch, a batch size of 1, a learning rate of 1×10^{-5}, and a cosine learning rate scheduler. During inference, we prompted the LLMs with a beam search width of 2 to balance accuracy and efficiency. Experiments involving MCTS and inference were conducted using up to four NVIDIA A100 Tensor Core GPUs (80GB). Lastly, to ensure the statistical validity of our performance gains, all reported results are averaged over 5 independent runs using different random seeds. We employed a two-tailed Student's t-test for all pairwise comparisons between our method and the baselines.

5.2. Main experimental results

We conducted comprehensive comparisons between variants of our proposed method (ChatGLM-4-9B-QSQL and Qwen2.5-Coder-7B-QSQL) and other baselines on Text-to-SQL tasks, evaluating both generation accuracy and efficiency.

Generation Accuracy. The SQL generation performance of our method and all baselines on the two benchmark datasets is presented in Table 4 and Table 5. The improvement metric is computed by comparing the best-performing variant of our method with the best-performing baseline for each dataset and evaluation metric. Based on these results, we highlight the following observations:

- **Our proposed method, QSQL, achieves a superior performance on the benchmark datasets.** It significantly outperforms all baselines, with an average improvement of over 9% across both datasets. Due to the Bird dataset's more complex database schemas, fewer states and Q-value samples could be collected compared to Spider (detailed statistics are provided in Table 3). These samples are used to train the Q-value probe networks; as a result, the improvement on Bird is slightly smaller than on Spider.

Table 4
Comparison of different methods on the Spider Dev dataset with a zero-shot setting. Shadowed rows denote our models, and the best method is highlighted in **bold**. The improvement ratio is computed using the accuracy of our Qwen2.5-Coder-PRM-7B model relative to the best baseline, Qwen2.5-Coder-7B (MCTS). A symbol of $*$ indicates that the performance improvement over the second-best result is statistically significant, with the p-value ≤ 0.05.

Method	Execution Accuracy(%)					Test-Suite Accuracy(%)				
	Easy	Medium	Hard	Extra Hard	Overall	Easy	Medium	Hard	Extra Hard	Overall
Codes-7B	75.0	69.1	40.8	24.70	58.6	72.6	58.3	31.0	13.9	50.0
CodeLlama-7B	51.2	46.9	32.2	27.1	42.3	42.9	37.7	23.0	12.7	32.2
ChatGLM-4-9B	76.6	68.4	43.7	38.0	61.3	73.4	59.0	34.5	21.1	52.2
Qwen2.5-Coder-7B	88.3	80.7	65.5	37.3	72.9	85.9	74.9	60.3	27.1	67.4
ChatGLM-4-9B(CoT)	75.4	65.0	47.1	27.7	58.5	71.8	54.5	38.7	18.7	50.4
Qwen2.5-Coder-7B(CoT)	85.1	74.7	62.1	37.3	69.1	81.9	67.5	54.6	25.9	62.1
Qwen2.5-Coder-7B(MCTS)	88.8	81.8	67.8	42.2	75.2	87.1	77.2	63.1	28.6	69.9
***ChatGLM-4-9B-QSQL**	82.1	74.9	52.4	45.5	68.8	79.7	66.7	41.7	25.0	59.0
***Qwen2.5-Coder-7B-QSQL**	**92.1**	**90.2***	**75.0***	**54.4***	**82.7***	**92.5**	**84.8***	**67.9***	**34.5***	**76.6***
Improvement	3.7%	10.2%	10.6%	28.9%	10.0%	6.2%	9.8%	7.6%	20.6%	9.6%

Table 5
Comparison of different methods on the Bird dev dataset with zero-shot settings.

Method	Execution Accuracy(%)			
	Simple	Moderate	Challenging	Overall
Codes-7B	21.8	6.8	9.0	16.1
CodeLlama-7B	21.0	7.3	6.0	15.5
ChatGLM-4-9B	31.9	13.3	13.9	24.6
Qwen2.5-Coder-7B	28.4	22.0	14.7	31.2
ChatGLM-4-9B(CoT)	30.0	12.0	9.7	22.7
Qwen2.5-Coder-7B(CoT)	36.0	19.2	14.5	28.9
Qwen2.5-Coder-7B(MCTS)	38.9	23.3	15.0	32.0
ChatGLM-4-9B-QSQL	34.2	19.7	11.3	27.7
Qwen2.5-Coder-7B-QSQL	**40.3**	**29.4***	**16.9***	**35.0***
Improvement	3.6%	26.2%	12.7%	9.3%

Table 6
Execution accuracy comparison of SFT model on Spider across difficulty levels.

Method	Easy	Medium	Hard	Extra Hard	Overall
Qwen2.5-Coder-7B	88.3	80.7	65.5	37.3	72.9
Qwen2.5-Coder-7B (SFT)	91.5	88.5	70.1	51.5	80.6
Qwen2.5-Coder-7B-QSQL	92.1	90.2	75.0	54.4	82.7
Qwen2.5-Coder-7B-QSQL (SFT)	93.0	91.2	75.7	56.8	83.8

- **QSQL excels at generating complex SQL queries, validating our initial intuition.** As the task complexity increases, the performance gap between QSQL and the baselines generally widens, with the exception being the *Challenging* level in Bird. This trend is evident in both evaluation metrics (see tables). The method's strength in handling complex tasks stems from its step-wise CoT decomposition and the guidance provided by the Q-value function, which enhances controllability for intricate queries.
- **Integration with supervised fine-tuning (SFT) further boosts performance.** To further demonstrate the effectiveness of our method, we incorporated QSQL into an SFT setting. Experiments with Qwen2.5-Coder-7B on the Spider dataset show that performance can be further improved by adding test-specific adaptations in LLMs. As shown in Table 6, the overall EX of Qwen2.5-Coder-7B-QSQL (SFT) surpasses that of Qwen2.5-Coder-7B (SFT) by 3.2%.

Generation Efficiency. To evaluate generation efficiency, we compare QSQL with the naive method and the representative MCTS (Chen et al., 2024) method. Specifically, due to differences in implementation, we compare the speed using similarly sized LLM backbones on the Spider Dev dataset, *i.e.,* CodeLlama-7B and Qwen2.5-Coder-7B. All inference tasks are executed on an NVIDIA A100 Tensor Core GPU (80GB), with the batch size set to 1 and beam width set to 2. The experimental results for the four models are presented in Fig. 4. As shown in the figure, **the proposed method only causes a minimal increase in inference time**. Our Qwen2.5-Coder-7B-QSQL achieves a significantly faster inference speed of **10.5 seconds per query**, compared to **88.2 seconds per query** for CodeLlama-7B (MCTS). This improvement is achieved while also maintaining superior accuracy. Although QSQL involves a longer CoT-based procedure, it is parallelized within a single generation step by using the Q-value probe network.

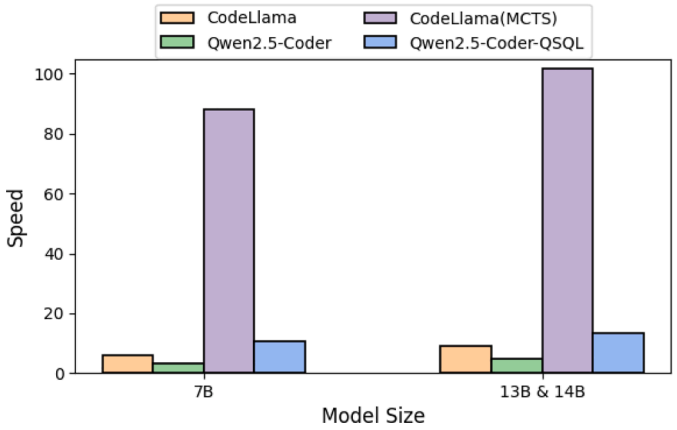


Fig. 4. Comparison of average inference speed (seconds) on the Spider Dev dataset.

Table 7
The impact of removing consensus filtering from MCTS using Qwen2.5-Coder-7B on the Spider Dev dataset.

Variant	Execution Accuracy (%)
w/ Consensus Filtering	82.7
w/o Consensus Filtering	68.3 (-14.4)

Table 8
Influence of the backbone language model.

Variant	Execution Accuracy (%)
Qwen2.5-Coder-7B-QSQL	82.7
Codes-7B	58.6
Codes-7B-QSQL	44.8 (-13.8)
CodeLlama-7B	42.3
CodeLlama-7B-QSQL	27.6 (-14.7)

5.3. Ablation studies

Our method comprises multiple components, and we conduct a series of ablation studies to analyze the contribution of each. This section investigates the impact of several key components in QSQL.

5.3.1. Effects of consensus filtering

To verify the effectiveness of the consensus-filtered mechanism, we removed it during the MCTS exploration phase and assessed its impact on the accuracy of the Q-value network. The experimental results on the Spider Dev dataset are shown in Table 7. As seen in the table, **removing consensus filtering leads to significantly worse performance compared to QSQL**. Specifically, the overall EX score drops from 82.7% to 68.3%, which is even lower than the intrinsic performance of 72.9%.

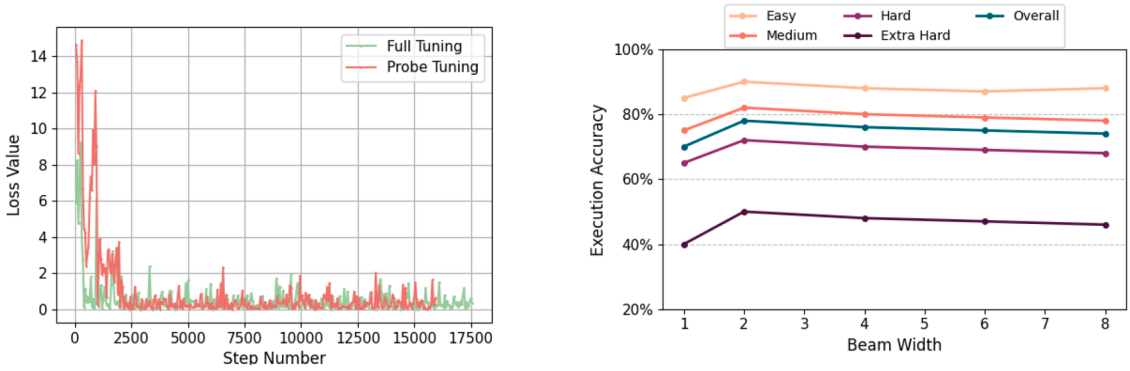
5.3.2. Influence of backbone model

The experiments reported in Table 4 and Table 5 do not include results for applying QSQL to Codes-7B and CodeLlama-7B. We add additional experiments on these models to address this, resulting in Codes-7B-QSQL and CodeLlama-7B-QSQL. The results are presented in Table 8. As shown in the table, **Codes-7B and CodeLlama-7B are not suitable for step-wise SQL generation**. We observe that the EX scores for both models dropped significantly, by over 10%. This issue stems from Codes-7B and CodeLlama-7B being overly specialized for coding tasks, and lacking the ability to follow step-by-step instructions.

To enhance computational efficiency, we fine-tune a lightweight probe network directly attached to the language model, diverging from the standard practice of using the full model for Q-value approximation. As a baseline comparison, we train an independent model by fine-tuning the entire architecture for Q-value prediction, leveraging the Qwen2.5-Coder-7B backbone. The training loss curves for both approaches are depicted in Fig. 5a. As illustrated, both methods achieve low loss values, demonstrating that the probe network attains comparable performance to the full model without compromising the model's inherent knowledge.

5.3.3. Influence of beam search width

We analyze the effect of beam search width in Fig. 5b, where the beam size b is varied over $1, 2, 4, 6, 8$. The results reveal a pattern: increasing b from 1 to 2 yields a substantial performance gain, indicating that modest exploration of multiple candidate sequences



(a) Training the Q-value probe network (red) vs full model fine-tuning (green).

(b) Impact of beam width under Qwen2.5-Coder-7B-QSQL.

Fig. 5. (a) Validity of Q-value probe network. (b) Influence of beam search width.

can significantly enhance SQL generation quality. However, further enlarging the beam width leads to stable or slightly degraded performance.

This phenomenon can be explained by the trade-off between search diversity and the inclusion of low-quality SQL candidates. When the beam width is small, moderate exploration enables the model to capture better candidate SQL queries, improving overall accuracy. In contrast, excessively wide beams increase the likelihood of including low-quality intermediate SQLs. Given that the Q-value network's predictions are not perfectly accurate, some of these poor candidates may receive erroneously high scores, resulting in their retention during decoding. This behavior is consistent with theoretical expectations in structured sequence generation.

6. Related work

In this section, we first review related work relevant to our approach, and then analyze the differences between QSQL and prior methods.

6.1. LLMs for Text-to-SQL

The Text-to-SQL task, which involves translating natural language queries into executable SQL statements, has seen significant advancements with the integration of LLMs. Current research primarily explores two approaches: prompt engineering and supervised fine-tuning. The former focuses on optimizing prompt engineering strategies to enhance LLM performance, such as Few-shot Learning (Tai et al., 2023), Retrieval Augmented Generation(RAG) (Gao, Xiong, Gao, Jia, Pan, Bi, Dai, Sun and Wang, 2023b), and Reasoning (Chen et al., 2024). The fine-tuning approach involves training LLMs on Text-to-SQL-specific datasets using either full fine-tuning (FFT) or parameter-efficient fine-tuning (PEFT) methods (Shi et al., 2024). For example, Sun et al. (2023) fine-tuned the PaLM-2 (Anil et al., 2023) model for Text-to-SQL tasks, achieving state-of-the-art results on the Spider dataset. Wang et al. (2023) developed a multi-agent framework for Text-to-SQL and fine-tuned CodeLlama for collaborative SQL generation, while (Gao, Wang, Li, Sun, Qian, Ding and Zhou, 2023a) fine-tuned several open-source LLMs, demonstrating the vast potential of fine-tuning strategies in this domain. The fine-tuning approach typically demands larger-scale training datasets. Our method aims to enhance performance by an efficient reasoning technique.

6.2. Reasoning methods for Text-to-SQL

Numerous studies have underscored the critical role of reasoning in enhancing the performance of LLMs (Huang & Chang, 2022; Plaat et al., 2024). Therefore, researchers have developed a variety of reasoning methods to refine LLMs for Text-to-SQL tasks. These methods range from the foundational approaches- CoT prompting such as Din-SQL (Pourreza & Rafiei, 2024), Divide-and-Promp (Liu & Tan, 2023), CoE-SQL (Zhang et al., 2024), ACT-SQL (Zhang et al., 2023), SELF-DEBUGGING (Chen et al., 2023a), and Least to Most prompt (Arora et al., 2023), to more sophisticated decomposition techniques that systematically break down complex queries into simpler sub-tasks (Khot et al., 2022). Decomposition methods, like QDecomp (Tai et al., 2023), iteratively break down complex questions into reasoning steps. MAC-SQL (Wang et al., 2023) employs a multi-agent framework (selector, decomposer, refiner) to generate intermediate sub-questions and SQLs sequentially. DEA-SQL (Xie et al., 2024) follows a global decomposition process, including information determination, classification Hint, SQL generation, self-correction, and active learning, to refine SQL generation systematically. However, these approaches often neglect to verify the correctness of each intermediate step based on final execution outcomes, leading to error propagation and unreliable reasoning. In contrast, our proposed method addresses this issue by introducing a consensus filter, which ensures the accuracy of intermediate steps and mitigates the risk of error propagation.

6.3. Differences with existing work

We have identified significant differences between our QSQL and existing works, especially the MCTS baseline (Chen et al., 2024), as follows. **(1) More Reasonable Step Definition in Reasoning**: The previous approaches (Chen et al., 2024; Liu & Tan, 2024; Pourreza & Rafiei, 2024; Xia et al., 2024) rely on SQL rewriting or correction, but overlook the error during the crucial reasoning process. But QSQL employs a step-wise CoT decomposition, breaking SQL generation into manageable intermediate steps, making it easier to verify intermediate results to ensure consensus reasoning. **(2) Optimized backup strategy in MCTS**: During the backup phase of our consensus-filtered MCTS, we integrate consensus filtering to discard noisy data with correct final outputs but flawed intermediate steps, ensuring robust training data. **(3) Significantly Improved Inference Efficiency**: The existing MCTS method utilizes computationally expensive MCTS-based planning to search for the optimal SQL during inference. But QSQL uses consensus-filtered MCTS only for collecting high-quality SQL generation states and estimated Q-values, which are then used to train the Q-value network, without relying on MCTS during inference. **(4) Lower Computational Cost**: The MCTS baseline method requires two LLMs (generator and discriminator), whereas QSQL uses a single LLM with a lightweight probe network, reducing memory and computation cost.

7. Conclusion and limitations

In conclusion, this paper presents QSQL, a novel Q-value guided text-to-SQL generation method that significantly improves the accuracy and efficiency of SQL query generation from natural language inputs. Through step-wise chain-of-thought decomposition, consensus-filtered Q-value estimation, and a memory-efficient Q-value probe network, QSQL addresses key challenges in existing LLM-based approaches, such as error propagation and computational inefficiency. Our experiments on benchmark datasets like Spider and Bird demonstrate a better performance both in translating accuracy and inference efficiency, highlighting the effectiveness of our method.

The limitations of the proposed method are primarily threefold. First, we observed limitations when applying QSQL to certain code-specific foundational models, such as the Codes model. Our approach depends on robust general language understanding and reasoning capabilities, which these models might lack. Second, the Q-value probe network is designed for specific models, requiring training and fine-tuning for each language model, despite its relatively low computational cost. Third, we acknowledge the challenge of domain shifts, as this work does not evaluate cross-dataset generalization. In future work, we plan to extend QSQL to cover a broader range of domains and propose a more general solution to reduce dependencies on specific models. We also aim to explore domain adaptation strategies to facilitate the transfer of the probe network's reasoning patterns.

CRediT authorship contribution statement

Min Tang: Writing – review & editing, Writing – original draft, Validation, Methodology, Investigation, Formal analysis, Conceptualization; **Lixin Zou:** Writing – review & editing, Writing – original draft, Resources, Methodology, Funding acquisition; **Shujie Cui:** Writing – review & editing, Supervision, Methodology; **Weiqing Wang:** Writing – review & editing, Supervision, Methodology; **Zhe Jin:** Writing – review & editing, Supervision; **Chengliang Li:** Writing – review & editing; **Shiuan-Ni Liang:** Writing – review & editing, Supervision, Resources.

Data availability

The code for replication is available at the link https://github.com/xuemingxxx/QSQL.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We express our sincere gratitude for the financial support provided by the National Natural Science Foundation of China (NO. U23A20305 and NO. 62302345), the Natural Science Foundation of Wuhan (NO. 2024050702030136) and the Xiaomi Young Scholar Program. The numerical calculations in this paper have been done on Monash High Performance Computing cluster and the supercomputing system in the Supercomputing Center of Wuhan University.

References

Anil, Rohan, Dai, Andrew M., Firat, Orhan, Johnson, Melvin, Lepikhin, Dmitry, Passos, Alexandre, Shakeri, Siamak, Taropa, Emanuel, Bailey, Paige, Chen, Zhifeng et al. (2023). PaLM 2 Technical Report. arXiv preprint. arXiv:2305.10403.

Arora, A., Bhaisaheb, S., Nigam, H., Patwardhan, M., Vig, L., & Shroff, G. (2023). Adapt and Decompose: Efficient Generalization of Text-to-SQL via Domain-Adapted Least-to-Most Prompting. arXiv preprint arXiv:2308.02582.

Brandfonbrener, D., Henniger, S., Raja, S., Prasad, T., Loughridge, C. R., Cassano, F., Hu, S. R., Yang, J., Byrd, W. E., Zinkov, R. et al. (2024). VerMCTS: Synthesizing multi-step programs using a verifier, a large language model, and tree search. In *The 4th workshop on mathematical reasoning and AI at neurIPS'24*.

Cai, Ruichu, Xu, Boyan, Yang, Xiaoping, Zhang, Zhifang, Li, Zhenjie, & Liang, Zhiyuan (2017). An Encoder–Decoder Framework Translating Natural Language to Database Queries. arXiv preprint arXiv:1711.06061.

Chen, Mark, Tworek, Jerry, Jun, Heewoo, Yuan, Qiming, Oliveira, Pablo H. P. de, Kaplan, Jared, Edwards, Harrison, Burda, Yuri, Joseph, Nicholas, Brockman, Greg et al. (2021a). Evaluating Large Language Models Trained on Code. arXiv preprint arXiv:2107.03374.

Chen, Xinyun, Lin, Mengdi, Schärli, Nathan, & Zhou, Denny (2023a). Teaching Large Language Models to Self-Debug. arXiv preprint arXiv:2304.05128.

Chen, Xinyun, Wang, Cheng, Zhou, Zhen, & Ross, Keith (2021b). Randomized Ensembled Double Q-Learning: Learning Fast Without a Model. arXiv preprint arXiv:2101.05982.

Chen, Zeyi, Chen, Shiqi, White, Michael, Mooney, Raymond, Payani, Ali, Srinivasa, Jayanth, Su, Yu, & Sun, Honglei (2023b). Text-to-SQL Error Correction with Language Models of Code. arXiv preprint arXiv:2305.13073.

Chen, Zeyi, White, Michael, Mooney, Raymond, Payani, Ali, Su, Yu, & Sun, Honglei (2024). When Is Tree Search Useful for LLM Planning? It Depends on the Discriminator. arXiv preprint arXiv:2402.10890.

ChoiDongHyun, ShinMyeong Cheol, KimEungGyun, ShinDong Ryeol (2021). Ryansql: Recursively applying sketch-based slot fillings for complex text-to-SQL in cross-domain databases. *Computational Linguistics*, *47*(2), 309–332.

Dong, Xuemei, Zhang, Chao, Ge, Yuhang, Mao, Yuren, Gao, Yunjun, Lin, Jinshu, Lou, Dongfang (2023). C3: Zero-shot Text-to-SQL with ChatGPT. https://arxiv.org/abs/2307.07306.

GaoDawei, WangHaibin, LiYaliang, SunXiuyu, QianYichen, DingBolin, ZhouJingren (2023a). Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. arXiv preprint https://arxiv.org/abs/2308.15363.

GaoYichen, XiongYiming, GaoXiaoyu, JiaKai, PanJing, BiYifan, DaiYang, SunJie, WangHaibin (2023b). Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv preprint https://arxiv.org/abs/2312.10997.

GLMTeam and ZengAohan and XuBin and WangBowen and ZhangChenhui and YinDa and ZhangDan (2024). ChatGLM: A family of large language models from GLM-130B to GLM-4 All Tools. arXiv preprint https://arxiv.org/abs/2406.12793.

GuoChunxi and TianZhiliang and TangJintao and LiShasha and WangTing (2025). Multi-pattern retrieval-augmented framework for Text-to-SQL with Poincaré-skeleton retrieval and meta-instruction reasoning. *Information Processing & Management*, *62*(3), 103978.

HePeng and MaoYu and ChakrabartiKaustubh and ChenWei (2019). X-SQL: Reinforce schema representation with context. arXiv preprint https://arxiv.org/abs/1908.08113.

HuangJie and ChangKevin Chen-Chuan (2022). Towards reasoning in large language models: A survey. https://arxiv.org/abs/2212.10403.

HuiBinyuan and YangJian and CuiZeyu and YangJiaxi and LiuDayiheng and ZhangLei and LiuTianyu and ZhangJia and YuBowen and LuKun (2024). Qwen2.5-Coder Technical Report. arXiv preprint https://arxiv.org/abs/2409.12186.

KhotTushar and TrivediHarsh and FinlaysonMatthew and FuYao and RichardsonKyle and ClarkPeter and SabharwalAshish (2022). Decomposed Prompting: A Modular Approach for Solving Complex Tasks. arXiv preprint https://arxiv.org/abs/2210.02406.

LiHaoyang and ZhangJing and LiuHanbing and FanJu and ZhangXiaokang and ZhuJun and WeiRenjie and PanHongyan and LiCuiping and ChenHong (2024a). CODES: Towards building open-source language models for Text-to-SQL. *Proceedings of the ACM on Management of Data*, *2*(3), 1–28.

LiJinyang, HuiBinyuan, QuGe, YangJiaxi, LiBinhua, LiBowen, WangBailin, QinBoxin, GengRui,& HuoNing (2024b). 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.

Lin, Kevin, Bogin, Ben, Neumann, Max, Berant, Jonathan, & Gardner, Matt (2019). Grammar-Based Neural Text-to-SQL Generation. https://arxiv.org/abs/1905.13326.

Liu, Xiping, & Tan, Zhao (2023). Divide and Prompt: Chain of Thought Prompting for Text-to-SQL. https://arxiv.org/abs/2304.11556.

Liu, Xiping, & Tan, Zhao (2024). Epi-SQL: Enhancing Text-to-SQL Translation with Error-Prevention Instructions. https://arxiv.org/abs/2404.14453.

Lyu, Qin, Chakrabarti, Kaushik, Hathi, Shobhit, Kundu, Souvik, Zhang, Jianwen, & Chen, Zheng (2020). Hybrid Ranking Network for Text-to-SQL. https://arxiv.org/abs/2008.04759.

Plaat, Aske, Wong, Annie, Verberne, Suzan, Broekens, Joost, van Stein, Niki, & Back, Thomas (2024). Reasoning with Large Language Models, a Survey. https://arxiv.org/abs/2407.11511.

Pourreza, Mohammadreza, & Rafiei, Davood (2024). DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. *Advances in Neural Information Processing Systems*, *36*, 36339–36348.

Puterman, Martin L. (1990). Markov Decision Processes. *Handbooks in Operations Research and Management Science*, *2*, 331–434.

Rajkumar, Nitarshan, Li, Raymond, & Bahdanau, Dzmitry (2022). Evaluating the Text-to-SQL Capabilities of Large Language Models. https://arxiv.org/abs/2204.00498.

Roziere, Benoit, Gehring, Jonas, Gloeckle, Felix, Sootla, Sander, Gat, Itay, Tan, Xin Eric, Adi, Yossi, Liu, Jian, Sauvestre, Romain, Remez, Tom et al. (2023). Code Llama: Open Foundation Models for Code. https://arxiv.org/abs/2308.12950.

Scholak, Torsten, Schucher, Nathan, Bahdanau, Dzmitry (2021). PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. https://arxiv.org/abs/2109.05093.

Setlur, Anirudh, Nagpal, Chirag, Fisch, Adam, Geng, Xinyi, Eisenstein, Jacob, Agarwal, Rohan, Agarwal, Ananya, Berant, Jonathan, & Kumar, Abhinav (2024). Rewarding Progress: Scaling Automated Process Verifiers for LLM Reasoning. https://arxiv.org/abs/2410.08146.

Shi, Lixin, Tang, Zheng, Zhang, Ning, Zhang, Xiaojun, & Yang, Zhi (2024). A Survey on Employing Large Language Models for Text-to-SQL Tasks. https://arxiv.org/abs/2407.15186.

Sun, Rui, Arik, Somshubra Ö., Muzio, Andrea, Miculicich, Leonardo, Gundabathula, Srini, Yin, Peng, Dai, Hao, Nakhost, Hamed, Sinha, Rishabh, & Wang, Zhen (2023). SQL-PaLM: Improved Large Language Model Adaptation for Text-to-SQL (Extended). https://arxiv.org/abs/2306.00739.

Tai, Chia-Yu, Chen, Zhi, Zhang, Tian, Deng, Xiang, & Sun, Hao (2023). Exploring Chain-of-Thought Style Prompting for Text-to-SQL. https://arxiv.org/abs/2305.14215.

Tan, Zhao, Liu, Xiping, Shu, Qi, Li, Xiaoyu, Wan, Cheng, Liu, Daming, Wan, Qiang, & Liao, Guodong (2024). Enhancing Text-to-SQL Capabilities of Large Language Models Through Tailored Promptings. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)* (pp. 6091–6109).

Wang, Bo, Ren, Cheng, Yang, Jia, Liang, Xiang, Bai, Jie, Zhang, Qi-Wei, Yan, Zhen, & Li, Zhi (2023). MAC-SQL: Multi-Agent Collaboration for Text-to-SQL. https://arxiv.org/abs/2312.11242.

Wei, Jason, Wang, Xuezhi, Schuurmans, Dale, Bosma, Maarten, Xia, Fei, Chi, Ed, Le, Quoc V.,& Zhou, Denny (2022). Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *Advances in Neural Information Processing Systems*, *35*, 24824–24837.

Wu, Tianyi, Yuan, Weijia, Golovneva, Olga, Xu, Jie, Tian, Yu, Jiao, Jian, Weston, Jason, & Sukhbaatar, Sainbay (2024). Meta-Rewarding Language Models: Self-Improving Alignment with LLM-as-a-Meta-Judge. https://arxiv.org/abs/2407.19594.

Xia, Haoran, Jiang, Fei, Deng, Neng, Wang, Cheng, Zhao, Guang, Mihalcea, Rada, & Zhang, Yujin (2024). "This Is My SQL, Are You With Me?": A Consensus-Based Multi-Agent System for Text-to-SQL Tasks. https://arxiv.org/abs/2402.14851.

Xie, Yifan, Jin, Xiaoyang, Xie, Ting, Lin, Ming, Chen, Liang, Yu, Chen, Cheng, Lei, Zhuo, Cheng, Hu, Bo, & Li, Zhe (2024). Decomposition for Enhancing Attention: Improving LLM-Based Text-to-SQL through Workflow Paradigm. https://arxiv.org/abs/2402.10671.

Xu, Fei-Fei, Alon, Uri, Neubig, Graham, & Hellendoorn, Vincent J. (2022). A Systematic Evaluation of Large Language Models of Code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming* (pp. 1–10).

Xu, Xiping, Liu, Chang, & Song, Dawn (2017). SQLNet: Generating Structured Queries from Natural Language without Reinforcement Learning. https://arxiv.org/abs/1711.04436.

Yu, Tao, Yasunaga, Michihiro, Yang, Kai, Zhang, Rui, Wang, Dong, Li, Zhiguo, & Radev, Dragomir (2018a). SyntaxSQLNet: Syntax Tree Networks for Complex and Cross-Domain Text-to-SQL Task. https://arxiv.org/abs/1810.05237.

Yu, Tao, Zhang, Rui, Yang, Kai, Yasunaga, Michihiro, Wang, Dongxu, Li, Zifan, Ma, James, Others (2018b). Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. https://arxiv.org/abs/1809.08887.

Zhang, H., Cao, R., Chen, L., Xu, H., & Yu, K. (2023). Act-SQL: In-Context Learning for Text-to-SQL with Automatically-Generated Chain-of-Thought. https://arxiv.org/abs/2310.17342.

Zhang, Hanchong, Cao, Ruisheng, Xu, Hongshen, Chen, Lu, & Yu, Kai (2024). CoE-SQL: In-Context Learning for Multi-Turn Text-to-SQL with Chain-of-Editions. https://arxiv.org/abs/2405.02712.

Zhang, Zhenru, Zheng, Chujie, Wu, Yangzhen, Zhang, Beichen, Lin, Runji, Yu, Bowen, Liu, Dayiheng, Zhou, Jingren, & Lin, Junyang (2025). The Lessons of Developing Process Reward Models in Mathematical Reasoning. https://arxiv.org/abs/2501.07301.

Zhong, Rui, Yu, Tao, & Klein, Dan (2020). Semantic Evaluation for Text-to-SQL with Distilled Test Suites. https://arxiv.org/abs/2010.02840.

Zhong, Victor, Xiong, Christopher, & Socher, Richard (2017). Seq2SQL: Generating Structured Queries from Natural Language Using Reinforcement Learning. https://arxiv.org/abs/1709.00103.