https://doi.org/10.1007/s11704-026-60238-0

RESEARCH ARTICLE

SF-SQL: a stage-free text-to-SQL framework enhanced by reinforcement learning

Yang SONG[1], Shurui KOU[1], Shuyuan LI[2]✉, Zimu ZHOU[2], Qian TAO[1], Yongxin TONG[1]✉

1. State Key Laboratory of Complex & Critical Software Environment, Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing, School of Computer Science and Engineering, Beihang University, Beijing 100191, China
2. Department of Data Science, City University of Hong Kong, Hong Kong 999077, China

Received February 4, 2026; accepted May 21, 2026

E-mail: shuyuan.li@cityu.edu.hk; yxtong@buaa.edu.cn

Abstract

Text-to-SQL, which translates natural language into SQL queries, has attracted significant attention in the database community. Recent advances leverage large language models (LLMs) to generate syntactically and semantically correct SQL. However, these approaches typically rely on prompt engineering and rigid multi-stage pipelines, making them sensitive to prompt variations and limiting their generalization across SQL dialects. To address these limitations, we propose SF-SQL, a Stage-Free Text-to-SQL framework enhanced by reinforcement learning. Instead of following a fixed pipeline, SF-SQL formulates SQL generation as a dynamic decision-making process, enabling the LLM to reason about the next action based on the context and execution feedback. To facilitate effective training, we develop a trajectory synthesis strategy that efficiently generates diverse and high-quality reasoning-action sequences. Moreover, SF-SQL proposes a reward scheme that integrates both final and intermediate query-aware signals without incurring additional inference overhead. Our experiments show that SF-SQL achieves 88.73% execution accuracy on Spider and 69.23% on BIRD, demonstrating strong generalization across five types of database dialects.

Keywords

Text-to-SQL; large language model; reinforcement learning; trajectory synthesis

1 Introduction

The task of Text-to-SQL focuses on converting natural language questions into SQL queries, enabling users to access structured data without knowledge of query languages. This capability underpins various applications such as interactive data analytics, intelligent database assistants, and automated customer support [1]. However, Text-to-SQL is challenging due to the need to interpret ambiguous natural language inputs, reason over complex database schemas, and generate SQL queries that are syntactically correct and semantically accurate.

Early Text-to-SQL methods rely on hand-crafted rules and templates [2–5], and later shift to pre-trained language models (PLMs) [6–11] such as BERT [6], TaBert [7]. Recent advancements have increasingly adopted large language models (LLMs) [12–14] adapted via prompt engineering [15–18] or fine-tuning [19–21]. These methods rely on carefully designed prompts or task-specific data to integrate schema-specific knowledge and guide reasoning. Yet both assume *rigid*, *multi-stage* pipelines, limiting their generalization. For instance, existing works like MCS-SQL [17] enforce a pipeline consisting of sequential stages like schema linking → SQL generation → selection, as shown in Fig. 1. This example highlights the rigidity of predefined pipelines, where errors in early stages propagate forward without the ability to backtrack. If an initial schema linking stage misses crucial context, the SQL generated in the second stage can be syntactically valid but semantically flawed. The final selection stage cannot correct this issue because it is limited to choosing among generated candidates, and cannot trigger a revision of the earlier schema linking stage. Addressing such errors requires manual adjustments and additional

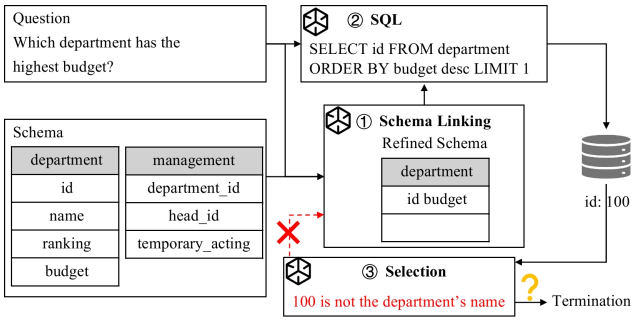

Fig. 1 Illustration of error propagation in a rigid multi-stage pipeline for Text-to-SQL generation

control logic, significantly complicating pipeline management.

To overcome these limitations, we propose SF-SQL, a *stage-free* Text-to-SQL framework enhanced by reinforcement learning. As illustrated in Fig. 2, previous LLM-based schemes decompose Text-to-SQL into a *fixed*, *multi-stage* pipeline, where each *stage* comprises a set of sequential *actions*. Specifically, traditional LLM-based methods decompose Text-to-SQL into a *fixed*, *multi-stage* pipeline: i.e., `retrieval`, `generation`, `correction`, where each *stage* comprises a set of sequential *actions*. For example, in `retrieval`, the LLM extracts relevant keywords for similarity search and links them to tables and columns. In `generation`, the LLM produces candidate SQL queries guided by in-context examples or by decomposed logical forms. In `correction`, the LLM revises the SQL with execution feedback or auxiliary models. These stages are carried out in a strict sequence and often require manual design, making the pipeline difficult to optimize. In contrast, we eliminate such stages and instead let the LLM reason the next action based on the current context, thus allowing the LLM to revise earlier decisions when necessary. Rather than reducing the reasoning space, the stage-free framework provides greater flexibility, where the model can dynamically choose among multiple actions according to the current context and environmental feedback.

However, such flexibility comes with two challenges. *(i)* The large action space in stage-free generation makes it difficult to collect high-quality reasoning-action trajectories for effective reinforcement training. *(ii)* The reward design must reflect the quality of not only the final result but also the intermediate reasoning process, all within the limited context window of LLMs. We tackle these challenges with a reinforcement learning framework that improves data reliability and optimization stability by integrating *trajectory synthesis* for consistent reasoning–action trajectories and *fine-grained rewards* derived from environmental feedback.

Our main contributions and results are as follows:

- We propose SF-SQL, a stage-free Text-to-SQL framework enhanced by reinforcement learning, which optimizes the policy of the LLM by leveraging environmental feedback, while mitigating training bias arising from flawed labels via consistency-validated supervision.
- We design a trajectory synthesis mechanism that prunes uninformative rollouts and retains high-quality reasoning-action trajectories, guiding policy learning under an action space.
- We introduce a novel reward design that integrates final outcomes with intermediate signals, without incurring additional inference overhead during reward computation.
- We conduct extensive evaluations across multiple benchmarks. SF-SQL achieves 69.23% execution accuracy on the BIRD and 88.73% on the Spider. Moreover, SF-SQL consistently improves performance in cross-dialect evaluations, yielding substantial gains on SQLite, MySQL, PostgreSQL, Oracle, and SQL Server. These results demonstrate the effectiveness and generalization capability of our approach.

2 Preliminaries

This section provides the formal definitions and essential concepts underlying the Text-to-SQL tasks.

2.1 Text-to-SQL

Problem definition. Given a relational database D, a natural language question Q, and optional external information E, Text-to-SQL generates an executable SQL query Y such that the execution of Y on D yields the correct answer to Q [22].

Formally, the Text-to-SQL task can be defined as:

$$Y = f(Q, D, E \mid \theta), \tag{1}$$

where $f(\cdot \mid \theta)$ is a model with learnable parameters θ. The database D provides metadata such as table names, column names, and foreign key constraints, as well as execution support for SQL queries. The external information E may include context such as column descriptions or explanatory notes that help resolve ambiguities in the question Q.

2.2 LLM-based Text-to-SQL

LLMs, such as GPT [12], LLaMA [13], and Qwen [14], have shown capabilities in understanding and generating natural language. OpenAI's o1 [23] introduced inference-time scaling by increasing the Chain-of-Thought reasoning process, which significantly boosts performance on complex reasoning tasks such as mathematics and code generation. LLMs can be adapted to specialized tasks, e.g., Text-to-SQL, through *prompt engineering* [15,16,24] or *fine-tuning* [19,21].

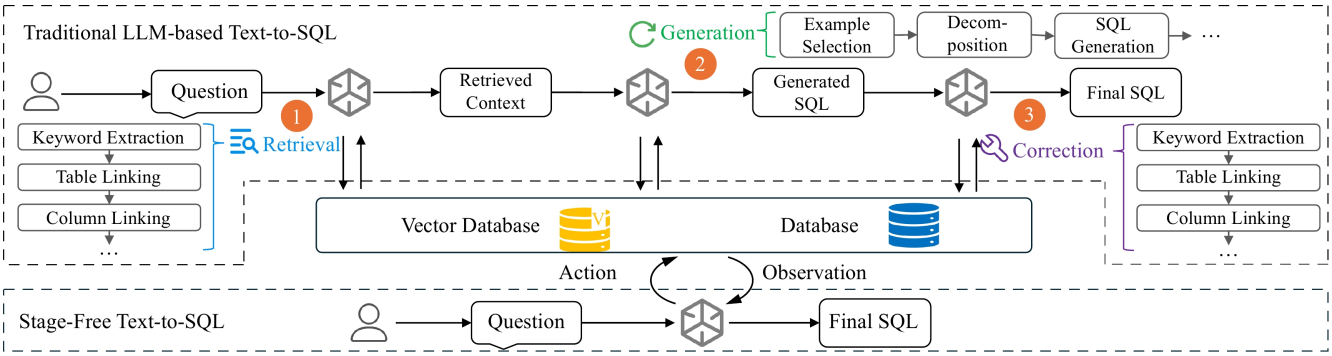

Fig. 2 Comparison between traditional LLM-based Text-to-SQL and Stage-Free Text-to-SQL

- Prompt engineering relies on the in-context learning abilities of LLMs. It guides LLMs in SQL generation with carefully crafted prompts and predefined pipelines (e.g., schema linking → SQL generation → SQL correction).
- Fine-tuning enables the LLM to acquire reasoning strategies tailored to the Text-to-SQL from training data, thus reducing its reliance on sophisticated prompt designs.

We adopt the fine-tuning approach because complex intermediate steps in Text-to-SQL generation cannot be explicitly controlled through prompt engineering alone, and it is impractical to enumerate all possible cases via handcrafted prompts [25]. In contrast, fine-tuning enables LLMs to implicitly learn these steps from data [26].

2.3 Database execution feedback

When a SQL query is executed, the database returns an error message or a result set. Such feedback offers a direct signal to refine SQL translation, hence improving the Text-to-SQL model. If the query contains syntax or semantic errors, the database indicates the cause (e.g., invalid column names or type mismatches). If the query executes successfully, the result can be compared against the question to detect logical inconsistencies or missing constraints. For example, if the result contains null values or returns more rows than expected, the model may revise the query by adding filters (e.g., a `WHERE` clause) or a `LIMIT` clause. Prior studies [15,18,27] have utilized database execution feedback to improve Text-to-SQL. However, they leverage it for post-processing or inference-time selection, rather than model fine-tuning.

2.4 Reasoning-action trajectory

Given a question q and a policy π_θ, the model interacts with the environment through a sequence of reasoning–action pairs. At each step t, the policy performs an implicit reasoning process h_t that explains the rationale for the next decision, and generates an action $a_t \in \mathcal{A}$ conditioned on the current context:

$$\rho_t = (h_t, a_t), \quad \rho_t \sim \pi_\theta(\cdot \mid q, s_t, o_t), \tag{2}$$

where $s_t \in \mathcal{S}$ denotes the internal context (e.g., partial query, intermediate feedback), and $o_t \in \mathcal{O}$ represents the observation obtained from environment feedback. The environment dynamics are characterized by a transition function:

$$\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}, \quad s_{t+1} = \mathcal{T}(s_t, a_t), \tag{3}$$

and an observation function:

$$\mathcal{E} : \mathcal{S} \times \mathcal{A} \to \mathcal{O}, \quad o_{t+1} = \mathcal{E}(s_t, a_t). \tag{4}$$

A reasoning–action trajectory is defined as the sequence:

$$\tau = \{\rho_1, \rho_2, \ldots, \rho_T\} = \{(h_1, a_1), \ldots, (h_T, a_T)\}, \tag{5}$$

which records the iterative reasoning and decision-making process of the policy π_θ. The reasoning states h_t represent implicit cognitive steps that guide subsequent actions, while actions a_t reflect explicit decisions interacting with the environment.

3 Method

This section presents SF-SQL, a reinforcement learning based framework to eliminate predefined pipelines and enable *stage-free* SQL generation from natural language queries.

3.1 SF-SQL overview

SF-SQL discards predefined pipelines, allowing the LLM to choose actions dynamically based on the context during inference and database execution feedback. We enable this *flexible*, *stage-free* generation through a reinforcement learning based fine-tuning strategy, supported by a consistency validation mechanism that mitigates the effect of noisy annotations in training data and ensures reliable supervision. Specifically, we extend Group Relative Policy Optimization (GRPO) [29] for stage-free Text-to-SQL, which has shown effectiveness in strengthening LLM reasoning capabilities in tasks like mathematical reasoning and code generation [28].

We define a set of actions inspired by prior studies [15,17,18], without enforcing a strict execution order. Furthermore, we utilize environmental feedback to design rewards and generate high-quality reasoning-action trajectories. Fig. 3 illustrates the overall architecture of SF-SQL.

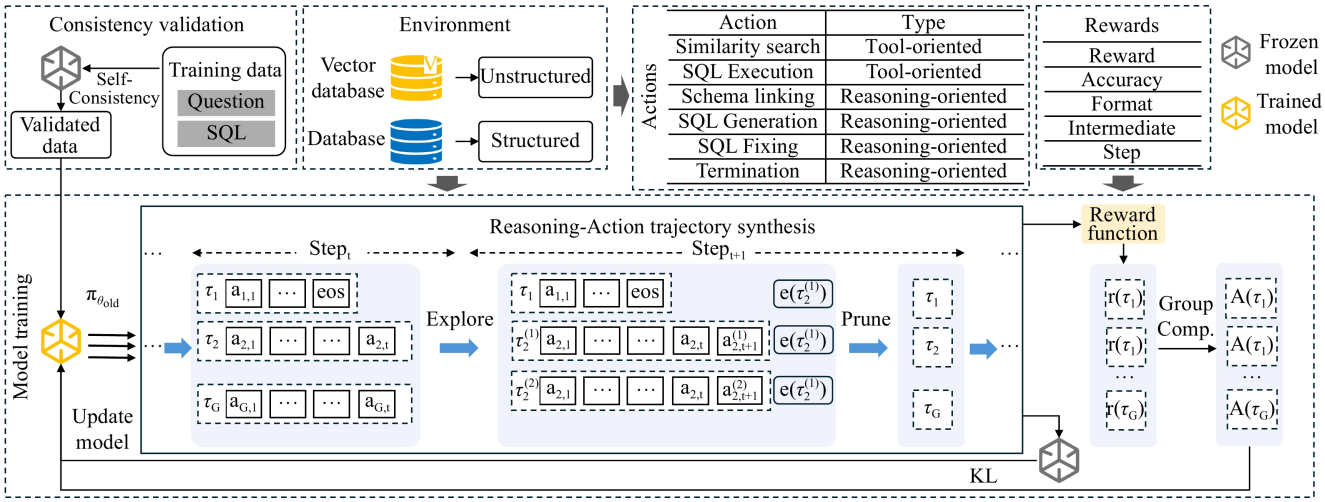

Fig. 3 Overview of the SF-SQL framework for Text-to-SQL

FCS-260238-YS 2026-08-17 09:13:23

GRPO-based fine-tuning. Reinforcement learning such as Proximal Policy Optimization (PPO) [30] and Direct Preference Optimization (DPO) [31] has recently been adopted to fine-tune LLMs to improve their task-specific capabilities. In particular, Group Relative Policy Optimization (GRPO) [29] has emerged as a promising approach, reducing memory overhead while achieving competitive performance by optimizing trajectory groups rather than individual actions.

Traditional GRPO is primarily designed for single-round inference, where multiple completions are generated per prompt and treated as independent optimization instances. In contrast, stage-free Text-to-SQL requires sequential reasoning, in which the validity of later actions depends on earlier reasoning steps. Group-wise optimization without modeling these dependencies can lead to inconsistent learning signals and unstable policy updates. To overcome this limitation, we extend GRPO to support reasoning–action trajectory groups. Specifically, SF-SQL introduces a reasoning-action trajectory synthesis method that generates multiple rollouts, and a trajectory pruning strategy to discard redundant or inconsistent rollouts, retaining only coherent trajectories for optimization. This extension enables effective and stable policy learning over temporally dependent reasoning sequences.

The detailed training procedure of SF-SQL is shown in Algorithm 1. Given a question q, the old policy $\pi_{\theta_{old}}$ produces a group of reasoning-action trajectories $\{\tau_1, \tau_2, \ldots, \tau_G\}$. These trajectories are then used to guide the policy update by maximizing the objective defined below:

$$\begin{aligned}\mathcal{J}_{\text{GRPO}}(\theta) = &\mathbb{E}_{q\sim P(Q)}\Bigg[\frac{1}{G}\sum_{i=1}^{G}\min\Bigg(\frac{\pi_\theta(\tau_i \mid q)}{\pi_{\theta_{\text{old}}}(\tau_i \mid q)}A_i,\\ &\text{clip}\Bigg(\frac{\pi_\theta(\tau_i \mid q)}{\pi_{\theta_{\text{old}}}(\tau_i \mid q)}, 1-\epsilon,\ 1+\epsilon\Bigg)A_i\Bigg)\\ &-\beta D_{\text{KL}}[\pi_\theta \| \pi_{\text{ref}}]\Bigg].\end{aligned} \tag{6}$$

where π_{ref} is the reference policy, G is a hyperparameter denoting the number of reasoning-action trajectories sampled for each question. ϵ and β are hyperparameters that control the update step and the divergence regularization, respectively. The advantage A_i for normalizing trajectory rewards within each group is defined as follows:

$$A_i = \frac{r_i - mean(\{r_1, r_2, \ldots, r_G\})}{std(\{r_1, r_2, \ldots, r_G\})}. \tag{7}$$

The KL-divergence regularization term between the current policy and the reference policy π_{ref} is computed as:

$$D_{KL}[\pi_\theta \| \pi_{ref}] = \frac{\pi_{ref}(\tau_i|q)}{\pi_\theta(\tau_i|q)} - \log\frac{\pi_{ref}(\tau_i|q)}{\pi_\theta(\tau_i|q)} - 1. \tag{8}$$

Challenges. Two challenges arise when realizing such a reinforcement learning method for the stage-free Text-to-SQL.

- *High-quality trajectory synthesis.* A trajectory consists of a sequence of *reasoning-action* pairs produced by the LLM, and its quality is critical for the effectiveness of reinforcement learning [32–34]. In our context, GRPO requires multiple trajectories per query to learn relative action preferences, and insufficient differentiation among trajectories' rewards can degrade policy optimization. Therefore, it is important to efficiently synthesize meaningful trajectories for LLM fine-tuning.
- *Fine-grained reward design.* Reinforcement learning is sensitive to the quality of reward signals. Unlike end-to-end supervised training, trajectory-based generation necessitates assigning reward signals not only at the end (final SQL correctness) but also throughout intermediate steps to guide LLM reasoning. Furthermore, each trajectory contains multiple *reasoning* and *action segments*. Accumulating them over long trajectories may exceed the LLM's context window. Thus, reward design must balance correctness and conciseness to ensure effective learning signals without context overflow.

Algorithm 1: Training Process of SF-SQL

Input: Consistent training set $\mathcal{R} = \{(Q_i, Y_i)\}_{i=1}^{N}$; Reference model π_{ref}; Policy model π_θ; Vector database $\mathcal{V}$; Database engine $\mathcal{D}$; Trajectory expansion rate c.

Output: Optimized policy π_θ.

foreach *epoch* **do**
  `// Trajectory synthesis`
  foreach *mini-batch* $\{(Q_i, Y_i)\}_{i=1}^{M} \subset \mathcal{R}$ **do**
    `// Generate` G `trajectories`
    Initialize buffer $\mathcal{B} \leftarrow \emptyset$;
    for $k \leftarrow 1$ **to** G **do**
      Initialize trajectory $\tau_k^i \leftarrow []$;
      Generate action $a_{k,1}^i$ using π_θ;
      Append $a_{k,1}^i$ to τ_k^i;
      Store τ_k^i into buffer $\mathcal{B}$;
    while *the number of terminated trajectories in* $\mathcal{B}$ *is less than* G **do**
      for $k = 1$ **to** G **do**
        $t = |\tau_k^i|$;
        if $a_{k,t}^i$ *not Termination* **then**
          Remove τ_k^i from $\mathcal{B}$;
          for $j \leftarrow 1$ **to** c **do**
            Generate action $a_{k,t+1}^{i\ (j)}$ using π_θ;
            Store new trajectory $([\tau_k^i, a_{k,t+1}^{i\ (j)}])$ in $\mathcal{B}$;
      Prune $\mathcal{B}$ using Algorithm 2;
    `// Compute advantage`
    foreach $\tau_k^i \in \mathcal{B}$ **do**
      Compute score $r(\tau_k^i)$ using Eq. (10);
      $A_i^k \leftarrow \dfrac{r(\tau_k^i) - mean(\{r(\tau_1^i), \ldots, r(\tau_G^i)\})}{std(\{r(\tau_1^i), \ldots, r(\tau_G^i)\})}$;
    `// Update policy`
    Compute loss $\mathcal{J}_{\text{GRPO}}(\theta)$ for π_θ;
    Update $\theta \leftarrow \theta - \mu \nabla_\theta(-\mathcal{J}_{\text{GRPO}}(\theta))$;
return π_θ;

3.2 Action design

Prior studies [15,18,27] typically decompose the Text-to-SQL task into predefined stages with fixed sequences of actions. In contrast, SF-SQL abstracts these stages into a set of actions that the model can freely select at each reasoning step. We define these actions as follows:

1. *Similarity search*: Search external information (e.g., database descriptions) via vector-based similarity search.
2. *SQL execution*: Execute a candidate SQL query on the database and return the execution result, offering feedback for verifying correctness and guiding further reasoning.
3. *Schema linking*: Identify and align entities from the input question with relevant schema elements (tables, columns), utilizing previous retrieval results, SQL execution results, and contextual cues.
4. *SQL generation*: Generate a candidate SQL query based on the reasoning context, incorporating the question's semantics and the linked schema information.
5. *SQL fixing*: Modify the SQL query in response to execution feedback, addressing syntax errors or semantic mismatches with the input question, such as incorrect conditions or missing clauses.
6. *Termination*: Signal the end of the inference when the LLM concludes the current SQL query sufficiently answers the question. The final SQL query generated is then returned.

Our action design is guided by the intuitive steps that humans take when solving Text-to-SQL problems. These actions constitute a minimal and effective closed-loop action set that supports retrieval, grounding, generation, verification, and refinement throughout the stage-free reasoning process. Broadly, the actions fall into two categories:

1. *Tool-oriented actions* include *Similarity Search* and *SQL Execution*. These actions serve as interfaces for retrieving similarity-based external information and database execution feedback. Prior works [27,35] retrieve cell values directly using Locality-Sensitive Hashing (LSH) [36] to support value-aware linking. However, we avoid this approach due to practical considerations. In real-world applications, database values are frequently updated, and maintaining an external index for values introduces additional synchronization costs and complexity.
2. *Reasoning-oriented actions* include *Schema Linking*, *SQL Generation*, *SQL Fixing*, and *Termination*. These actions reflect the core reasoning process that humans perform when translating natural language questions into SQL, from grounding question semantics in the schema to generating and refining queries based on intermediate feedback. Prior prompt-based methods [15,17,24] introduce other actions such as *Example Selection* to retrieve few-shot examples from the training set. However, such actions significantly increase context length and are often incompatible with fine-tuning. Therefore, we deliberately avoid introducing this type of action.

The action set is designed to be both operationally feasible and sufficiently expressive to support stage-free Text-to-SQL generation. Future work may explore incorporating additional actions to further enhance reasoning flexibility. Such extensions, while potentially beneficial, would likely introduce more intricate reasoning dynamics that call for stronger planning and reasoning capabilities from the LLM.

Action environment setup. To enable stage-free Text-to-SQL, SF-SQL operates within a unified environment that integrates both structured and unstructured information sources. This environment provides tools and execution feedback, enabling the system to perform retrieval, grounding, and verification within a single interactive process. During execution, the environment serves as an intermediary between the LLM and the underlying data sources. Each operation, including retrieval, schema linking, and query execution, is carried out through a unified interface. Environmental feedback, such as database execution results, error messages, and similarity search outputs, is dynamically incorporated into subsequent reasoning steps, forming a closed-loop interaction process that allows continuous refinement.

The environment is modularly designed with dedicated components for schema management, query execution, and external knowledge retrieval. *Structured information* is instantiated as JDBC-accessible relational databases, which support both metadata extraction and query execution. *Unstructured information* is vectorized and stored in a vector database, enabling semantic retrieval of relevant external context to improve generation accuracy. Here, the vector database is introduced solely to index the auxiliary external information E, whereas the relational database D is reserved for structured data access and SQL execution. We index only metadata rather than building BM25 [37] or LSH [36] indexes over raw database values, as this approach is impractical in large-scale real-world systems. To ensure coherent reasoning, the environment maintains contextual states that record retrieved entities, intermediate results, and previous reasoning traces. These states are continuously updated and validated to prevent inconsistencies caused by outdated or conflicting information.

3.3 Reasoning-action trajectory synthesis

In the SF-SQL framework, the actions are specified in the prompt, allowing the LLM to autonomously select the next action based on the current reasoning context. This design avoids explicitly encoding the entire pipeline and enables the model to flexibly explore reasoning-action trajectories guided by execution feedback. Nevertheless, the large exploration space of Text-to-SQL introduces challenges in generating high-quality reasoning trajectories. This complexity arises not only from the wide range of available actions the LLM can choose from at each step, but also from the accompanying reasoning process that determines how and when these actions should be applied. The combination of diverse action types and multi-step reasoning leads to a highly combinatorial

decision space, making effective trajectory synthesis and policy learning more difficult.

As shown in Eq. (7), GRPO relies on relative preference comparisons within trajectory groups to update the policy. A larger group size G generally provides more robust estimates of the group-wise reward statistics used for advantage normalization, but it also incurs higher rollout cost and training resource consumption. In practice, the effectiveness of GRPO depends not only on the group size, but also on the quality of the sampled trajectories within each group. If the trajectories within a group are uniformly poor, it becomes difficult for the model to distinguish which reasoning path is better, thereby weakening the effectiveness of relative policy learning. Therefore, pruning low-quality or redundant trajectories ensures that the retained examples provide strong training signals and allow the model to learn meaningful distinctions in reasoning quality. Furthermore, the step-by-step nature of the reasoning makes inference computationally intensive, as each step involves sampling. To mitigate these issues, we introduce a pruning strategy that prunes useless trajectories at each reasoning step. This not only reduces inference latency but also improves the quality of the resulting reasoning data.

Consistency validation. High-quality reasoning–action trajectories depend critically on the correctness of the supervision used for synthesis. However, empirical studies [38] reveal that up to 49% of examples in certain domains of the BIRD dataset [22] contain annotation errors. If such erroneous annotations are used for trajectory sampling, the synthesized reasoning paths will inevitably reflect these inconsistencies, misleading the model to learn incorrect reasoning patterns and degrading overall policy learning. As illustrated in Fig. 4, an example with the question "list all movies released in 1945 in ranked order" is incorrectly annotated with a SQL query containing a `LIMIT` clause, which returns only a single result. Such inconsistencies directly corrupt the trajectories derived from them.

To mitigate this issue, we introduce a *consistency validation* module, inspired by [39], before trajectory synthesis to ensure alignment between the question and its reference SQL. For each sample, the LLM independently evaluates the logical consistency between the question and the SQL multiple times according to a set of checking rules, and the final decision is made by majority voting. The detailed prompt for consistency validation is provided in our released code. Samples identified as inconsistent are filtered out, ensuring that the subsequent trajectory synthesis is performed on a reliable training set.

```
# Question:
Name movie titles released in year 1945.
Sort the listing by the descending order of movie popularity.
# Gold SQL
SELECT movie_title FROM movies
WHERE
  movie_release_year = 1945
  ORDER BY movie_popularity DESC
LIMIT 1
```

Fig. 4 An error case in the BIRD dataset

Prompt design. In our framework, we employ a prompt composed of three main components: base instructions, format instructions, and task-specific information. The base instructions include the definition of the LLM's role and a description of the task. To facilitate data extraction, the format instructions specify constraints on the output format. The task-specific information includes the input for each task, such as the question, schema, and external information. The overall prompt for trajectory synthesis is provided in Fig. 5.

Trajectory pruning. Constructing informative groups of trajectories plays a vital role in GRPO training. Given that the framework includes six distinct types of actions, each potentially associated with an additional reasoning process, the decision space for the LLM is substantially large. To prevent the model from arbitrarily sampling actions during reasoning, it is necessary to evaluate each action step and prune trajectories. As shown in Eq. (7), to make GRPO training effective, the reward distribution across trajectories should exhibit sufficient variance. If all rewards are similar, the resulting advantage values A_i approach zero, weakening the learning signal and impeding policy optimization. Therefore, it is crucial to ensure that the rewards of trajectories within each group maintain a certain degree of diversity. Although pruning operates on partial trajectories based on intermediate signals, the quality of these intermediate steps is strongly correlated with the final task reward. This pruning process ensures that the retained reasoning–action trajectories deliver informative and reliable supervision, facilitating effective GRPO policy learning.

For each reasoning–action step a_i in a partial trajectory $\hat{\tau} = \{\rho_1, \rho_2, \ldots, \rho_T\}$, we design a set of rule-based scoring functions to evaluate the utility of each action. The scoring rules for different

```
You are an expert in Text-to-SQL tasks. For each reasoning step,
you may freely choose and output one of the following actions,
each enclosed in its corresponding tag:
# Action and Format Instructions:
- `<search>...</search>`: Similarity Search...
- `<execute>...</execute>`: SQL Execution...
- `<link>...</link>`: Schema Linking...
- `<sql>...</sql>`: SQL Generation...
- `<fix>...</fix>`: SQL Fixing...
- `<answer>...</answer>`: Termination...
...
# Output Example:
Please strictly follow the output example below:
<reasoning> This is the reasoning process. </reasoning>
<search> keywords </search>
<reasoning>...</reasoning>
<execute> SELECT order_date FROM orders LIMIT 5 </execute>
...
<answer> The final answer is \boxed{SELECT ...} </answer>
# Task Information:
Question: {question}
Evidence: {evidence}
Database Schema: {schema}
```

Fig. 5 Prompt for reasoning-action trajectory synthesis

types of actions are as follows:

1. *Similarity search*: Assign a score of 1 if the retrieved metadata includes any table or column name that appears in the ground truth SQL. Otherwise, assign a score of 0.
2. *SQL execution*: Assign a score of 1 if the execution result of the current SQL query contains any keywords mentioned in the question. Otherwise, assign a score of 0.
3. *Schema linking*: Let M_{gold} denote the set of table and column names referenced in the ground truth SQL, and let M_{linked} be the set of metadata linked by the model. The score is computed as the ratio of correctly matched elements in M_{linked} to the total number of elements in M_{gold}, i.e., $\frac{|M_{gold} \cap M_{linked}|}{|M_{gold}|}$.
4. *SQL generation*: Decompose both the generated SQL and the ground truth SQL into structural clauses (e.g., `SELECT`, `WHERE`, `ORDER BY`, `JOIN ON`). The score is computed as the proportion of correctly matched clause types between two SQL queries.
5. *SQL fixing*: If the revised SQL corrects errors in the original version and aligns with the structure and semantics of the ground truth SQL, assign a score of 1. Otherwise, assign a score of 0.
6. *Termination*: Assign a score of 1 if the execution result of the final SQL query exactly matches that of the ground truth SQL. If the result is incorrect, assign a score of 0.

We aggregate the score of each action to compute the overall score for the partial reasoning–action trajectory. Specifically, given a partial trajectory $\tau = \{\rho_1, \rho_2, \ldots, \rho_t\} = \{(h_1, a_1), (h_2, a_2), \ldots, (h_t, a_t)\}$, its score is defined as the average of individual action scores:

$$e(\hat{\tau}) = \frac{1}{t}\sum_{i=1}^{t} e(\rho_i) = \frac{1}{t}\sum_{i=1}^{t} e(a_i), \tag{9}$$

where $e(a_i)$ denotes the score assigned to the i-th action a_i in the partial trajectory $\hat{\tau}$. Here $e(\rho_i)$ reduces to $e(a_i)$ because the implicit reasoning process h_i is not directly executed or verified, whereas the action a_i yields observable feedback.

To construct a candidate group of G trajectories for GRPO updates, we adopt a two-stage process: initial sampling followed by diversity-aware selection. In the first stage, we sample G reasoning–action steps for each input question from the current policy $\pi_{\theta_{old}}$. Each action is evaluated using the intermediate scoring function defined in Eq. (9). Since GRPO depends on relative preferences within trajectory groups, we select G trajectories that not only span a wide reward range but also represent varied reasoning patterns. Specifically, all candidate trajectories are sorted by their scores, and a stratified sampling strategy is applied, as detailed in Algorithm 2. This strategy partitions the score distribution into discrete intervals and uniformly samples trajectories within each, ensuring a balanced distribution of trajectory quality across the group. Importantly, we avoid including all candidate trajectories, as doing so would incur exponentially higher training costs and introduce many trajectories that contribute little to learning.

Algorithm 2: Trajectory Pruning

Input: Trajectory buffer $\mathcal{B}$; Target size G.
Output: Pruned buffer $\mathcal{B}$ of size G.

Compute each partial trajectory score $e(\cdot)$ in $\mathcal{B}$ using Eq. (9);
Partition $\mathcal{B}$ into G strata $\{\mathcal{B}_1, \ldots, \mathcal{B}_G\}$ based on $e(\cdot)$;
while $|\mathcal{B}| > G$ **do**
  $p \leftarrow |\mathcal{B}| - G$;
  for $i \leftarrow 1$ **to** p **do**
    Select an unterminated trajectory $\tau \in \mathcal{B}_i$ randomly;
    Remove τ from $\mathcal{B}_i$;
    if $\sum_{i=1}^{G} |\mathcal{B}_i| \le G$ **then break**;
$\mathcal{B} \leftarrow \bigcup_{i=1}^{G} \mathcal{B}_i$; `// Merge`
return $\mathcal{B}$;

By including trajectories with both high and low scores, this selection strategy promotes variance in trajectory rewards, which enhances the effectiveness of advantage-based optimization. Maintaining such diversity encourages the model to better differentiate between suboptimal and high-quality reasoning behaviors, thereby improving both the stability and learning efficiency of GRPO training.

Example 1. Fig. 6 presents examples of scoring rules for different action types. For Similarity Search, the retrieved metadata correctly includes `CharterNum`, which also appears in the ground truth SQL, yielding a score of 1.0. For *SQL Execution*, the generated query `" SELECT cds, AvgScrWrite FROM satscores WHERE AvgScrWrite = 499 "` returns relevant results overlapping with the query keywords, achieving a score of 1.0. In Schema Linking, three relevant elements: `schools`, `satscores`, and `schools.CharterNum` are correctly identified. However, some gold SQL elements are missing, resulting in a partial match (score 0.5). During SQL Generation, the model constructs most structural

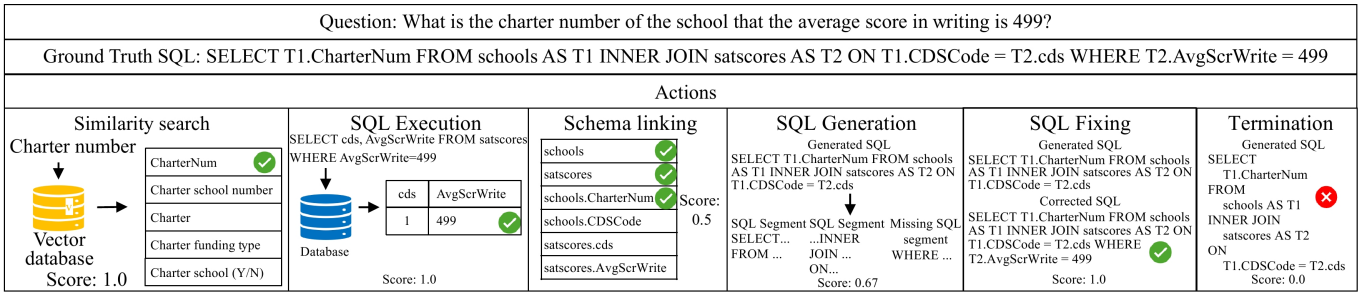

Fig. 6 Examples of scoring rules for each type of action

elements correctly (`SELECT`, `JOIN`) but omits the `WHERE` clause, leading to a reduced score of 0.67. The subsequent SQL Fixing action successfully adds the missing condition `" T2.AvgScrWrite = 499 "`, aligning the revised SQL with the ground truth and restoring a score of 1.0. Finally, in Termination, the model outputs an incomplete SQL lacking the `WHERE` clause, resulting in an incorrect final answer and a score of 0.0.

Illustration of reasoning–action trajectory synthesis. As illustrated in Fig. 7, the trajectory synthesis process constructs and maintains a fixed number of trajectory groups during GRPO training. In this example, the number of retained trajectories in the group is set to two ($G = 2$). To better preserve reward diversity, we divide all candidate trajectories into two strata based on their cumulative scores: *high-score* and *low-score* groups. At each reasoning step, the model samples multiple action candidates conditioned on the current context and continuously updates the cumulative score for each partial trajectory. Within the group, if multiple trajectories fall into the same stratum, we randomly prune one of them to ensure that the total number of retained trajectories remains equal to G. Only the retained trajectories are expanded in the next reasoning step, while pruned ones are discarded. This process continues until G trajectories reach the termination action, at which point the rollout process stops.

As shown in Fig. 7, *high-score* trajectories are retained across steps, while *low-score* ones are pruned during intermediate stages. The resulting set of trajectories achieves both diversity and consistency, forming informative groups for stable and efficient GRPO optimization.

3.4 Reward design

The design of the reward function is crucial for aligning reinforcement learning objectives with the target task. In Text-to-SQL, the model must not only generate correct SQL but also follow structured output formats and maintain efficiency within limited context windows. A naive reward, such as a simple correctness signal, fails to capture these multifaceted requirements, as it provides sparse feedback and overlooks the reasoning quality essential for reliable SQL generation. To address these limitations, we employ a set of rewards that decompose supervision into complementary signals. Specifically, the *accuracy reward* ensures that the model's outputs achieve functional correctness by directly comparing the execution results of the database, the *format reward* enforces structural consistency in reasoning–action representations, facilitating the reliable extraction of actions from the model's output, the *intermediate reward* provides feedback for each reasoning step, improving learning efficiency and guiding the model to develop coherent reasoning strategies, and the *step reward* regularizes trajectory length, balancing performance and computational efficiency.

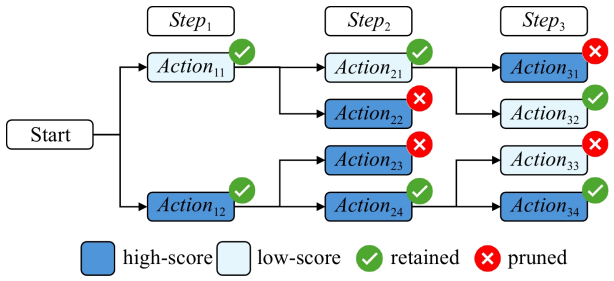

Fig. 7 Illustration of the reasoning-action trajectory synthesis and trajectory pruning

This fine-grained reward design provides transparent, interpretable, and computationally efficient supervision signals that are well suited to iterative reasoning tasks, while avoiding the cost and noise associated with model-based evaluators. The detailed designs are as follows:

1. *Accuracy reward*: It measures correctness by executing the predicted SQL on the target database and comparing the results with the ground truth SQL, and follows the same verification function $V(Y_{pred}, Y_{gt})$ described in Section 4.1.
2. *Format reward*: Although LLMs generally follow the instructions, they occasionally deviate from specified output formats [40]. To facilitate reliable extraction of intermediate reasoning steps, we introduce a format reward. Specifically, each reasoning–action step is formatted by first enclosing the natural language reasoning segment within `<reasoning>...</reasoning>` tags, followed by an action enclosed in its corresponding tag: `<search>...</search>` for similarity search, `<execute>...</execute>` for SQL execution, `<link>...</link>` for schema linking, `<sql>...</sql>` for SQL generation, `<fix>...</fix>` for SQL fixing, and `<answer>...</answer>` for termination.
3. *Intermediate reward*: To enable the model to learn effective reasoning processes, we assign reasoning scores to each action taken by the model. The scoring mechanism for the intermediate reward follows the same strategy used in trajectory pruning, providing feedback that reflects the quality and relevance of each reasoning step.
4. *Step reward*: Although SQL generation often involves relatively short reasoning processes, excessively long reasoning trajectories can incur higher inference costs and may exceed the LLM's context window, preventing effective inference. Hence, we introduce a step reward to encourage concise reasoning trajectories.

The overall reward function r is defined as follows:

$$r = \begin{cases} 1 - 0.05 \cdot step, & \text{if SQL is correct} \\ 0.1 + e(\tau), & \text{if SQL is incorrect and format is correct} \\ 0, & \text{if SQL and format are incorrect.} \end{cases} \tag{10}$$

We design a set of rule-based rewards comprising *accuracy reward*, *format reward*, *intermediate reward*, and *step reward*. The *accuracy* and *format* rewards are adopted to guide the model toward producing semantically correct SQL queries and adhering to structured output formats, respectively. In addition, we propose an *intermediate reward* to assess the coherence and relevance of the

model's reasoning process. To control inference cost and ensure compatibility with the LLM's context window, we introduce a *step reward* to discourage unnecessarily long reasoning-action sequences. The coefficient 0.05 serves as a mild linear decay factor.

4 Experiments

4.1 Experimental setup

Datasets. To assess the performance of the proposed SF-SQL framework, we conducted experiments on three well-recognized datasets: BIRD [22], Spider [41], and BIRD Mini-Dev [22].

1. BIRD: A real-world, large-scale Text-to-SQL benchmark comprising 12,751 queries over 95 large databases covering 37 professional domains. The dataset is divided into 9,428 training examples from 69 databases, 1,534 development examples from 11 databases, and a held-out test set of 1,789 examples from 15 databases. Compared to Spider, BIRD features more complex SQL constructs such as *LEFT JOIN*, *PARTITION BY*, and functions like *IIF* and *ROUND*.
2. Spider: A large-scale, complex, and cross-domain Text-to-SQL dataset consisting of 10,181 natural language questions and 5,693 unique complex SQL queries over 200 databases spanning 138 domains. It is split into 8,659 training examples from 146 databases, 1,034 development examples from 20 databases, and a held-out test set of 2,147 examples from 34 databases.
3. BIRD Mini-Dev: A subset of the BIRD development set, comprising 500 high-quality examples from 11 databases, collected based on community feedback. It is available in SQLite, MySQL, and PostgreSQL formats. We translated it into Oracle and SQL Server formats for further cross-dialect evaluation.

After consistency validation, 8,622 of the original 8,659 Spider training examples and 6,982 of the original 9,428 BIRD training examples remain.

Evaluation metrics. We adopt Execution Accuracy (EX) [41] as the evaluation metric to assess model performance. EX compares the execution result of the predicted SQL query with the ground truth on a given database. As multiple SQL queries may yield the same answer for a question, EX focuses on execution equivalence rather than syntactic similarity, providing a robust measure of semantic correctness.

To verify the correctness of the predicted SQL Y_{pred}, we compare its execution results with the ground truth SQL Y_{gt}. A function $\mathrm{Exec}(Y, D)$ is defined to return the execution results of the SQL query Y on the database D. Additionally, we define a verification function $V(Y_{pred}, Y_{gt})$ to assess the correctness of the predicted SQL $\hat{Y}$. If $\mathrm{Exec}(Y_{pred}, D) = \mathrm{Exec}(Y_{gt}, D)$, the predicted SQL correctly answers the question, and $V(Y_{pred}, Y_{gt}) = 1$, otherwise, $V(Y_{pred}, Y_{gt}) = 0$. The overall EX score is then computed as:

$$\mathrm{EX} = \frac{1}{N} \sum_{i=1}^{N} V(Y_{pred}^{(i)}, Y_{gt}^{(i)}), \tag{11}$$

where N is the total number of examples.

Baselines. We choose some state-of-the-art LLM-based methods as baselines, including (1) *Prompt-based methods*: DAIL-SQL, MAC-SQL, CHESS, Distillery, PURPLE, MCS-SQL, SuperSQL, and DIN-SQL, and (2) *Fine-tuned methods*: SFT, CodeS, and Reasoning-SQL:

1. DAIL-SQL [15]: The DAIL-SQL selects relevant examples based on question and query similarities and encodes database schema in code style to include full database schema information.
2. MAC-SQL [42]: The MAC-SQL consists of three collaborative agents. The Decomposer splits questions into sub-questions for chain-of-thought reasoning. The Selector prunes the schema to remove irrelevant information. The Refiner uses external tools to correct SQL errors.
3. CHESS [18]: The CHESS employs four specialized agents: Information Retriever, Schema Selector, Candidate Generator, and Unit Tester to tackle key challenges such as data retrieval, schema pruning, query generation, and validation.
4. Distillery [43]: An approach leveraging LLM to bypass schema linking when the schema fits within the context window, employing augmentation, selection, and correction to improve accuracy.
5. PURPLE [24]: A retrieval-based approach that fetches demonstrations with the required logical operator compositions for Text-to-SQL, guiding LLMs to produce accurate SQL translations.
6. MCS-SQL [17]: The MCS-SQL leverages multiple prompts and a multiple choice selection mechanism in three steps: schema linking, generating multiple SQL queries, and selecting the optimal one.
7. SuperSQL [44]: The SuperSQL integrates techniques such as schema linking, multi-step reasoning, and self-consistency to optimize SQL query generation.
8. DIN-SQL [16]: A decomposed in-context learning framework that sequentially applies schema linking, query classification and decomposition, intermediate representation guided SQL generation, and a self-correction module to substantially boost LLM performance on Text-to-SQL.
9. SFT: The SFT leverages a carefully designed prompt used for supervised fine-tuning. The prompt requires LLM to follow the instruction and generate SQL for the question.
10. CodeS [19]: The CodeS builds upon the open-source LLM StarCoder [45], undergoing incremental pretraining and supervised fine-tuning to ultimately achieve a Text-to-SQL domain-specific LLM with sizes of 1B, 3B, 7B, and 15B.
11. Reasoning-SQL [21]: It introduces Group Relative Policy Optimization (GRPO), incorporating partial rewards for schema linking, SQL syntax validity, n-gram similarity, and execution accuracy to enhance Text-to-SQL generation.

Experimental settings. In all of our experiments, we use the 7B version of Qwen2.5-Coder-Instruct [46] as the base model and adopt the verl framework [47] for reinforcement learning. For consistency

FCS-260238-YS 2026-08-17 09:13:23

validation, which does not require fine-tuning, we employ the Qwen2.5-Coder-32B model [46] to ensure high reliability. A pretrained embedding model, denoted as all-MiniLM-L6-v2 [48], is used to generate vector representations, which are stored in Chroma [49]. All experiments are conducted on 8 NVIDIA A100 (80GB) GPUs, and rollout generation is accelerated using vLLM [50].

For training the model with GRPO, we set the temperature to 0.6, the learning rate to 5e-6 with a warm-up ratio of 0.3%, and the batch size to 16. The model is trained for 3 epochs with the number of trajectories per group $G = 8$ and the expansion rate $c = 2$. For SFT, we use a learning rate of 5e-6, a batch size of 16, and 1 epoch.

4.2 Overall performance

The overall results on the BIRD and Spider datasets are reported in Table 1 and Table 2, respectively. On BIRD, Reasoning-SQL reports two BIRD results with different backbone models, namely Qwen2.5-7B and Qwen2.5-Coder-7B, and we include both here for completeness. SF-SQL attains 69.23% execution accuracy, surpassing CHESS and Reasoning-SQL, the latter also using GRPO for reinforcement learning fine-tuning. On Spider, SF-SQL reaches 88.73% execution accuracy, showing consistent improvements over strong baselines. In particular, it exceeds Reasoning-SQL by 10.01%. Despite being based on a relatively compact 7B-scale open-source model, SF-SQL surpasses baselines utilizing larger-scale LLMs such as GPT-4 and CodeS-15B. SF-SQL attains the second-best results on the Spider dataset, falling slightly short of MCS-SQL. This is mainly because many tables in Spider are sparsely populated or even empty, which limits the environmental feedback our framework depends on. However, on the more realistic BIRD dataset, our framework significantly outperforms MCS-SQL. This demonstrates that our method effectively unlocks greater reasoning and execution capabilities from relatively smaller-scale models. Besides, we observe the decline in performance with SFT of Qwen2.5 on both datasets. This can be attributed to the fact that the training data has already been included during the Qwen2.5-Coder's post-training phase. As a result, SFT provides no gains and can potentially hinder generalization.

Table 1 Performance comparison of different Text-to-SQL methods on BIRD dev set

Method	Model	EX/%
DAIL-SQL [15]	GPT-4	54.76
MAC-SQL [42]	GPT-4	59.39
CHESS [18]	GPT-4	68.31
Distillery [43]	GPT-4o	67.21
PURPLE [24]	GPT-4o	62.97
PURPLE+RED [24]	GPT-4o	68.12
MCS-SQL [17]	GPT-4	63.36
SuperSQL [44]	GPT-4	58.50
DIN-SQL [16]	GPT-4	55.90
CodeS [19]	CodeS-15B	58.47
Reasoning-SQL [21]	Qwen2.5-7B	68.05
Reasoning-SQL [21]	Qwen2.5-Coder-7B	66.12
Qwen2.5-Coder-7B [46]	Qwen2.5-Coder-7B	48.50
Qwen2.5-Coder-7B SFT [46]	Qwen2.5-Coder-7B	39.89
SF-SQL	**Qwen2.5-Coder-7B**	**69.23**

Table 2 Performance comparison of different Text-to-SQL methods on Spider test set

Method	Model	EX/%
DAIL-SQL [15]	GPT-4	86.60
MAC-SQL [42]	GPT-4	82.80
CHESS [18]	GPT-4	87.20
MCS-SQL [17]	GPT-4	89.60
SuperSQL [44]	GPT-4	87.00
DIN-SQL [16]	GPT-4	85.30
CodeS [19]	CodeS-15B	84.90
Reasoning-SQL [21]	Qwen2.5-Coder-7B	78.72
Qwen2.5-Coder-7B [46]	Qwen2.5-Coder-7B	74.43
Qwen2.5-Coder-7B SFT [46]	Qwen2.5-Coder-7B	72.61
SF-SQL	**Qwen2.5-Coder-7B**	**88.73**

To further examine the impact of query complexity, we report the performance across different difficulty levels on BIRD in Table 3. SF-SQL achieves competitive performance across all difficulty levels, attaining the best results on simple and challenging queries. In particular, SF-SQL reaches 61.1% on challenging queries, outperforming the strongest baseline CHESS by 1.8% and substantially surpassing Reasoning-SQL by 14.9%. These results provide further evidence that the stage-free framework is especially beneficial for more complex SQL queries, where flexible multi-step reasoning and iterative revision are more critical.

4.3 Cross-dialect performance

To evaluate the cross-dialect generalizability of SF-SQL, we conduct experiments on the BIRD Mini-Dev set across five SQL dialects:

Table 3 Performance across different difficulty levels on BIRD

Method	Simple	Moderate	Challenge	All
MAC-SQL	65.7	52.7	40.3	59.4
CHESS	73.5	**63.0**	59.3	68.3
MCS-SQL	70.4	53.1	51.4	63.4
SuperSQL	66.9	46.5	43.8	58.5
CodeS	65.8	48.8	42.4	58.5
Reasoning-SQL	71.6	53.7	46.2	64.0
SF-SQL	**74.2**	61.9	**61.1**	**69.2**

SQLite, MySQL, PostgreSQL, Oracle, and SQL Server. We compare the base and SFT models to clearly isolate and highlight the performance gains brought by SF-SQL on different dialects.

The original BIRD Mini-Dev benchmark provides SQLite, MySQL, and PostgreSQL versions. To further extend the evaluation to Oracle and SQL Server, we construct corresponding versions of the dataset based on the original SQLite data. Specifically, we first migrate the databases from SQLite to Oracle and SQL Server. We then translate the SQL queries using a parser-based translation approach. When the translation approach fails, we use an LLM as a fallback. To verify correctness, we execute each translated SQL query on both the original SQLite database and the target Oracle/SQL Server database, and compare whether the execution results are consistent. If the results are fully consistent, we regard the translation as successful. Otherwise, we further ask the LLM to judge whether the discrepancy is caused by format differences rather than translation errors. Using this pipeline, we successfully translate 498 SQLite SQL queries into Oracle and SQL Server, and manually correct the final 2 cases.

According to the results in Table 4, the baseline Qwen2.5-Coder-7B model achieves an execution accuracy (EX) of 46.00% on SQLite, but drops to 35.00% on MySQL, 37.40% on PostgreSQL, 39.40% on Oracle, and 35.20% on SQL Server, revealing a clear training stage bias toward SQLite. In contrast, SF-SQL achieves EX of 65.00%, 54.00%, 55.40%, 56.80%, and 52.20% on SQLite, MySQL, PostgreSQL, Oracle, and SQL Server, respectively. Compared with the base model, this corresponds to absolute gains of 19.00%, 19.00%, 18.00%, 17.40%, and 17.00% on the five dialects. These results demonstrate that SF-SQL consistently delivers substantial improvements across diverse SQL dialects, underscoring its robustness and effectiveness in cross-dialect environments.

4.4 Effectiveness of consistency validation

To further assess the effectiveness of our consistency validation process, we employ a community-curated correction set for 500 examples from the BIRD Mini-Dev [22]. Examples identified as containing errors are each annotated with exactly one of eight correction types, indicating the nature of the error. For our evaluation, we treat three correction types: *fix_question_and_sql*, *fix_sql*, and *fix_evidence* as negative examples, as they correspond to genuine logical faults. The remaining types, which mainly involve superficial edits or clarification (e.g., rewording, schema hints), along with examples that require no correction, are treated as positive examples. We deployed the Qwen2.5-Coder-32B model across two NVIDIA A100 (80GB) GPUs. Notably, this consistency validation is a one-time offline preprocessing step applied before model training and therefore introduces no additional inference-time overhead. The comprehensive consistency validation of the BIRD training dataset took approximately 8 hours.

As shown in Table 5, the module achieves a recall of 0.85, a precision of 0.91, and an F_1 score of 0.88, accurately identifying the majority of real logical errors while keeping false positives at a low level, demonstrating its potential for practical deployment in automated SQL debugging. Training data containing a moderate amount of noise can still be beneficial to model performance, as demonstrated in prior studies. However, excessive noise may lead to performance degradation [51–53]. The consistency validation module helps mitigate the impact of noisy data in Text-to-SQL datasets by identifying and filtering out incorrect examples.

4.5 Inference cost analysis

To better understand how query difficulty influences the inference cost of SF-SQL, we examine its impact on two key aspects: the number of reasoning steps and the response token length. Since the GPT-4 API was unavailable during our evaluation, we use GPT-4o for this analysis. Results are reported in Table 6. We observe that the average number of reasoning steps increases from 3.5 for simple questions to 5.6 for moderate ones and 7.2 for challenging ones, with an overall mean of 4.5. This indicates that more difficult queries require deeper reasoning and more intermediate steps before producing the final SQL. Correspondingly, the average response length exhibits a substantial increase, rising from 1.2k for simple queries to 2.2k for moderate ones and 7.8k for challenging ones, with an overall average of 2.1k. This trend suggests that an increased number of reasoning steps often leads to more verbose responses, as the model tends to generate longer thinking traces and explore more actions to handle complex queries.

As further shown in Fig. 8, SF-SQL maintains a relatively concise response length compared with representative baselines. In particular, on both simple and moderate queries, SF-SQL produces fewer tokens

Table 4 Execution accuracy (%) on BIRD Mini-Dev

Database	Qwen2.5-Coder-7B	Qwen2.5-Coder-7B SFT	SF-SQL
SQLite	46.00	33.40	**65.00**
MySQL	35.00	32.20	**54.00**
PostgreSQL	37.40	33.00	**55.40**
Oracle	39.40	33.60	**56.80**
SQL Server	35.20	30.80	**52.20**
Average	38.60	32.60	**56.48**

Table 5 Performance of consistency validation on BIRD Mini-Dev Correction

Total		Predicted		Metrics		
		P	N	Recall	Precision	F_1 Score
Actual	**P**	361	66	0.85	0.91	0.88
	N	37	34			

Table 6 The average response length and the number of reasoning steps across different difficulty levels on BIRD

Method	Simple	Moderate	Challenge	All
Token length	1.2k	2.2k	7.8k	2.1k
Reasoning steps	3.5	5.6	7.2	4.5

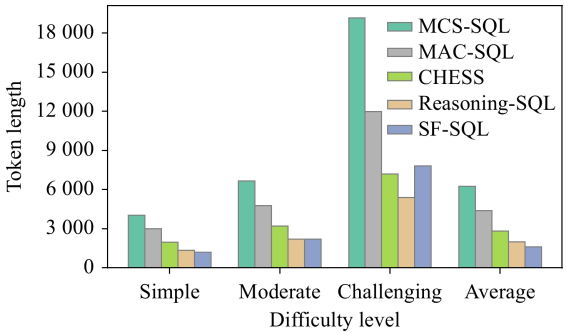

Fig. 8 Inference cost across different difficulty levels on BIRD

than Reasoning-SQL. We attribute this difference to the fact that Reasoning-SQL follows the CHASE-SQL [27] pipeline, where a single query may involve multiple rounds of generation, thereby increasing the overall token consumption even for relatively easier cases. In contrast, SF-SQL adopts a stage-free process, which allows the model to flexibly decide the next action according to the current context and terminate earlier when sufficient information has been obtained. These observations suggest that SF-SQL achieves a favorable trade-off between inference cost and reasoning flexibility.

4.6 Ablation study

We conduct an ablation study on the BIRD dataset to assess the contribution of different components in SF-SQL. The study includes the base model Qwen2.5-Coder-7B, its supervised fine-tuned variant (SFT), and several configurations of SF-SQL. Specifically, we evaluate the full model, as well as the versions without consistency validation, without trajectory pruning, and without each of the different rewards (i.e., accuracy, format, intermediate, and step).

As shown in Table 7, full SF-SQL achieves the best performance. Removing the consistency validation module results in a 2.22% drop in execution accuracy (EX), and omitting the trajectory pruning significantly impacts performance, causing EX to drop to 64.99%, confirming the importance of pruning for stable reinforcement policy optimization.

Finally, examining the contributions of individual components within the composite reward function provides further insight into their roles. Excluding the accuracy or step rewards leads to a substantial decline in performance, with execution accuracy dropping to 64.54%, representing a 4.69% decrease, and to 64.54%, representing a 4.69% decrease, respectively. The accuracy reward directs the policy toward correct final execution, while the step reward enforces structured, progressive decomposition by crediting only intermediate transitions that make measurable forward progress and penalizing redundant or stagnant moves. Omitting either signal weakens the formation of reliable reasoning trajectories. In contrast, format and intermediate rewards function as auxiliary signals. Format reward helps enforce the surface-level structure of the query, while intermediate reward provides additional local guidance on substeps. Removing them leads to slight drops in accuracy, with scores decreasing to 68.12%, a reduction of 1.11%, and 67.47%, a reduction of 1.76%, respectively.

Table 7 Ablation study on BIRD

Method	EX/%
Qwen2.5-Coder-7B	48.50
Qwen2.5-Coder-7B SFT	39.89
SF-SQL ALL	**69.23**
SF-SQL w/o Consistency Validation	67.01 (↓ 2.22)
SF-SQL w/o Trajectory Pruning	64.99 (↓ 4.24)
SF-SQL w/o Reward$_{(accuracy)}$	64.54 (↓ 4.69)
SF-SQL w/o Reward$_{(format)}$	68.12 (↓ 1.11)
SF-SQL w/o Reward$_{(intermediate)}$	67.47 (↓ 1.76)
SF-SQL w/o Reward$_{(step)}$	64.54 (↓ 4.69)

4.7 Case study

A case study of SF-SQL on the BIRD dataset is illustrated in Fig. 9, which involves generating a SQL query to retrieve the highest eligible free rate for K-12 students in schools located in Alameda County. SF-SQL initially constructs a query using `"County Name" = 'Alameda County'`. However, this query returns null, suggesting that the filter condition may not match any existing records. To address the issue, SF-SQL examines whether any records exist for `'Alameda County'` and finds none. It then inspects the distinct entries under `"County Name"` and discovers that the actual value for Alameda County is `'Alameda'`. Finally, SF-SQL fixes the query with the corrected filter condition.

SF-SQL initially applies an incorrect filter condition, `"County Name" = 'Alameda County'`. By leveraging execution feedback from the database, it detects the mismatch and

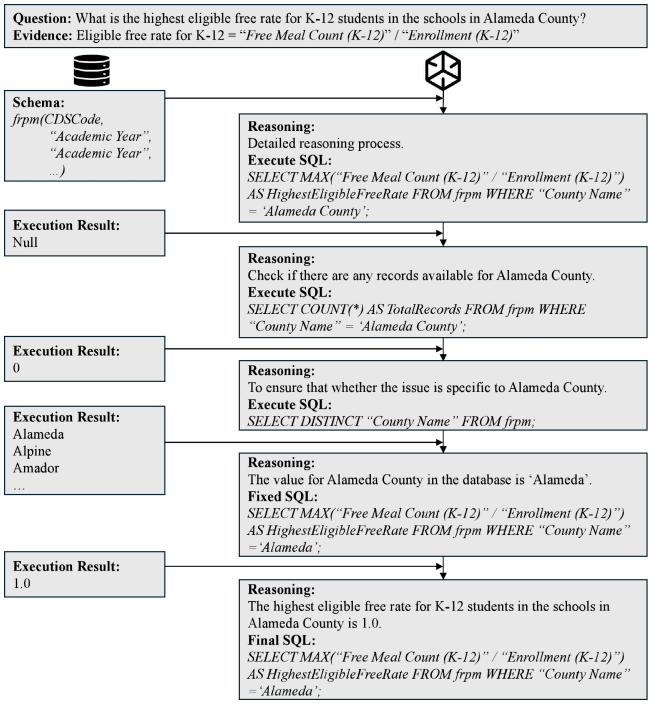

Fig. 9 A case study of SF-SQL on BIRD

inspects the values in the `"County Name"` column, correcting the condition to `"County Name" = 'Alameda'`. The revised SQL successfully returns the desired result. This case demonstrates the ability of SF-SQL to utilize environmental feedback to refine its predictions through contextual reasoning and self-correction iteratively.

5 Related work

Text-to-SQL. Early Text-to-SQL studies rely on hand-crafted rules or semantic parsers to interpret natural language [54–56]. While interpretable, they generalize poorly and incur high maintenance costs. To address these limitations, neural network models are introduced [9,57–58], particularly sequence-to-sequence models. Transformer-based models such as BERT [6], BART [59], and T5 [60] have been used to improve Text-to-SQL performance by enhancing the model's ability in capturing question-schema semantics [61–63].

LLMs further improve Text-to-SQL performance, driven by their strong reasoning capabilities and broad general-purpose knowledge. However, existing LLM-based methods [15–17,24] rely on multi-stage pipelines incorporating techniques such as chain-of-thought prompting, retrieval augmentation, and execution feedback. For example, MCS-SQL [17] combines schema linking, multiple candidate generation, and few-shot selection to improve accuracy and robustness. While reinforcement learning has also been explored [9,64], it is applied to stages within fixed pipelines. For instance, Reasoning-SQL [21] follows the CHASE-SQL pipeline [27]. It still lacks flexibility and relies on task-specific modular designs and manual prompt engineering.

RL for LLMs. Reinforcement learning has become increasingly popular in training LLMs. For example, Reinforcement Learning from Human Feedback (RLHF) notably improves the LLM's instruction-following capabilities, as validated in InstructGPT [40]. Other RL strategies such as Direct Preference Optimization (DPO) [31], Reinforcement Learning from AI Feedback (RLAIF) [65], and Group Relative Policy Optimization (GRPO) [29] have further enhanced LLM training by improving model alignment, safety, and scalability. These RL-based methods are widely adopted in modern LLMs such as OpenAI's o1 [23], DeepSeek-R1 [29], and QwQ [66].

The quality of training data is crucial in RL fine-tuning of LLMs [32–34]. Factors such as data diversity and preference pair quality have been shown to directly impact the model performance. Although Text-to-SQL shares similarities with general reasoning tasks in terms of question comprehension, instruction following, and response generation, it introduces distinct challenges, particularly regarding the evaluation of training data quality and the design of reward signals.

6 Conclusion

In this paper, we present SF-SQL, a stage-free Text-to-SQL framework enhanced by reinforcement learning. Instead of following a rigid, predefined pipeline, SF-SQL formulates SQL generation as a flexible sequence of reasoning-action steps, enabling dynamic decision-making guided by environmental feedback, which improves model performance. To support effective policy learning over a large and diverse action space, we introduce a trajectory pruning strategy that discards uninformative samples while preserving diverse, high-quality reasoning-action trajectories. We further design a fine-grained reward mechanism that combines final and intermediate signals, providing rich supervision without incurring additional inference overhead. Extensive experiments on Spider and BIRD benchmarks show that SF-SQL achieves high accuracy. Furthermore, cross-dialect evaluations on SQLite, MySQL, PostgreSQL, Oracle, and SQL Server confirm its robustness and generalization across heterogeneous database environments. These results highlight the potential of reinforcement learning to improve the reasoning capabilities of LLMs for Text-to-SQL.

Acknowledgements

This work was partially supported by National Natural Science Foundation of China (NSFC) (Grant Nos. 62425202, 62336003), the Beijing Natural Science Foundation (Z230001), and the State Key Laboratory of Complex & Critical Software Environment (SKLCCSE). Zimu Zhou's research is supported by CityU APRC grant (No. 9610633).

Competing interests

Yongxin TONG is an Action Editor of the journal and a co-author of this article. To minimize bias, he was excluded from all editorial decision-making related to the acceptance of this article for publication. The remaining authors declare no conflict of interest.

References

[1] Xue S, Qi D, Jiang C, Cheng F, Chen K, Zhang Z, Zhang H, Wei G, Zhao W, Zhou F, Yi H, Liu S, Yang H, Chen F. Demonstration of DB-GPT: next generation data interaction system empowered by large language models. Proceedings of the VLDB Endowment, 2024, 17(12): 4365−4368

[2] Li F, Jagadish H V. Constructing an interactive natural language interface for relational databases. Proceedings of the VLDB Endowment, 2014, 8(1): 73−84

[3] Hristidis V, Papakonstantinou Y. DISCOVER: keyword search in relational databases. In: Proceedings of 28th International Conference on Very Large Data Bases. 2002, 670−681

[4] Hristidis V, Gravano L, Papakonstantinou Y. Efficient IR-style keyword search over relational databases. In: Proceedings of 29th International Conference on Very Large Data Bases. 2003, 850−861

[5] Luo Y, Lin X, Wang W, Zhou X. Spark: top-k keyword query in relational databases. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. 2007, 115−126

[6] Devlin J, Chang M W, Lee K, Toutanova K. BERT: pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. 2019, 4171−4186

[7] Yin P, Neubig G, Yih W T, Riedel S. TaBERT: pretraining for

joint understanding of textual and tabular data. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. 2020, 8413−8426

[8] Brunner U, Stockinger K. ValueNet: a natural language-to-SQL system that learns from database information. In: Proceedings of 2021 IEEE 37th International Conference on Data Engineering (ICDE). 2021, 2177−2182

[9] Zhong V, Xiong C, Socher R. Seq2SQL: generating structured queries from natural language using reinforcement learning. 2017, arXiv preprint arXiv: 1709.00103

[10] Wang B, Shin R, Liu X, Polozov O, Richardson M. RAT-SQL: relation-aware schema encoding and linking for text-to-SQL parsers. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. 2020, 7567−7578

[11] Liu H, Shi Y, Zhang J, Wang X, Li H, Kong F. Multi-hop relational graph attention network for text-to-SQL parsing. In: Proceedings of 2023 International Joint Conference on Neural Networks (IJCNN). 2023, 1−8

[12] Achiam J, Adler S, Agarwal S, Ahmad L, Akkaya I, et al. GPT-4 technical report. 2023, arXiv preprint arXiv: 2303.08774

[13] Touvron H, Lavril T, Izacard G, Martinet X, Lachaux M A, Lacroix T, Rozière B, Goyal N, Hambro E, Azhar F, Rodriguez A, Joulin A, Grave E, Lample G. LLaMA: open and efficient foundation language models. 2023, arXiv preprint arXiv: 2302.13971

[14] Bai J, Bai S, Chu Y, Cui Z, Dang K, et al. Qwen technical report. 2023, arXiv preprint arXiv: 2309.16609

[15] Gao D, Wang H, Li Y, Sun X, Qian Y, Ding B, Zhou J. Text-to-SQL empowered by large language models: a benchmark evaluation. Proceedings of the VLDB Endowment, 2024, 17(5): 1132−1145

[16] Pourreza M, Rafiei D. DIN-SQL: decomposed in-context learning of text-to-SQL with self-correction. In: Proceedings of the 37th International Conference on Neural Information Processing Systems. 2023, 1577

[17] Lee D, Park C, Kim J, Park H. MCS-SQL: Leveraging multiple prompts and multiple-choice selection for text-to-SQL generation. In: Proceedings of the 31st International Conference on Computational Linguistics. 2025, 337−353

[18] Talaei S, Pourreza M, Chang Y C, Mirhoseini A, Saberi A. CHESS: contextual harnessing for efficient SQL synthesis. 2024, arXiv preprint arXiv: 2405.16755

[19] Li H, Zhang J, Liu H, Fan J, Zhang X, Zhu J, Wei R, Pan H, Li C, Chen H. CodeS: Towards building open-source language models for text-to-SQL. Proceedings of the ACM on Management of Data, 2024, 2(3): 127

[20] Yang J, Hui B, Yang M, Yang J, Lin J, Zhou C. Synthesizing text-to-SQL data from weak and strong LLMs. In: Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2024, 7864−7875

[21] Pourreza M, Talaei S, Sun R, Wan X, Li H, Mirhoseini A, Saberi A, Arık S Ö. Reasoning-SQL: Reinforcement learning with SQL tailored partial rewards for reasoning-enhanced text-to-SQL. In: Proceedings of the 2nd Conference on Language Modeling. 2025

[22] Li J, Hui B, Qu G, Yang J, Li B, Li B, Wang B, Qin B, Geng R, Huo N, Zhou X, Ma C, Li G, Chang K C C, Huang F, Cheng R, Li Y. Can LLM already serve as a database interface? A big bench for large-scale database grounded text-to-SQLs. In: Proceedings of the 37th International Conference on Neural Information Processing Systems. 2023, 1835

[23] Jaech A, Kalai A, Lerer A, Richardson A, El-Kishky A, et al. OpenAI o1 system card. 2024, arXiv preprint arXiv: 2412.16720

[24] Ren T, Fan Y, He Z, Huang R, Dai J, Huang C, Jing Y, Zhang K, Yang Y, Wang S. PURPLE: making a large language model a better SQL writer. In: Proceedings of 40th IEEE International Conference on Data Engineering. 2024, 15−28

[25] Shin J, Tang C, Mohati T, Nayebi M, Wang S, Hemmati H. Prompt engineering or fine-tuning: an empirical assessment of LLMs for code. In: Proceedings of 2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR). 2025, 490−502

[26] Trung L, Zhang X, Jie Z, Sun P, Jin X, Li H. ReFT: reasoning with reinforced fine-tuning. In: Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2024, 7601−7614

[27] Pourreza M, Li H, Sun R, Chung Y, Talaei S, Kakkar G T, Gan Y, Saberi A, Ozcan F, Arik S Ö. Chase-SQL: multi-path reasoning and preference optimized candidate selection in text-to-SQL. In: Proceedings of the Thirteenth International Conference on Learning Representations. 2025

[28] Shao Z, Wang P, Zhu Q, Xu R, Song J, Bi X, Zhang H, Zhang M, Li Y K, Wu Y, Guo D. DeepSeekMath: pushing the limits of mathematical reasoning in open language models. 2024, arXiv preprint arXiv: 2402.03300

[29] Guo D, Yang D, Zhang H, Song J, Wang P, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. 2025, arXiv preprint arXiv: 2501.12948

[30] Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal policy optimization algorithms. 2017, arXiv preprint arXiv: 1707.06347

[31] Rafailov R, Sharma A, Mitchell E, Ermon S, Manning C D, Finn C. Direct preference optimization: your language model is secretly a reward model. In: Proceedings of the 37th International Conference on Neural Information Processing Systems. 2023, 2338

[32] Ji Y, Zhao S, Tian X, Wang H, Chen S, Peng Y, Zhao H, Li X. How difficulty-aware staged reinforcement learning enhances LLMs' reasoning capabilities: a preliminary experimental study. 2025, arXiv preprint arXiv: 2504.00829

[33] Morimura T, Sakamoto M, Jinnai Y, Abe K, Ariu K. Filtered direct preference optimization. In: Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing. 2024, 22729−22770

[34] Zhang J, Zuo C. GRPO-LEAD: a difficulty-aware reinforcement learning approach for concise mathematical reasoning in language models. In: Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing. 2025, 5642−5654

[35] Li B, Zhang J, Fan J, Xu Y, Chen C, Tang N, Luo Y. Alpha-SQL: zero-shot text-to-SQL using Monte Carlo tree search. In: Proceedings of Forty-Second International Conference on Machine Learning. 2025

[36] Indyk P, Motwani R. Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing. 1998, 604−613

[37] Robertson S, Zaragoza H. The probabilistic relevance framework: BM25 and beyond. Foundations and Trends in Information Retrieval, 2009, 3(4): 333−389

[38] Wretblad N, Riseby F, Biswas R, Ahmadi A, Holmström O. Understanding the effects of noise in text-to-SQL: an examination of the BIRD-bench benchmark. In: Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers). 2024, 356−369

[39] Wang X, Wei J, Schuurmans D, Le Q V, Chi E H, Narang S, Chowdhery A, Zhou D. Self-consistency improves chain of thought reasoning in language models. In: Proceedings of the Eleventh International Conference on Learning Representations. 2023

[40] Ouyang L, Wu J, Jiang X, Almeida D, Wainwright C L, Mishkin P, Zhang C, Agarwal S, Slama K, Ray A, Schulman J, Hilton J, Kelton F, Miller L, Simens M, Askell A, Welinder P, Christiano P, Leike J, Lowe R. Training language models to follow instructions with human feedback. In: Proceedings of the 36th International Conference on Neural Information Processing Systems. 2022, 2011

[41] Yu T, Zhang R, Yang K, Yasunaga M, Wang D, Li Z, Ma J, Li I, Yao Q, Roman S, Zhang Z, Radev D R. Spider: a large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing. 2018, 3911−3921

[42] Wang B, Ren C, Yang J, Liang X, Bai J, Chai L, Yan Z, Zhang Q W, Yin D, Sun X, Li Z. MAC-SQL: a multi-agent collaborative framework for text-to-SQL. In: Proceedings of the 31st International Conference on Computational Linguistics. 2025, 540−557

[43] Maamari K, Abubaker F, Jaroslawicz D, Mhedhbi A. The death of schema linking? Text-to-SQL in the age of well-reasoned language models. 2024, arXiv preprint arXiv: 2408.07702

[44] Li B, Luo Y, Chai C, Li G, Tang N. The dawn of natural language to SQL: are we fully ready? Proceedings of the VLDB Endowment, 2024, 17(11): 3318−3331

[45] Li R, Allal L B, Zi Y, Muennighoff N, Kocetkov D, et al. StarCoder: may the source be with you! 2023, arXiv preprint arXiv: 2305.06161

[46] Hui B, Yang J, Cui Z, Yang J, Liu D, et al. Qwen2.5-Coder technical report. 2024, arXiv preprint arXiv: 2409.12186

[47] Sheng G, Zhang C, Ye Z, Wu X, Zhang W, Zhang R, Peng Y, Lin H, Wu C. HybridFlow: a flexible and efficient RLHF framework. In: Proceedings of the Twentieth European Conference on Computer Systems. 2025, 1279−1297

[48] Wang W, Wei F, Dong L, Bao H, Yang N, Zhou M. MiniLM: deep self-attention distillation for task-agnostic compression of pre-trained transformers. In: Proceedings of the 34th International Conference on Neural Information Processing Systems. 2020, 485

[49] Taipalus T. Vector database management systems: Fundamental concepts, use-cases, and current challenges. Cognitive Systems Research, 2024, 85: 101216

[50] Kwon W, Li Z, Zhuang S, Sheng Y, Zheng L, Yu C H, Gonzalez J E, Zhang H, Stoica I. Efficient memory management for large language model serving with PagedAttention. In: Proceedings of the 29th Symposium on Operating Systems Principles. 2023, 611−626

[51] Zhu D, Hedderich M A, Zhai F, Adelani D I, Klakow D. Is BERT robust to label noise? A study on learning with noisy labels in text classification. In: Proceedings of the Third Workshop on Insights from Negative Results in NLP. 2022, 62−67

[52] Wang S, Tan Z, Guo R, Li J. Noise-robust fine-tuning of pretrained language models via external guidance. In: Proceedings of Findings of the Association for Computational Linguistics: EMNLP 2023. 2023, 12528−12540

[53] Rolnick D, Veit A, Belongie S, Shavit N. Deep learning is robust to massive label noise. 2017, arXiv preprint arXiv: 1705.10694

[54] Katsogiannis-Meimarakis G, Koutrika G. A survey on deep learning approaches for text-to-SQL. The VLDB Journal, 2023, 32(4): 905−936

[55] Zelle J M, Mooney R J. Learning to parse database queries using inductive logic programming. In: Proceedings of the Thirteenth National Conference on Artificial Intelligence. 1996, 1050−1055

[56] Li F, Jagadish H V. NaLIR: an interactive natural language interface for querying relational databases. In: Proceedings of International Conference on Management of Data. 2014, 709−712

[57] Xu X, Liu C, Song D. SQLNet: generating structured queries from natural language without reinforcement learning. 2017, arXiv preprint arXiv: 1711.04436

[58] Gur I, Yavuz S, Su Y, Yan X. DialSQL: dialogue based structured query generation. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2018, 1339−1349

[59] Lewis M, Liu Y, Goyal N, Ghazvininejad M, Mohamed A, Levy O, Stoyanov V, Zettlemoyer L. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. 2020, 7871−7880

[60] Raffel C, Shazeer N, Roberts A, Lee K, Narang S, Matena M, Zhou Y, Li W, Liu P J. Exploring the limits of transfer learning with a unified text-to-text transformer. The Journal of Machine Learning Research, 2020, 21(1): 5485−5551

[61] Li J, Hui B, Cheng R, Qin B, Ma C, Huo N, Huang F, Du W, Si L, Li Y. Graphix-T5: mixing pre-trained transformers with graph-aware layers for text-to-SQL parsing. In: Proceedings of Thirty-Seventh AAAI Conference on Artificial Intelligence. 2023, 13076−13084

[62] Li H, Zhang J, Li C, Chen H. RESDSQL: decoupling schema linking and skeleton parsing for text-to-SQL. In: Proceedings of Thirty-Seventh AAAI Conference on Artificial Intelligence. 2023, 13067−13075

[63] Gan Y, Chen X, Xie J, Purver M, Woodward J R, Drake J, Zhang Q. Natural SQL: making SQL easier to infer from natural language specifications. In: Proceedings of Findings of the Association for Computational Linguistics: EMNLP 2021. 2021, 2030−2042

[64] Bai Z, Yu B, Wu B, Wang Z, Wang B. Learning to generate structured queries from natural language with indirect supervision. Computer Speech & Language, 2021, 67: 101185

[65] Bai Y, Kadavath S, Kundu S, Askell A, Kernion J, et al. Constitutional AI: harmlessness from AI feedback. 2022, arXiv preprint arXiv: 2212.08073

[66] Yang A, Yang B, Zhang B, Hui B, Zheng B, et al. Qwen2.5 technical report. 2024, arXiv preprint arXiv: 2412.15115

Yang SONG is currently working toward the PhD degree in the School of Computer Science and Engineering, Beihang University, China. His research interests include text-to-sql, large language model, and llm agent.

Shurui KOU received the BE degree in the computer science and technology from Beihang University, China in 2024. He is currently working toward the PhD degree in the School of Computer Science and Engineering, Beihang University, China. His major research interests include text-to-sql, large language model, and llm agent.

Zimu ZHOU received the BE from the Department of Electronic Engineering, Tsinghua University, China in 2011 and the PhD from the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, China in 2015. His research focuses on mobile, ubiquitous computing, and federated learning.

Qian TAO received the PhD degree in Computer Science and Technology from Beihang University, China, in 2021, and the BS degree from the same institution. Currently, he serves as an Associate Professor at the School of Artificial Intelligence, Beihang University, and has been an Algorithm Engineer at Alibaba Tongyi Lab. His research interests primarily focus on federated large language models and federated spatial-temporal data analysis.

Yongxin TONG received the PhD degree in Computer Science and Engineering from the Hong Kong University of Science and Technology, China in 2014. He is currently a professor in the School of Computer Science and Engineering, Beihang University, China. His research interests include federated large language model, federated learning, spatio-temporal data analytics, and reinforcement learning. He received the championship of KDD Cup 2020 RL track.