Every Step Counts: Step-Level Credit Assignment for Tool-Integrated Text-to-SQL

Anonymous ACL submission

Abstract

Tool-integrated reasoning has emerged as a promising paradigm for Text-to-SQL, framing SQL generation as sequential decision-making interleaved with tool execution. However, existing reinforcement learning approaches rely on coarse-grained outcome supervision, creating a credit assignment problem: models receive identical rewards regardless of whether intermediate steps are redundant or erroneous. We observe that tool-integrated reasoning offers a unique opportunity: each tool invocation produces *structured, verifiable outputs* that can serve as training-free step-level signals, eliminating the need for expensive reward model training. Building on this insight, we propose **FineStep**, a step-level credit assignment framework that leverages *training-free process rewards* derived from tool execution feedback. FineStep constructs a dual-channel value estimate for each step, combining backward-propagated outcome value with forward-smoothed process value, and synthesizes them via group-based normalization into a unified step-level advantage for policy optimization. Experiments on Text-to-SQL benchmarks show that FineStep achieves SOTA performance with a 3.25% EX gain over GRPO at the 4B scale, while results on AIME24 demonstrate transferability beyond SQL.

1 Introduction

The Text-to-SQL task, which aims to translate natural language questions into executable SQL queries, serves as a critical bridge connecting non-expert users with structured data (Liu et al., 2024; Zhang et al., 2024). With the rapid advancement of Large Language Models (LLMs), the research paradigm in this field is undergoing a profound shift from single-step semantic parsing to Tool-augmented Agentic Reasoning (Singh et al., 2025; Mai et al., 2025; Wang et al., 2025). Under this paradigm, the model no longer generates the final query in a single pass but iteratively approaches the solution through multi-turn interactions with an SQL executor: generating SQLs, observing execution feedback, and refining logic in a loop until the correct answer is derived (Dai et al., 2025).

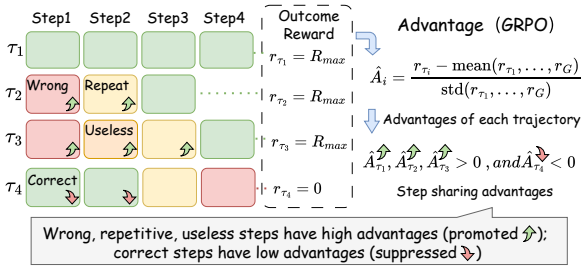

Figure 1: The credit assignment problem in standard GRPO (Shao et al., 2024). τ denotes a reasoning trajectory whose final step outputs the answer. Colors mark wrong, repetitive, useless, and correct steps, identified via tool calls in tool-integrated reasoning.

To optimize decision-making in this complex search space, Agentic Reinforcement Learning (Zhang et al., 2026) has become the dominant paradigm (Shang et al., 2025; Dong et al., 2025). However, existing methods (e.g., GRPO (Shao et al., 2024)) rely solely on final execution rewards (Dai et al., 2025; Xu et al., 2025), facing a severe Credit Assignment Problem: sparse feedback rewards trajectories as a whole, failing to distinguish effective steps from redundant operations (Feng et al., 2025; Li et al., 2025c; Xue et al., 2025; Ji et al., 2025; Xia et al., 2025). As illustrated in Figure 1, a trajectory filled with erroneous steps receives high rewards solely because the final answer is correct, inducing the model to learn "blind trial-and-error" strategies rather than genuine reasoning (Chen et al., 2025; Xia et al., 2025).

Our key insight is that tool-integrated reasoning provides a unique opportunity for *training-free process supervision*: each tool invocation produces structured, verifiable outputs (e.g., SQL execution

results) that can be automatically assessed, unlike free-form reasoning which requires expensive annotation or reward model training.

Building on this insight, we propose **FineStep**, a step-level credit assignment framework with training-free process rewards. FineStep follows a principled three-stage design: (1) **Signal Construction**: extracting dual-channel rewards from tool feedback, including an outcome reward for global correctness and a process reward for step-wise information gain; (2) **Value Propagation**: transforming raw signals into step-level value estimates via outcome discounting (temporal causality) and process smoothing (logical dependencies); (3) **Advantage Synthesis**: fusing both channels through group-based normalization into a unified step-level advantage that identifies "correct steps in failed trajectories" and "redundancies in successful ones." In summary, the main contributions of this paper are as follows:

- We propose **FineStep**, a step-level credit assignment framework with *training-free* process rewards derived from structured tool outcome.
- We design a value estimation pipeline with dual-channel reward signal, temporal-logical value propagation, and group advantage synthesis.
- Experiments on Text-to-SQL benchmarks show that FineStep achieves SOTA accuracy while reducing redundant tool interactions. Results on AIME24 further demonstrate cross-domain transferability.

2 Related Work

2.1 LLM Reasoning in Text-to-SQL

The Text-to-SQL paradigm has evolved through three stages: (1) *Prompt Engineering* (Wei et al., 2022; Pourreza and Rafiei, 2023; Li et al., 2025a; Pourreza et al., 2024) via static workflows; (2) *Single-turn RL* (Pourreza et al., 2025; Ma et al., 2025; Yao et al., 2025) that fine-tunes models with final execution accuracy but lacks self-correction (Dai et al., 2025; Yang et al., 2025); and (3) *Tool-integrated Agentic Reasoning* (Dai et al., 2025; Xu et al., 2025; Liu et al., 2025; Yang et al., 2025) that enables iterative refinement through multi-turn model-tool interaction. Despite this evolution, existing optimization methods still rely on coarse-grained outcome supervision (Shao et al., 2024; Yu et al., 2025): the entire trajectory receives positive reinforcement as long as the final SQL is correct, failing to distinguish intermediate step quality and highlighting a critical limitation in credit assignment for multi-turn Text-to-SQL.

Response Format:
<reasoning> Reasoning process here. </reasoning>
<tool_call> {"name": "sql_executor", "arguments": {"sql": "YOUR_SQL"}} </tool_call>
<result> Execution result from the engine. </result>
<reasoning> Further reasoning. </reasoning>
<answer> (Final verified SQL) </answer>

Figure 2: Response format for FineStep.

2.2 Credit Assignment in LLM Reasoning

To mitigate feedback sparsity, process supervision has attracted growing interest (Setlur et al., 2024; Zhang et al., 2025). Existing approaches include: (1) *Learned PRMs* (Lightman et al., 2024; Wang et al., 2024; Cui et al., 2025; Zhang et al., 2025; Guo et al., 2025) that train auxiliary models to score steps but require expensive annotation; (2) *Heuristic decay* (Feng et al., 2025; Chen et al., 2025; Fan et al., 2026) that discounts outcome rewards over time without step-wise semantics; and (3) *Monte Carlo estimation* (Wang et al., 2024) via rollouts, which is costly for long-horizon reasoning. FineStep differs by being *training-free* yet *semantics-aware*: it exploits structured tool execution outputs to construct dual-channel step-level value estimates (temporal + logical), requiring neither model training nor expensive rollouts. Existing PRMs (Cui et al., 2025; Zhang et al., 2025) target internal multi-step reasoning (CoT without environment interaction), whereas FineStep addresses *tool-integrated* reasoning where each step involves external tool execution and feedback.

3 Methodology

We propose **FineStep**, a step-level credit assignment framework illustrated in Figure 3. The core idea is to exploit the structured, verifiable nature of tool execution outputs to construct training-free step-level value estimates. FineStep implements a three-stage value estimation pipeline: (1) **Signal Construction** (§3.2): extracting dual-channel reward signals from tool feedback; (2) **Value Propagation** (§3.3): transforming raw signals into step-level values via outcome discounting and process smoothing; (3) **Advantage Synthesis** (§3.3): fusing both channels through group-based normaliza-

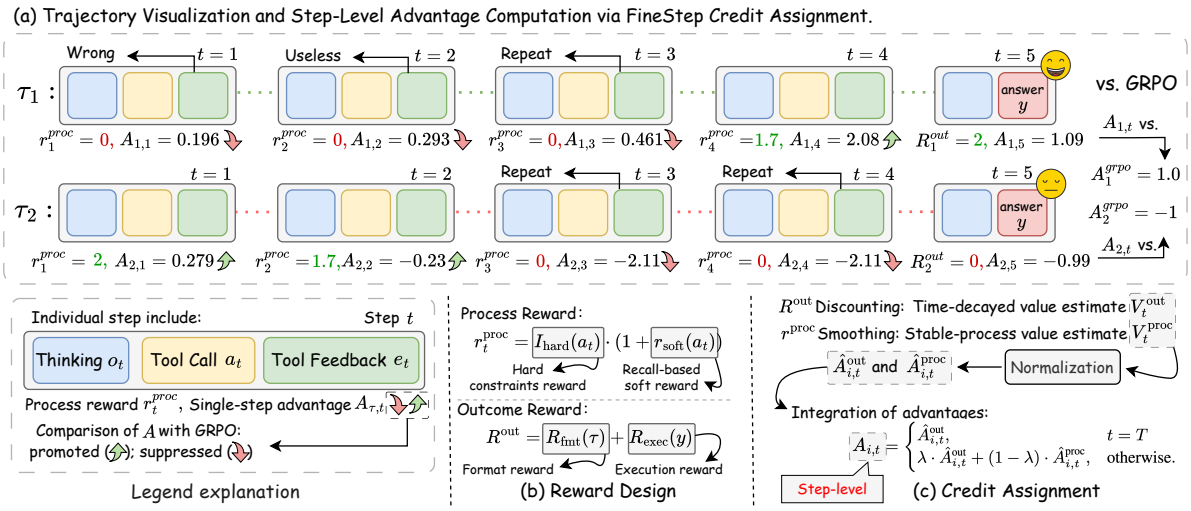

Figure 3: Overview of FineStep, illustrating its step-level credit assignment and reward design. (a) Trajectory visualization and step-level advantage computation, showing how FineStep promotes or suppresses specific reasoning steps compared to GRPO; (b) reward design with process and outcome rewards; and (c) credit assignment integrating discounted outcome and smoothed process advantages. More details are provided in Section 3.

tion into a unified step-level advantage for policy optimization (§3.4).

3.1 Problem Definition

We formulate the tool-augmented Text-to-SQL task as a sequential decision-making process interacting with an SQL executor. Given a natural language question x and a database schema $\mathcal{S}$, the initial state is denoted as $s_{\text{init}} = \{x, \mathcal{S}\}$. As illustrated in Figure 2, the model adheres to a "Think-Act-Observe" paradigm. Here, we explicitly define a single **reasoning step** t as a complete interaction cycle: the model's thought o_t, action a_t, and the execution feedback e_t. Finally, the model autonomously generates the final answer y. The resulting trajectory τ of length $T+1$ is:

$$\tau = \{s_{\text{init}}, o_0, a_0, e_0, \ldots, o_{T-1}, a_{T-1}, e_{T-1}, o_T, y\} \tag{1}$$

3.2 Reward Design

We construct dual-channel reward signals: an outcome reward R^{out} for global correctness, and a process reward r_t^{proc} for step-wise information gain, mitigating the sparsity of outcome supervision.

3.2.1 Outcome Reward Design

The outcome reward evaluates the global consistency of the reasoning trajectory and the correctness of the final answer, defined as the sum of format and execution rewards:

$$R^{\text{out}} = R_{\text{fmt}}(\tau) + R_{\text{exec}}(y) \tag{2}$$

We define $R_{\text{fmt}} \in \{0, 1\}$ to enforce the structural format in Figure 2, such as `<think>` and `<answer>` tokens, and $R_{\text{exec}} \in \{0, 2\}$ to reward exact matches between the execution result of the final SQL y and the ground truth, following the higher weighting strategy of ReEx-SQL (Dai et al., 2025).

3.2.2 Process Reward Design

The process reward r_t^{proc} combines a hard constraint on validity with a soft reward for information gain, both derived from tool feedback.

Hard Constraint Invalid steps not only provide zero information gain but also impair the reasoning process by introducing noise and unnecessarily lengthening the chain of thought. We define a binary gate I_{hard} that assigns zero reward to invalid steps (syntax errors or repeated queries):

$$I_{\text{hard}}(a_t) = \begin{cases} 0, & \text{if } a_t \text{ is invalid or repetitive} \\ 1, & \text{otherwise} \end{cases} \tag{3}$$

Recall-based Soft Reward In multi-step SQL reasoning, intermediate results naturally form supersets of the final answer (broad filtering → specific refinement). We thus use cell-level recall to quantify information gain:

$$r_{\text{soft}}(a_t) = \frac{|E(a_t) \cap E(y)|}{|E(y)| + \epsilon} \tag{4}$$

where $E(\cdot)$ denotes the flattened set of cell values from execution results.

Comprehensive Process Reward The final process reward gates the soft reward behind validity:

$$r_t^{\text{proc}} = I_{\text{hard}}(a_t) \cdot (1 + r_{\text{soft}}(a_t)) \tag{5}$$

3.3 Step-level Credit Assignment

Despite the rich signals from multi-granular rewards, direct optimization still faces challenges: (1) How to attribute credit to individual steps for the outcome reward? (2) How to incentivize indispensable "exploration actions" that yield low immediate returns? (3) How to distinguish high-quality steps from redundant operations, overcoming the limitations of trajectory-level normalization? To address these, we propose a framework comprising Outcome Reward Discounting, Process Reward Smoothing, and Step-level Advantage Estimation.

3.3.1 Outcome Reward Discounting

Since late steps have stronger causal links to the outcome than early ones, uniformly distributing R^{out} blurs credit assignment. We introduce exponential discounting γ to propagate the outcome reward backward:

$$V_t^{\text{out}} = \gamma^{T-t} \cdot R^{\text{out}}, t \in [0, T], \gamma \in (0, 1) \tag{6}$$

This prioritizes recent actions while retaining incentives for early foundational decisions.

3.3.2 Process Reward Smoothing

Rule-based process rewards may suffer from low recall, causing critical exploration actions to be overlooked. This can mislead the model into evading these essential steps and falling into local optima. To capture the underlying logical dependencies, we propose an reverse smoothing mechanism that backtracks future potential to the current step. The smoothed process value V_t^{proc} is defined as:

$$V_t^{\text{proc}} = \begin{cases} r_t^{\text{proc}}, & \text{if } t = T-1 \\ (1-\beta) \cdot r_t^{\text{proc}} + \beta \cdot V_{t+1}^{\text{proc}}, & \text{if } t \in [0, T-1) \end{cases} \tag{7}$$

where $\beta \in [0, 1]$ controls the smoothing strength. This design credits intermediate steps that enable later success, encouraging globally coherent reasoning rather than myopic optimization.

3.3.3 Step-level Advantage Estimation

To fuse the global signal V^{out} and local signal V^{proc} into a discriminative advantage, we normalize across all steps in the sampled group rather than within individual trajectories. For G trajectories, we collect value sets $\mathcal{V}^{\text{out}} = \{V_{i,t}^{\text{out}}\}$ and $\mathcal{V}^{\text{proc}} = \{V_{i,t}^{\text{proc}}\}$ from all steps, and compute standardized advantages:

$$\hat{A}_{i,t}^{\text{out}} = \frac{V_{i,t}^{\text{out}} - \text{Mean}(\mathcal{V}^{\text{out}})}{\text{Std}(\mathcal{V}^{\text{out}}) + \epsilon}, \quad \hat{A}_{i,t}^{\text{proc}} = \frac{V_{i,t}^{\text{proc}} - \text{Mean}(\mathcal{V}^{\text{proc}})}{\text{Std}(\mathcal{V}^{\text{proc}}) + \epsilon}. \tag{8}$$

The final step-level advantage is computed as

$$A_{i,t} = \begin{cases} \hat{A}_{i,t}^{\text{out}}, & t = T \\ \lambda \cdot \hat{A}_{i,t}^{\text{out}} + (1-\lambda) \cdot \hat{A}_{i,t}^{\text{proc}}, & \text{otherwise.} \end{cases} \tag{9}$$

The advantage $A_{i,t}$ is broadcast to all tokens in step t of trajectory i. For the final step ($t = T$), we use only the outcome signal, as it directly produces the answer. Group-based normalization removes baseline discrepancies and reinforces correct steps even in failed trajectories.

3.4 Policy Optimization

We optimize the policy π_θ by extending GRPO (Shao et al., 2024) to the step level, replacing trajectory-level advantages with $A_{i,t}$:

$$\mathcal{J}(\theta) = \mathbb{E}_{x \sim \mathcal{D}, \{y_i\}_{i=1}^{G} \sim \pi_{\theta_{\text{old}}}(\cdot|x;\mathcal{E})} \left[\frac{1}{G} \sum_{i=1}^{G} \frac{1}{T_i} \sum_{t=1}^{T_i} \left(\min \left(r_{i,t}(\theta) \cdot A_{i,t}, \mathit{clip} \right) - \beta \, \mathbb{D}_{\text{KL}} \right) \right],$$
$$\mathit{clip} = \text{clip}\left(r_{i,t}(\theta), 1-\epsilon, 1+\epsilon \right) A_{i,t} \tag{10}$$

where $r_{i,t}(\theta) = \frac{\pi_\theta(a_{i,t}|s_{i,t})}{\pi_{\theta_{\text{old}}}(a_{i,t}|s_{i,t})}$ is the probability ratio, and β is the KL penalty coefficient. Crucially, we mask environment feedback tokens during backpropagation, ensuring the optimization targets only the model's generative logic (thoughts and actions) rather than tool outputs.

4 Experiments

Datasets and Metrics We evaluate our method on two widely used text-to-SQL benchmarks: BIRD (Li et al., 2023) and Spider (Yu et al., 2018). For BIRD, we report execution accuracy (EX) and valid efficiency score (VES), where VES measures the execution efficiency of predicted SQL queries relative to the ground truth. For Spider, we use execution accuracy (EX) and test-suite accuracy (TS) (Zhong et al., 2020), which verifies whether predicted SQL queries produce correct results across multiple database test cases. We also evaluate three Spider robustness variants: Spider-Syn (Gan et al., 2021a), Spider-Realistic (Deng et al., 2021), and Spider-DK (Gan et al., 2021b). More details are provided in Appendix A.1.

Baselines We compare our method with four RL algorithms: GRPO (Shao et al., 2024), DAPO (Yu

Method	BIRD Dev		Spider Dev		Spider Test	Spider-Realistic		Spider-Syn		Spider-DK	Avg.
	EX	VES	EX	TS	EX	EX	TS	EX	TS	EX	
Model Size: 4B											
No-Train	65.58	67.97	85.6	75.4	85.2	82.5	66.3	76.1	65.0	73.6	74.34
GRPO	66.23	**70.65**	88.5	81.4	85.0	83.7	75.2	78.9	71.2	**80.7**	78.15
DAPO	67.67	69.93	**89.7**	**84.1**	**89.3**	87.2	78.5	80.0	**72.5**	79.6	79.85
GSPO	**68.06**	69.99	89.5	83.9	89.2	86.0	78.1	80.1	72.1	78.9	79.59
GIGPO	67.99	70.20	89.6	83.4	89.1	**87.6**	**80.5**	**80.2**	71.7	80.4	**80.07**
FineStep (Our)	**68.51**$_{\uparrow 2.28}$	**71.65**$_{\uparrow 1.00}$	**90.3**$_{\uparrow 1.8}$	**84.3**$_{\uparrow 2.9}$	**89.9**$_{\uparrow 4.9}$	**88.6**$_{\uparrow 4.9}$	**80.9**$_{\uparrow 5.7}$	**83.1**$_{\uparrow 4.2}$	**75.2**$_{\uparrow 4.0}$	**81.5**$_{\uparrow 0.8}$	**81.40**$_{\uparrow 3.25}$
Model Size: 8B											
No-Train	62.71	66.76	84.4	72.1	85.0	80.9	65.4	75.4	62.0	75.0	72.97
GRPO	66.56	68.13	87.2	79.1	88.7	83.7	75.8	79.2	70.0	78.7	77.71
DAPO	65.97	68.36	**88.6**	81.2	88.5	85.4	76.0	79.5	71.1	79.4	78.40
GSPO	66.62	**69.40**	88.4	**81.5**	**89.5**	84.8	77.0	79.6	70.3	**80.2**	78.73
GIGPO	**66.88**	68.27	**88.6**	81.2	89.4	**85.6**	**77.0**	**80.4**	**71.7**	79.1	**78.82**
FineStep (Our)	**68.06**$_{\uparrow 1.50}$	**70.90**$_{\uparrow 2.77}$	**88.9**$_{\uparrow 1.7}$	**82.4**$_{\uparrow 3.3}$	**89.9**$_{\uparrow 1.2}$	**86.0**$_{\uparrow 2.3}$	**77.9**$_{\uparrow 2.1}$	**81.4**$_{\uparrow 2.2}$	**71.6**$_{\uparrow 1.6}$	**80.0**$_{\uparrow 1.3}$	**79.71**$_{\uparrow 2.00}$
Model Size: 30B-A3B											
No-Train	67.08	69.63	82.4	69.2	82.1	82.5	64.6	75.4	61.2	75.3	72.94
GRPO	**68.32**	**70.77**	**88.0**	**78.6**	**88.9**	**83.3**	**73.6**	**79.6**	**70.4**	**78.7**	**78.02**
FineStep (Our)	**69.23**$_{\uparrow 0.91}$	**73.30**$_{\uparrow 2.53}$	**87.6**$_{\downarrow 0.4}$	**81.2**$_{\uparrow 2.6}$	**89.5**$_{\uparrow 0.6}$	**84.1**$_{\uparrow 0.8}$	**76.0**$_{\uparrow 2.4}$	**80.8**$_{\uparrow 1.2}$	**72.6**$_{\uparrow 2.2}$	**81.9**$_{\uparrow 3.2}$	**79.62**$_{\uparrow 1.60}$

Table 1: Overall performance of different algorithms. Subscripts indicate improvements over GRPO. The best results are highlighted in **bold**, and the second-best in **gray bold**. The same notation is used in Tables 3 and 4.

et al., 2025), GSPO (Zheng et al., 2025), and GIGPO (Feng et al., 2025). Among them, GIGPO clusters intermediate reasoning trajectories to normalize process-level advantages, while the others compute advantages solely from outcome rewards.

Implementation Details We evaluate three models: Qwen3-4B-Instruction, Qwen3-8B, and Qwen3-30B-A3B, representing an instruction model, a base model, and an MoE model, respectively. All RL algorithms are implemented in the Verl (Sheng et al., 2024) framework with a batch size of 128, 5 epochs, and a group size of 8. Database execution tools are implemented using SQLite (SQLite, n.d.) and FastAPI (Tiangolo, 2024). Unless otherwise specified, all results are obtained by sampling 8 responses with temperature $T = 1$ and selecting the final prediction via self-consistency. For FineStep, the smoothing coefficient β and advantage weight λ are both set to 0.5, discount factor γ is set to 0.98. More implementation details are provided in Appendix A.3.

4.1 Overall Performance

Cross-Model Adaptability Table 1 reports the performance of different RL algorithms on the BIRD and Spider benchmarks across three model configurations (4B, 8B, and 30B-A3B). FineStep consistently achieves the best average performance across all model architectures. Compared with GRPO, FineStep improves the average score by +3.25%, +2.00%, and +1.60% on the 4B, 8B, and 30B-A3B models, respectively. These results demonstrate that FineStep adapts well to different model scales and architectures, including instruction-tuned, base, and MoE models.

Comparison with Baselines FineStep consistently outperforms existing RL methods (GRPO, DAPO, GSPO, and GIGPO) on most evaluation metrics. By introducing step-level credit assignment, FineStep provides denser supervision for long-chain SQL reasoning. This design mitigates cases where correct final execution masks flawed intermediate reasoning, leading to reliable result.

4.2 Process Reward Behavior

Figure 5 shows the relationship between hard constraint violation rate and recall-based process reward across RL algorithms. FineStep consistently occupies the upper-left region across both 4B and 8B scales, achieving the lowest violation rates (0.018 and 0.014) and the highest recall-based rewards (1.739 and 1.710). In contrast, outcome-only methods cluster in the lower-left with higher violations and lower rewards. These results confirm that outcome-only optimization cannot ensure valid reasoning trajectories, while FineStep shifts the distribution toward more reliable paths.

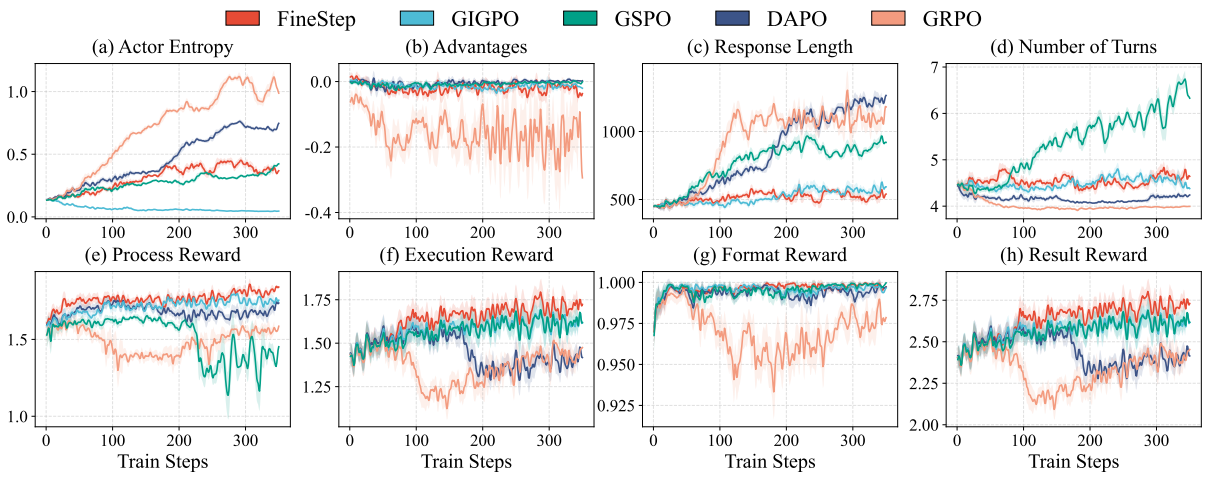

Figure 4: Comparison of indicators of different algorithms during RL training.

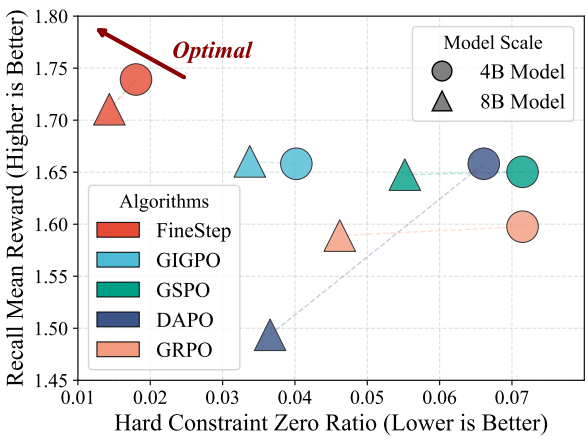

Figure 5: Comparison of process rewards on Bird Dev.

Configuration on Bird Dev	EX (%)	VES (%)
GRPO (Baseline)	66.56	68.13
FineStep		
w/ *Precision-based Reward*	66.95	69.24
w/ *F1-based Reward*	66.82	69.28
w/ *Recall-based Reward*	**68.06**	**70.90**

Table 2: Comparison of soft reward on FineStep-8B.

4.3 Training Dynamics Analysis

Figure 4 illustrates the training dynamics of different RL algorithms. FineStep exhibits more stable optimization and consistently higher reward signals. As shown in Fig. 4(a), FineStep maintains moderate actor entropy, avoiding the entropy collapse in GSPO and the entropy explosion in GRPO. In terms of reasoning efficiency, Fig. 4(c,d) shows that FineStep produces more concise trajectories with fewer interaction turns, whereas GSPO tends to generate longer responses with redundant tool calls. FineStep also achieves higher process and execution rewards with near-perfect formatting (Fig. 4(e,f,g,h)), while baselines such as DAPO and GRPO suffer reward collapse in later training stages. Overall, these results indicate that FineStep shifts the reasoning trajectory distribution toward more reliable and informative paths.

4.4 Effect of Soft Reward Design

Table 2 shows that the recall-based soft reward achieves the best performance. This advantage stems from the nature of multi-step Text-to-SQL reasoning, where intermediate execution results often form supersets of the final answer. In such cases, recall captures whether useful information is preserved during intermediate steps. By contrast, precision-based and F1-based rewards tend to penalize exploratory yet informative intermediate results.

4.5 Decoding Strategy Comparison

Table 3 compares decoding strategies on BIRD Dev. FineStep achieves the best performance across all settings and model scales. Under greedy decoding, it matches or surpasses the strongest baselines, and further improves Voting@8 and Pass@8 with multiple samples (e.g., +2.28% and +2.81% on 4B over GRPO), indicating a higher-quality candidate distribution.

4.6 Performance Across Difficulty Levels

Table 4 reports EX across difficulty levels on BIRD Dev. FineStep consistently achieves the best overall performance, with the most pronounced gains

Method	BIRD Dev (%)		
	Greedy	Voting@8	Pass@8
Model Size: 4B			
Training-free	63.95	65.58	73.53
GRPO	65.78	66.23	71.77
DAPO	66.75	67.67	72.29
GSPO	**67.01**	68.06	74.05
GIGPO	66.82	67.99	73.08
FineStep (Our)	**67.01**$_{\uparrow 1.23}$	**68.51**$_{\uparrow 2.28}$	**74.58**$_{\uparrow 2.81}$
Model Size: 8B			
Training-free	56.13	62.71	69.43
GRPO	64.60	66.56	73.73
DAPO	63.10	65.97	73.21
GSPO	64.86	66.62	73.79
GIGPO	65.19	66.88	73.53
FineStep (Our)	**65.78**$_{\uparrow 1.18}$	**68.06**$_{\uparrow 1.50}$	**74.64**$_{\uparrow 0.91}$

Table 3: Comparison of decoding strategies.

Method	Easy	Mid	Hard	Total
Model Size: 4B				
No-Train	71.24	59.48	48.97	65.58
GRPO	71.68	59.48	53.10	66.23
DAPO	73.73	59.05	56.55	67.67
GSPO	73.19	**60.99**	57.93	68.06
GIGPO	73.95	59.48	57.24	67.99
FineStep (Our)	**74.16**$_{\uparrow 2.48}$	60.13$_{\uparrow 0.65}$	**59.31**$_{\uparrow 6.21}$	**68.51**$_{\uparrow 2.28}$
Model Size: 8B				
No-Train	70.27	54.09	42.07	62.71
GRPO	73.41	58.19	49.66	66.56
DAPO	72.86	56.90	51.03	65.97
GSPO	73.19	57.76	53.10	66.62
GIGPO	73.41	58.84	51.03	66.88
FineStep (Our)	**74.05**$_{\uparrow 0.64}$	**58.84**$_{\uparrow 0.65}$	**59.31**$_{\uparrow 9.65}$	**68.06**$_{\uparrow 1.50}$

Table 4: Difficulty analysis on Bird Dev.

on hard queries (+6.21% on 4B, +9.65% on 8B over GRPO), where multi-step reasoning requires reliable intermediate decisions.

4.7 Ablation Study

Table 5 presents an ablation study of FineStep, examining three components: reward design, reward propagation, and advantage composition.

Effect of Reward Design. Removing either the hard constraint reward I_{hard} or the soft reward r_{soft} degrades performance. The larger drop without r_{soft} suggests that recall-based process reward provides the primary learning signal for intermediate reasoning, while I_{hard} mainly stabilizes tool usage by preventing invalid or repetitive actions.

Effect of Reward Propagation. Removing outcome reward discounting ($\gamma = 1$) or process reward smoothing ($\beta = 0$) reduces performance, indicating that both mechanisms help stabilize credit assignment across long reasoning trajectories.

Method / Ablation	Greedy	Voting@8	Pass@8
FineStep	**67.01**	**68.51**	**74.58**
w/o Hard Reward I_{hard}	66.95$_{\downarrow 0.06}$	67.41$_{\downarrow 1.10}$	73.86$_{\downarrow 0.72}$
w/o Soft Reward r_{soft}	66.17$_{\downarrow 0.84}$	67.21$_{\downarrow 1.30}$	72.69$_{\downarrow 1.89}$
No discounting ($\gamma = 1$)	62.67$_{\downarrow 4.34}$	66.62$_{\downarrow 1.89}$	73.01$_{\downarrow 1.57}$
No smoothing ($\beta = 0$)	64.93$_{\downarrow 2.08}$	65.84$_{\downarrow 2.67}$	72.75$_{\downarrow 1.83}$
Only $\hat{A}^{\text{out}}_{i,t}$ ($\lambda = 1$)	65.19$_{\downarrow 1.82}$	65.97$_{\downarrow 2.54}$	72.75$_{\downarrow 1.83}$
Only $\hat{A}^{\text{proc}}_{i,t}$ ($\lambda = 0$)	66.43$_{\downarrow 0.58}$	67.41$_{\downarrow 1.10}$	73.21$_{\downarrow 1.37}$

Table 5: Ablation of FineStep-4B on BIRD Dev.

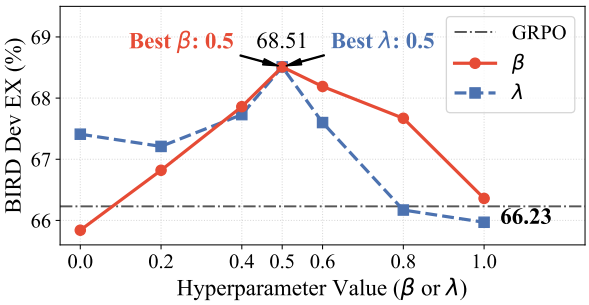

Figure 6: Sensitivity analysis of FineStep-4B.

Effect of Advantage Composition. Using only outcome advantage ($\lambda = 1$) or only process advantage ($\lambda = 0$) both harms performance, showing that outcome supervision and process supervision are complementary: the former provides global task-level guidance, while the latter supplies fine-grained signals for intermediate reasoning steps.

4.8 Hyperparameter Sensitivity

Figure 6 analyzes the effects of β and λ. Performance peaks at $\beta = 0.5$ and $\lambda = 0.5$, while FineStep consistently outperforms GRPO across a wide range of values, demonstrating that improvements stem from the overall design rather than specific hyperparameter choices. More details in Appendix A.9.

4.9 Effect of Sampling Size

Figure 7 analyzes the effect of inference sample size on execution accuracy. The upper bound (Pass@k) steadily improves with more samples, but self-consistency voting saturates quickly. For example, increasing from 8 to 72 samples raises Pass@k from 74.6% to 79.7% on 4B, while voting accuracy only improves from 68.5% to 68.8%.

4.10 Inference Overhead Analysis

Table 6 compares inference overhead under self-consistency decoding. FineStep achieves the high-

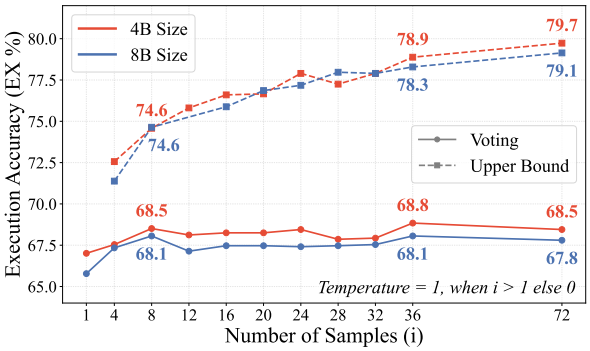

Figure 7: The impact of sample numbers on Bird Dev.

Method	EX(%)	Tool Calls	Avg. Time (s)	Resp. Len.
GRPO	66.23	8.71	1.064	6728
DAPO	67.67	8.83	1.295	8688
GSPO	68.06	29.27	2.981	5856
GIGPO	67.99	9.84	1.174	4416
FineStep	**68.51**	9.73	1.162	4328

Table 6: Inference overhead under self-consistency (8 samples) on BIRD Dev with Qwen3-4B.

est EX while maintaining tool calls, inference time, and response length comparable to GIGPO and substantially lower than GSPO (e.g., 9.73 vs. 29.27 tool calls, 1.162s vs. 2.981s). This confirms that FineStep's gains come from more efficient reasoning rather than increased test-time computation. Additional overhead analysis including greedy decoding and training cost is provided in Appendix A.6.

4.11 Generalizability Beyond Text-to-SQL

To evaluate whether FineStep's step-level credit assignment generalizes beyond Text-to-SQL, we conduct a preliminary experiment on AIME24, a mathematical reasoning benchmark, using Qwen2.5-7B-Instruct with a Python execution tool for interactive reasoning.

Method	Pass@1 Accuracy
No-Train	16.7
GRPO	33.3
FineStep	**34.6**

Table 7: Results on AIME24 (Qwen2.5-7B-Instruct).

Since AIME24 lacks SQL-specific execution feedback, we adapt FineStep by retaining only the hard-constraint process reward and outcome reward discounting, while disabling the recall-based soft reward and process smoothing. The outcome weight is increased to $\lambda = 0.8$ to reflect the weaker process signal. Despite minimal task-specific adaptation, FineStep still outperforms GRPO, demonstrating that the core step-level credit assignment mechanism transfers to non-SQL tool-integrated reasoning. Detailed adaptation and analysis are provided in Appendix A.8.

Method	Process Signal	EX (%)	Train Time
GRPO	None	66.23	35h
LLM-as-Judge	Qwen3-235B	64.57	81h
FineStep (Ours)	Rule-based	**68.51**	37h

Table 8: Comparison with PRM-based baseline on BIRD Dev (Qwen3-4B). Details in Appendix A.7.

4.12 Comparison with PRM-based Approach

We compare FineStep with an LLM-as-Judge baseline (Qwen3-235B scoring each step, details in Appendix A.7). As shown in Table 8, LLM-as-Judge underperforms GRPO (64.57% vs. 66.23%) while requiring over 2× training time (81h vs. 37h), suggesting that general-purpose LLM scoring introduces noise in tool-integrated reasoning. FineStep's rule-based rewards directly leverage structured execution feedback, achieving superior performance without additional model overhead.

5 Conclusion

We propose **FineStep**, a step-level credit assignment framework for tool-augmented reasoning that constructs training-free process rewards from structured tool execution feedback. Through a coherent pipeline of dual-channel signal construction, temporal-logical value propagation, and group-based advantage synthesis, FineStep precisely quantifies each reasoning step's contribution without auxiliary reward model training. Experiments on Text-to-SQL benchmarks show that FineStep achieves state-of-the-art accuracy while reducing redundant tool interactions and stabilizing RL training. Comparisons with an LLM-as-Judge baseline further confirm that rule-based execution feedback outperforms general-purpose LLM scoring in tool-integrated settings. Preliminary results on AIME24 demonstrate transferability beyond SQL, suggesting that step-level credit assignment is a general principle for long-horizon agentic reasoning.

Limitation

Although FineStep's core credit assignment mechanism (outcome discounting, step-level normaliza-

tion) is task-agnostic as shown by the AIME24 experiment, the recall-based soft reward is most effective in domains with structured execution feedback (e.g., SQL, code). Extending this design to domains with less structured feedback remains future work. Additionally, the rule-based reward design ensures high precision without training but may lack flexibility for ambiguous reasoning paths. Combining rule-based signals with learned reward models and conducting broader multi-domain evaluation (e.g., web agents, code generation) are promising directions.

References

Yuhan Chen, Yuxuan Liu, Long Zhang, Pengzhi Gao, Jian Luan, and Wei Liu. 2025. Step: Success-rate-aware trajectory-efficient policy optimization. *Preprint*, arXiv:2511.13091.

Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Yuchen Zhang, Jiacheng Chen, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, Qixin Xu, Weize Chen, Jiarui Yuan, Huayu Chen, Kaiyan Zhang, Xingtai Lv, Shuo Wang, Yuan Yao, Xu Han, and 6 others. 2025. Process reinforcement through implicit rewards. *Preprint*, arXiv:2502.01456.

Yaxun Dai, Wenxuan Xie, Xialie Zhuang, Tianyu Yang, Yiying Yang, Haiqin Yang, Yuhang Zhao, Pingfu Chao, and Wenhao Jiang. 2025. Reex-sql: Reasoning with execution-aware reinforcement learning for text-to-sql. *CoRR*, abs/2505.12768.

Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2021. Structure-grounded pretraining for text-to-sql. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 1337–1350. Association for Computational Linguistics.

Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, Guorui Zhou, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. 2025. Agentic reinforced policy optimization. *Preprint*, arXiv:2507.19849.

Kaixuan Fan, Kaituo Feng, Manyuan Zhang, Tianshuo Peng, Zhixun Li, Yilei Jiang, Shuang Chen, Peng Pei, Xunliang Cai, and Xiangyu Yue. 2026. Exploring reasoning reward model for agents. *Preprint*, arXiv:2601.22154.

Yuankai Fan, Tonghui Ren, Can Huang, Zhenying He, and X. Sean Wang. 2024. Grounding natural language to SQL translation with data-based self-explanations. *CoRR*, abs/2411.02948.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. 2025. Group-in-group policy optimization for LLM agent training. *CoRR*, abs/2505.10978.

Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R. Woodward, Jinxia Xie, and Pengsheng Huang. 2021a. Towards robustness of text-to-sql models against synonym substitution. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 2505–2515. Association for Computational Linguistics.

Yujian Gan, Xinyun Chen, and Matthew Purver. 2021b. Exploring underexplored limitations of cross-domain text-to-sql generalization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 8926–8931. Association for Computational Linguistics.

Yingqi Gao, Yifu Liu, Xiaoxia Li, Xiaorong Shi, Yin Zhu, Yiming Wang, Shiqi Li, Wei Li, Yuntao Hong, Zhiling Luo, Jinyang Gao, Liyu Mou, and Yu Li. 2024. Xiyan-sql: A multi-generator ensemble framework for text-to-sql. *CoRR*, abs/2411.08599.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, and 175 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638.

Yuxiang Ji, Ziyu Ma, Yong Wang, Guanhua Chen, Xiangxiang Chu, and Liaoni Wu. 2025. Tree search for llm agent reinforcement learning. *Preprint*, arXiv:2509.21240.

Yonghui Kong, Hongbing Hu, Dan Zhang, Siyuan Chai, Fan Zhang, and Wei Wang. 2025. Bridging the gap: Transforming natural language questions into SQL queries via abstract query pattern and contextual schema markup. *CoRR*, abs/2502.14682.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. 2025. MCS-SQL: leveraging multiple prompts and multiple-choice selection for text-to-sql generation. In *COLING*, pages 337–353. Association for Computational Linguistics.

Boyan Li, Jiayi Zhang, Ju Fan, Yanwei Xu, Chong Chen, Nan Tang, and Yuyu Luo. 2025a. Alpha-sql: Zero-shot text-to-sql using monte carlo tree search. *CoRR*, abs/2502.17248.

Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, Hong Chen, and Cuiping Li. 2025b. Omnisql: Synthesizing high-quality text-to-sql data at scale. *CoRR*, abs/2503.02240.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024. Codes: Towards building open-source language models for text-to-sql. *Proc. ACM Manag. Data*, 2(3):127.

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

Xuefeng Li, Haoyang Zou, and Pengfei Liu. 2025c. Torl: Scaling tool-integrated rl. *Preprint*, arXiv:2503.23383.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Shu Liu, Sumanth Hegde, Shiyi Cao, Alan Zhu, Dacheng Li, Tyler Griggs, Eric Tang, Akshay Malik, Kourosh Hakhamaneshi, Richard Liaw, Philipp Moritz, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. 2025. Skyrl-sql: Matching gpt-4o and o4-mini on text2sql with multi-turn rl.

Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuyu Luo, Yuxin Zhang, Ju Fan, Guoliang Li, and Nan Tang. 2024. A survey of NL2SQL with large language models: Where are we, and where are we going? *CoRR*, abs/2408.05109.

Shuai Lyu, Haoran Luo, Zhonghong Ou, Yifan Zhu, Xiaoran Shang, Yang Qin, and Meina Song. 2025. Sql-o1: A self-reward heuristic dynamic search method for text-to-sql. *CoRR*, abs/2502.11741.

Peixian Ma, Xialie Zhuang, Chengjin Xu, Xuhui Jiang, Ran Chen, and Jian Guo. 2025. SQL-R1: training natural language to SQL reasoning model by reinforcement learning. *CoRR*, abs/2504.08600.

Xinji Mai, Haotian Xu, Zhong-Zhi Li, Xing W, Weinong Wang, Jian Hu, Yingying Zhang, and Wenqiang Zhang. 2025. Agent rl scaling law: Agent rl with spontaneous code execution for mathematical problem solving. *Preprint*, arXiv:2505.07773.

Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O Arik. 2024. Chase-sql: Multi-path reasoning and preference optimized candidate selection in text-to-sql. *arXiv preprint arXiv:2410.01943*.

Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: decomposed in-context learning of text-to-sql with self-correction. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Mohammadreza Pourreza, Shayan Talaei, Ruoxi Sun, Xingchen Wan, Hailong Li, Azalia Mirhoseini, Amin Saberi, and Sercan Ö. Arik. 2025. Reasoning-sql: Reinforcement learning with SQL tailored partial rewards for reasoning-enhanced text-to-sql. *CoRR*, abs/2503.23157.

Yang Qin, Chao Chen, Zhihang Fu, Ze Chen, Dezhong Peng, Peng Hu, and Jieping Ye. 2024. ROUTE: robust multitask tuning and collaboration for text-to-sql. *CoRR*, abs/2412.10138.

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

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. 2024. Rewarding progress: Scaling automated process verifiers for llm reasoning. *Preprint*, arXiv:2410.08146.

Ning Shang, Yifei Liu, Yi Zhu, Li Lyna Zhang, Weijiang Xu, Xinyu Guan, Buze Zhang, Bingcheng Dong, Xudong Zhou, Bowen Zhang, Ying Xin, Ziming Miao, Scarlett Li, Fan Yang, and Mao Yang. 2025. rstar2-agent: Agentic reasoning technical report. *Preprint*, arXiv:2508.20722.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, abs/2402.03300.

Zhili Shen, Pavlos Vougiouklis, Chenxin Diao, Kaustubh Vyas, Yuanyi Ji, and Jeff Z. Pan. 2024. Improving retrieval-augmented text-to-sql with ast-based ranking and schema pruning. In *EMNLP*, pages 7865–7879. Association for Computational Linguistics.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2024. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*.

Joykirat Singh, Raghav Magazine, Yash Pandya, and Akshay Nambi. 2025. Agentic reasoning and tool integration for llms via reinforcement learning. *Preprint*, arXiv:2505.01441.

SQLite. n.d. Sqlite home page. Accessed: 2025-05-10.

Ruoxi Sun, Sercan Ö. Arik, Hootan Nakhost, Hanjun Dai, Rajarishi Sinha, Pengcheng Yin, and Tomas Pfister. 2024. Sql-palm: Improved large language model adaptation for text-to-sql. *CoRR*, abs/2306.00739.

Sebastián Ramírez Tiangolo. 2024. Fastapi: A modern, high-performance, python web framework. GitHub repository.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2024. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 9426–9439. Association for Computational Linguistics.

Pengfei Wang, Baolin Sun, Xuemei Dong, Yaxun Dai, Hongwei Yuan, Mengdie Chu, Yingqi Gao, Xiang Qi, Peng Zhang, and Ying Yan. 2025. Agentar-scale-sql: Advancing text-to-sql through orchestrated test-time scaling. *CoRR*, abs/2509.24403.

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

Yu Xia, Jingru Fan, Weize Chen, Siyu Yan, Xin Cong, Zhong Zhang, Yaxi Lu, Yankai Lin, Zhiyuan Liu, and Maosong Sun. 2025. AgentRM: Enhancing agent generalization with reward modeling. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 19277–19290, Vienna, Austria. Association for Computational Linguistics.

Zekun Xu, Siyu Xia, Chuhuai Yue, Jiajun Chai, Mingxue Tian, Xiaohan Wang, Wei Lin, Haoxuan Li, and Guojun Yin. 2025. MTIR-SQL: multi-turn tool-integrated reasoning reinforcement learning for text-to-sql. *CoRR*, abs/2510.25510.

Zhenghai Xue, Longtao Zheng, Qian Liu, Yingru Li, Xiaosen Zheng, Zejun Ma, and Bo An. 2025. Simpletir: End-to-end reinforcement learning for multi-turn tool-integrated reasoning. *Preprint*, arXiv:2509.02479.

Haolin Yang, Jipeng Zhang, Zhitao He, and Yi R. Fung. 2025. MARS-SQL: A multi-agent reinforcement learning framework for text-to-sql. *CoRR*, abs/2511.01008.

Jiaxi Yang, Binyuan Hui, Min Yang, Jian Yang, Junyang Lin, and Chang Zhou. 2024. 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), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 7864–7875. Association for Computational Linguistics.

Zhewei Yao, Guoheng Sun, Lukasz Borchmann, Zheyu Shen, Minghang Deng, Bohan Zhai, Hao Zhang, Ang Li, and Yuxiong He. 2025. Arctic-text2sql-r1: Simple rewards, strong reasoning in text-to-sql. *CoRR*, abs/2505.20315.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, and 16 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. *Preprint*, arXiv:2503.14476.

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

Shuozhi Yuan, Liming Chen, Miaomiao Yuan, Jin Zhao, Haoran Peng, and Wenming Guo. 2025. MCTS-SQL: an effective framework for text-to-sql with monte carlo tree search. *CoRR*, abs/2501.16607.

Guibin Zhang, Hejia Geng, Xiaohang Yu, Zhenfei Yin, Zaibin Zhang, Zelin Tan, Heng Zhou, Zhongzhi Li, Xiangyuan Xue, Yijiang Li, Yifan Zhou, Yang Chen, Chen Zhang, Yutao Fan, Zihu Wang, Songtao Huang, Francisco Piedrahita-Velez, Yue Liao, Hongru Wang, and 6 others. 2026. The landscape of agentic reinforcement learning for llms: A survey. *Preprint*, arXiv:2509.02547.

Weixu Zhang, Yifei Wang, Yuanfeng Song, Victor Junqiu Wei, Yuxing Tian, Yiyan Qi, Jonathan H. Chan, Raymond Chi-Wing Wong, and Haiqin Yang. 2024. Natural language interfaces for tabular data querying and visualization: A survey. *IEEE Trans. Knowl. Data Eng.*

Yuxin Zhang, Meihao Fan, Ju Fan, Mingyang Yi, Yuyu Luo, Jian Tan, and Guoliang Li. 2025. Reward-sql: Boosting text-to-sql via stepwise reasoning and process-supervised rewards. *Preprint*, arXiv:2505.04671.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong

Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. 2025. Group sequence policy optimization. *Preprint*, arXiv:2507.18071.

Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic evaluation for text-to-sql with distilled test suites. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 396–411. Association for Computational Linguistics.

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

A Appendices

A.1 Benchmarks

In this study, we evaluate our method on several representative Text-to-SQL benchmarks.

BIRD (Li et al., 2023) is a large-scale cross-domain Text-to-SQL benchmark designed for realistic applications. It contains 12,751 natural language questions paired with SQL queries across 95 databases from 37 domains, with a total data scale of 33.4 GB. Compared with earlier datasets such as Spider (Yu et al., 2018) and WikiSQL (Zhong et al., 2017), BIRD places greater emphasis on database contents (e.g., field values and real data distributions), making it closer to real-world query scenarios.

Spider (Yu et al., 2018) is a widely used Text-to-SQL benchmark consisting of 8,659 training examples, 1,034 development examples, and 2,147 test examples across 200 databases from 138 domains. It adopts a cross-database evaluation setting, where models must generalize to unseen database schemas.

Spider-DK (Gan et al., 2021b), **Spider-Syn** (Gan et al., 2021a), and **Spider-Realistic** (Deng et al., 2021) are three variants derived from Spider to better simulate real-world queries. Spider-DK introduces domain knowledge and semantic distractions to increase reasoning difficulty, Spider-Syn evaluates robustness to paraphrasing and syntactic variations, and Spider-Realistic reformulates queries into more conversational and incomplete forms.

Table 9 summarizes the statistics of the Spider and BIRD datasets. In general, SQL queries in BIRD are more complex than those in Spider, reflecting its closer alignment with real-world database applications.

A.2 Reinforcement Learning Baselines

This appendix briefly introduces the reinforcement learning (RL) baseline algorithms used in this study. All selected methods follow a critic-free or group-relative optimization paradigm, which removes the need for an auxiliary value network and improves training efficiency for large reasoning models.

A.2.1 Group Relative Policy Optimization

Group Relative Policy Optimization (GRPO) (Shao et al., 2024), introduced in the DeepSeekMath framework, replaces the traditional critic with group-based advantage estimation. Given a prompt

	Examples	Databases	Tables	Domains	Rows
BIRD	12,751	92	7.3	37	549k
Spider	10,181	200	5.1	138	2k

Table 9: Statistics of the BIRD and Spider benchmarks

q, the policy samples a group of G outputs $\{o_1, \ldots, o_G\}$. The advantage $\hat{A}_i$ of each output is computed by normalizing the rewards $\{r_1, \ldots, r_G\}$ within the group:

$$\hat{A}_i = \frac{r_i - \text{mean}(r_1, \ldots, r_G)}{\text{std}(r_1, \ldots, r_G)}. \tag{11}$$

The policy is optimized using a clipped surrogate objective with a KL-divergence regularization term to stabilize training.

A.2.2 Decoupled Clip and Dynamic Sampling Policy Optimization

DAPO (Yu et al., 2025) extends GRPO with several techniques designed to improve exploration and sampling efficiency in long chain-of-thought (CoT) reasoning. The main components include:

1. **Clip-Higher.** An asymmetric clipping mechanism where the upper clipping bound ϵ_{high} (e.g., 0.28) is larger than the lower bound ϵ_{low} (e.g., 0.20), allowing more aggressive updates for high-reward samples.

2. **Dynamic Sampling.** Groups with zero reward variance are filtered out during training, ensuring that computation focuses on samples providing meaningful gradient signals.

3. **Token-Level Loss.** Gradients are aggregated at the token level to mitigate the bias toward shorter responses that may arise in long reasoning trajectories.

A.2.3 Group Sequence Policy Optimization

Group Sequence Policy Optimization (GSPO) (Zheng et al., 2025) addresses training instability in large Mixture-of-Experts (MoE) models. Instead of token-level optimization, GSPO operates at the sequence level. The importance ratio $s_i(\theta)$ is computed using length-normalized sequence likelihood:

$$s_i(\theta) = \left(\frac{\pi_\theta(y_i \mid x)}{\pi_{\theta_{\text{old}}}(y_i \mid x)} \right)^{\frac{1}{|y_i|}}. \tag{12}$$

By shifting the optimization granularity from tokens to sequences, GSPO reduces the high variance introduced by token-level importance sampling in MoE architectures and improves training stability.

Parameter	Value
Hardware Configuration	
GPU (4B / 8B)	8 × A100 (80GB)
GPU (30B-A3B MoE)	32 × A100 (80GB)
Common Training Parameters	
Max prompt length	8192
Max response length	3072
Max tool response length	256
Responses per prompt (n_{resp})	8
Training prompt batch size	128
Max assistant turns	10
Learning rate	1×10^{-6}
Rollout temperature	1.0
Loss aggregation mode	token-mean
Rollout Type	vLLM (Kwon et al., 2023)
GRPO	
Clip ratio	0.2
KL regularization	Enabled
DAPO	
Clip ratio (low, high)	(0.2, 0.28)
Overlong buffer length	2048
Dynamic sampling	Enabled
Token-level loss	Enabled
GSPO	
Optimization granularity	Sequence-level
Length normalization	Enabled
GIGPO	
Hierarchical advantage	Enabled
Step weight (ω)	1.0
FineStep (Ours)	
Step advantage weight (λ)	0.5
Reward smoothing (β)	0.5

Table 10: Training configuration and algorithm-specific hyperparameters.

A.2.4 Group-in-Group Policy Optimization

Group-in-Group Policy Optimization (GIGPO) (Feng et al., 2025) introduces a hierarchical advantage structure for fine-grained credit assignment in multi-turn LLM agents. It groups trajectories that visit identical environment states (referred to as anchor states) and computes advantages using both global and local signals. The final advantage is defined as

$$A(a_t^{(i)}) = A_E(\tau_i) + \omega A_S(a_t^{(i)}), \tag{13}$$

where A_E denotes the episode-level relative advantage capturing task completion, and A_S denotes the step-level relative advantage measuring local action effectiveness. However, both signals are still derived from outcome-level rewards and do not explicitly evaluate the utility of intermediate reasoning steps. As a result, compared with methods

Methods	BIRD Dev		Spider Dev		Spider Test
	EX	VES	EX	TS	EX
Closed-source methods					
XiYan-SQL (Gao et al., 2024)	**73.3**	-	-	-	**89.6**
CHASE-SQL (Pourreza et al., 2024)	73.0	**73.0**	-	-	87.6
MCTS-SQL + GPT-4 (Yuan et al., 2025)	69.4	66.2	88.7	-	86.6
PAS-SQL + GPT-4o (Kong et al., 2025)	64.7	65.0	87.9	-	86.8
MCS-SQL + GPT-4 (Lee et al., 2025)	63.4	64.8	**89.5**	-	**89.6**
Training base methods					
SFT CodeS-7B (Li et al., 2024)	57.2	58.8	85.4	80.3	-
OmniSQL-7B (Li et al., 2025b)	66.1	-	81.6	-	88.9
Reasoning-SQL (Pourreza et al., 2025)	64.0	-	-	-	78.7
SQl-R1-7B (Ma et al., 2025)	63.1	-	84.5	-	86.1
Arcitc-Text2SQL-R1 (Yao et al., 2025)	64.8	-	-	-	87.1
SQL-o1-8B (Lyu et al., 2025)	63.4	64.7	87.4	79.6	85.4
ReEx-SQL-7B (Dai et al., 2025)	65.3	**73.3**	89.1	83.9	86.9
MTIR-SQL + Qwen3-14B (Xu et al., 2025)	68.1	-	86.7	-	87.2
Our method					
FineStep-4B	**68.51**	71.65	**90.3**	**84.3**	**89.9**
FineStep-8B	68.06	70.90	88.9	82.4	**89.9**

Table 11: Performance comparison on BIRD and Spider benchmarks (%).

Methods	Syn		Realistic		DK
	EX	TS	EX	TS	EX
Closed-source methods					
FastRAT$_{ext}$+GPT-4 (Shen et al., 2024)	74.4	-	**80.9**	-	72.3
TA-SQL + GPT-4 (Qu et al., 2024)	-	-	79.5	-	**72.9**
CYCLESQL + GPT-4 (Fan et al., 2024)	**76.0**	**66.3**	70.6	**56.9**	68.5
SQL-PaLM + PaLM2 (Sun et al., 2024)	74.6	-	77.6	-	66.5
Training base methods					
ROUTE + Llama3-8B (Qin et al., 2024)	77.4	70.2	80.9	72.6	74.6
SFT CodeS-15B (Li et al., 2024)	77.0	69.4	83.1	75.6	70.7
SENSE-13B (Yang et al., 2024)	77.6	70.2	84.1	76.6	80.2
OmniSQL-32B (Li et al., 2025b)	72.1	-	77.2	-	77.6
Reasoning-SQL* (Pourreza et al., 2025)	69.3	-	-	-	73.3
SQl-R1-7B* (Ma et al., 2025)	76.7	-	83.3	-	78.1
SQL-o1 + Llama3-8B (Lyu et al., 2025)	77.6	69.2	82.7	72.8	78.7
ReEx-SQL-7B (Dai et al., 2025)	78.7	72.0	85.5	79.9	80.2
MTIR-SQL + Qwen3-14B (Xu et al., 2025)	81.0	-	81.1	-	76.3
Our method					
FineStep-4B	**83.1**	**75.2**	**88.6**	**80.9**	**81.5**
FineStep-8B	81.4	71.6	86.0	77.9	80.0

Table 12: Performance comparison on Spider variants robustness benchmarks (%).

such as FineStep that directly model process-level rewards, the credit assignment in GIGPO remains relatively coarse-grained.

A.3 More experimental Configuration.

All RL algorithms are implemented using the `verl` (Sheng et al., 2024) framework under a unified training setup. For the 4B and 8B models, experiments are conducted on 8 NVIDIA A100 GPUs (80GB), while the MoE model (30B-A3B) is trained on 32 NVIDIA A100 GPUs (80GB). During inference, we fix the random seed for vLLM to ensure reproducibility under both greedy decoding and sampling settings. Implementation details can be found in Table 10.

A.4 Comparison with Existing Methods

Tables 11 and 12 compare FineStep with a wide range of existing Text-to-SQL systems on both in-domain and out-of-domain benchmarks.

In-domain performance. As shown in Table 11, FineStep achieves competitive results against both closed-source and training-based approaches on the BIRD and Spider benchmarks. Notably, FineStep-4B achieves 90.3% EX and 84.3% TS on Spider Dev, outperforming many existing methods despite using a significantly smaller model. On Spider Test, both FineStep-4B and FineStep-8B reach 89.9% EX, which is competitive with several GPT-4-based systems. On the BIRD Dev, FineStep-4B also achieves strong performance, reaching 68.51% EX and 71.65% VES.

Out-of-domain robustness. Table 12 further evaluates model robustness on Spider variants, including Spider-Syn, Spider-Realistic, and Spider-DK. FineStep consistently achieves state-of-the-art or competitive performance across these benchmarks. For example, FineStep-4B obtains 81.4% EX on Spider-Syn and 86.0% EX on Spider-Realistic, outperforming many larger models. FineStep-8B further achieves the best performance on Spider-DK with 81.9% EX.

Overall, these results demonstrate that FineStep maintains strong performance not only on standard benchmarks but also under distribution shifts, highlighting its effectiveness in improving both reasoning accuracy and robustness.

A.5 Theoretical Analysis of FineStep

We provide a simple theoretical interpretation showing why trajectory-level outcome optimization may incorrectly reinforce useless intermediate step in successful trajectory, while FineStep can suppress them through step-level credit assignment.

Proposition 1 (Successful τ may contain useless a_k) *Consider a multi-step reasoning trajectory*

$$\tau = (s_0, a_0, s_1, \ldots, a_T, s_{T+1}) \tag{14}$$

with final outcome reward $R^{\text{out}}(\tau) = 1$. A step a_k is useless *if removing it does not change the final answer or outcome:*

$$R^{\text{out}}(\tau) = R^{\text{out}}(\tau \setminus a_k). \tag{15}$$

Such steps naturally exist in tool-integrated reasoning. For example, an SQL tool call may be syntactically valid but irrelevant to the final query,

or may repeat already obtained information. The trajectory can still be successful if later steps recover and produce the correct final SQL.

Proposition 2 (GRPO reinforces a_k in successful τ) *In trajectory-level GRPO, all steps in the same trajectory share the same normalized outcome advantage:*

$$A_{i,t}^{\text{GRPO}} = \frac{R_i^{\text{out}} - \text{Mean}(\{R_j^{\text{out}}\}_{j=1}^G)}{\text{Std}(\{R_j^{\text{out}}\}_{j=1}^G) + \epsilon}. \quad (16)$$

For a successful trajectory with above-average outcome reward, we have $A_{i,t}^{\text{GRPO}} > 0, \forall t$. The policy-gradient update for step t is proportional to

$$A_{i,t}^{\text{GRPO}} \nabla_\theta \log \pi_\theta(a_{i,t} \mid s_{i,t}). \quad (17)$$

Therefore, if $a_{i,k}$ is a useless step inside a successful trajectory, it still receives a positive update $A_{i,k}^{\text{GRPO}} > 0$. Thus, GRPO cannot distinguish useful steps from useless ones within a successful trajectory, and will incorrectly increase the probability of both.

Proposition 3 (FineStep can suppress useless a_k) *FineStep assigns each step its own advantage:*

$$A_{i,t}^{\text{FineStep}} = \lambda \hat{A}_{i,t}^{\text{out}} + (1-\lambda)\hat{A}_{i,t}^{\text{proc}}. \quad (18)$$

For a useless step $a_{i,k}$, its process value is low because it provides little or no information gain:

$$V_{i,k}^{\text{proc}} < \text{Mean}(\mathcal{V}^{\text{proc}}), \quad (19)$$

which leads to $\hat{A}_{i,k}^{\text{proc}} < 0$. FineStep suppresses this step when

$$\lambda \hat{A}_{i,k}^{\text{out}} + (1-\lambda)\hat{A}_{i,k}^{\text{proc}} < 0, \quad (20)$$

equivalently,

$$|\hat{A}_{i,k}^{\text{proc}}| > \frac{\lambda}{1-\lambda}\hat{A}_{i,k}^{\text{out}}. \quad (21)$$

Under this condition, the policy-gradient update for the useless step becomes negative:

$$A_{i,k}^{\text{FineStep}} \nabla_\theta \log \pi_\theta(a_{i,k} \mid s_{i,k}), \quad A_{i,k}^{\text{FineStep}} < 0, \quad (22)$$

which decreases the probability of the useless action.

This analysis shows that trajectory-level GRPO gives identical credit to all steps in a successful trajectory, including redundant or uninformative ones. In contrast, FineStep can assign lower or even negative advantages to useless steps by incorporating step-level process signals. Therefore, FineStep provides a more discriminative credit assignment mechanism for long-horizon tool-integrated reasoning.

Method	EX(%)	Tool Calls	Avg. Time (s)	Resp. Len.
GRPO	65.78	1.09	0.231	841.7
DAPO	66.75	1.10	0.276	1086.2
GSPO	67.01	3.66	0.421	732.4
GIGPO	66.82	1.23	0.252	552.2
FineStep	**67.01**	1.22	0.250	541.4

Table 13: Inference overhead under greedy decoding.

A.6 Computational Overhead Analysis

We demonstrate the computational overhead in both training and inference scenarios. Our results show that the gains of FineStep do not come from longer reasoning, more tool calls, or higher test-time computation, but from more efficient step-level credit assignment.

A.6.1 Training Overhead Analysis

From the training dynamics as shown in Figure 4, different RL methods exhibit distinct overhead:

- **GSPO** tends to produce more interaction turns, leading to substantially more tool calls.
- **GRPO** and **DAPO** rely more on longer reasoning text, resulting in larger response lengths.
- **GIGPO** and **FineStep** obtain external feedback through more stable tool usage and shorter reasoning traces.

By assigning credit at the step level, FineStep suppresses redundant reasoning and invalid tool interactions. As a result, it improves accuracy without noticeably increasing rollout overhead.

A.6.2 Inference Overhead Analysis

We further measure inference overhead on the BIRD Dev set using Qwen3-4B-Instruct as the base model. Table 13 reports results under greedy decoding, and Table 6 reports results under self-consistency with 8 samples. In both tables, Tool Calls, Avg. Time, and Resp. Len. denote the average number of tool calls, average inference time, and average response length per sample, respectively.

FineStep achieves the highest EX under self-consistency, while maintaining tool calls, inference time, and response length comparable to GIGPO and substantially lower than GSPO. Compared with GSPO, FineStep improves EX from 68.06 to 68.51, while reducing tool calls from 29.27 to 9.73 and average inference time from 2.981s to 1.162s. These results indicate that FineStep's improvements do

not rely on increased test-time computation, but instead come from more efficient and reliable tool-integrated reasoning.

A.7 LLM-as-Judge Baseline Details

To compare FineStep with PRM-based approaches, we implement an LLM-as-Judge baseline using Qwen3-235B as the step-level evaluator. For each reasoning step, the judge receives the full context (question, database schema, reasoning history up to the current step, and the ground truth SQL) and assigns a discrete score:

- **Score 0**: The step is invalid (SQL syntax error, execution error, or exact duplicate of a previous step).
- **Score 1**: The step is valid but provides limited information gain.
- **Score 2**: The step is valid and provides high information gain (execution result contains key information relevant to the correct answer).

Scores are normalized to $[0, 1]$ and used as step-level rewards. Advantage estimation follows step-level GRPO (Shao et al., 2024): all step-level rewards across the sampled group are normalized to compute per-step advantages. The judge is called after each RL rollout, introducing additional inference overhead proportional to the total number of reasoning steps. All other training settings (base model, batch size, epochs, group size) remain identical to FineStep for fair comparison.

A.8 Generalizability on Non-SQL Reasoning

This section provides detailed adaptation strategies for the generalizability experiment reported in the main text (Table 7).

Task Setup. We evaluate FineStep on AIME24, a mathematical reasoning benchmark requiring multi-step problem solving. The model (Qwen2.5-7B-Instruct) interacts with a Python execution tool for interactive reasoning, where each step involves generating Python code, executing it, and observing the output—analogous to the tool-integrated reasoning paradigm in Text-to-SQL but in a different domain.

Adaptation Strategy. Since AIME24 lacks SQL-specific execution feedback (e.g., intermediate query results for recall computation), we adapt FineStep with the following modifications:

1. **No SQL-specific soft reward.** The recall-based reward is not applicable without intermediate execution results that can be compared against a ground truth.
2. **Only hard-constraint process reward.** We retain format validity and execution success as step-level signals, transferring the credit assignment idea without task-specific reward shaping.
3. **No process smoothing ($\beta = 0$).** The process signal in AIME24 is binary (execution success/failure) and weaker than SQL feedback; smoothing may propagate noisy signals.
4. **Larger outcome weight ($\lambda = 0.8$).** Since final-answer correctness is more reliable than the weak process signal, we let the outcome reward dominate while keeping the process reward as an auxiliary constraint.

Analysis. Despite using only a minimal subset of FineStep's components (outcome discounting + hard constraint), the method still outperforms GRPO (34.6 vs. 33.3 Pass@1). This suggests that the core mechanism—step-level credit assignment via outcome reward discounting and group-based normalization—is task-agnostic and transferable. The domain-specific components (recall-based soft reward, process smoothing) can be viewed as optional enhancements when rich intermediate feedback is available. Broader multi-domain evaluation (e.g., web agents, multi-hop code generation) remains an important direction for future work.

A.9 Reward Robustness and Hyperparameter Sensitivity

FineStep's reward design relies only on general SQL executor feedback, without benchmark-specific rules or annotations. Therefore, its gains mainly come from step-level credit assignment rather than benchmark-specific reward tuning.

To further verify this, we conduct cross-dataset hyperparameter sensitivity analysis. Using Qwen3-4B-Instruct, we fix $\lambda = 0.5$ and vary β, and then fix $\beta = 0.5$ and vary λ. We report results on BIRD Dev, Spider Dev, and Spider-Realistic.

β	0	0.2	0.4	0.5	0.6	1
BIRD Dev	65.84	66.82	67.86	**68.51**	68.19	66.36
Spider Dev	87.3	88.4	89.9	**90.3**	89.7	87.2
Spider-Realistic	86.9	87.6	87.4	**88.6**	88.2	86.0

Table 14: EX (%) with varying β (fixed $\lambda = 0.5$).

λ	0	0.2	0.4	0.5	0.6	1
BIRD Dev	67.41	67.21	67.73	**68.51**	67.60	65.97
Spider Dev	88.2	88.3	89.5	**90.3**	88.9	87.6
Spider-Realistic	87.4	87.6	88.1	**88.6**	87.8	85.4

Table 15: EX (%) with varying λ (fixed $\beta = 0.5$).

The results show that $\beta = 0.5$ and $\lambda = 0.5$ achieve the best or near-best performance not only on BIRD Dev, but also on Spider Dev and Spider-Realistic. This demonstrates FineStep's robustness to reward design choices and its consistent cross-dataset generalization.

A.10 Prompt Design

FineStep Prompt (Figure 8) Following the paradigm of MTIR-SQL (Xu et al., 2025), we designed a multi-turn tool-integrated reasoning prompt to elicit structured and verifiable logic. This design facilitates an iterative "hypothesize-verify-refine" process by providing distinct modules for <reasoning>, tool-based validation, and final answer encapsulation. Such a mechanism effectively reduces hallucinations in complex SQL generation by forcing the model to validate its internal reasoning against the database engine before committing to a final output.

A.11 Case Study

Figure 9 presents a representative example from the BIRD development set (question_id = 29) to illustrate the behavioral differences between FineStep and GRPO. The middle block shows a successful reasoning trajectory produced by FineStep. With explicit step-level supervision, the model correctly identifies the relevant tables (`schools` and `frpm`) and generates a valid SQL query in a single interaction step, leading to the correct execution result.

In contrast, the bottom block shows a trajectory generated by a GRPO-trained model. Without explicit process-level guidance, the model initially hallucinates a nonexistent column and repeatedly attempts invalid queries, eventually falling back to an unrelated table (`satscores`). Although the final query executes successfully, it does not correspond to the correct reasoning path and produces an incorrect result.

This example highlights the advantage of FineStep in guiding models toward valid and informative intermediate steps, thereby avoiding hallucination loops and improving reasoning reliability.

Multi-Turn Tool-Integrated Reasoning Prompt

Task Overview:
You are a helpful SQL expert assistant. You should first think about how to write the SQL query by analyzing the question, database schema, and external knowledge, you can use the provided SQL execution tool to verify and refine your reasoning hypotheses. Finally, you provide the final SQL query in `<answer> </answer>`.

Database Information:

- **Database Engine:** {dialect}
- **Database Schema:** {db_schemas}
- **Matched Contents:** {matched_contents}

Requirements:

1. **Precision:** Make sure you only output the information that is asked in the question. If the question asks for a specific column, make sure to only include that column in the SELECT clause, nothing more.
2. **Completeness:** The generated query should return all of the information asked in the question without any missing or extra information.
3. **Correctness:** Before generating the final SQL query, please think through the steps of how to write the query. Validate your reasoning through tool testing.

Output Format [Important]:
Respond strictly in one of the following two modes. Do not mix the structures:

Option A (Tool call for validation or exploration):
<reasoning> Concise reasoning here. </reasoning>
`[Tool call code/statement here]`

Option B (Final answer):
<reasoning> Concise reasoning here. </reasoning>
<answer> YOUR FINAL SQL QUERY HERE </answer>

[Question] {question}
[Hint] {hint}

Figure 8: The proposed multi-turn reasoning prompt for FineStep.

Problem Definition

Question: When did the first-through-twelfth-grade school with the largest enrollment open?
Hint: K-12 means First-through-twelfth-grade.

Ground Truth SQL:

```
SELECT T1.OpenDate FROM schools AS T1 JOIN frpm AS T2 ON T1.CDSCode = T2.CDSCode
ORDER BY T2."Enrollment (K-12)" DESC LIMIT 1
```

Case 1: Positive Sample (Efficient Reasoning)

Step 1: Hypothesis & Tool Execution
<reasoning> To find the K-12 school with the largest enrollment, I need the `Enrollment (K-12)` column from the `frpm` table and the `OpenDate` from the `schools` table.
<tool_call>

```
SELECT s.OpenDate FROM schools s JOIN frpm f ON s.CDSCode = f.CDSCode
WHERE f."Enrollment (K-12)" = (SELECT MAX("Enrollment (K-12)") FROM frpm);
```

Observation: `| OpenDate: 2006-08-29 | (Row Count: 1)`

Step 2: Final Verification
<reasoning> The execution returned a valid date. This JOIN correctly maps question entities to schema.
<answer> `SELECT s.OpenDate FROM schools s JOIN frpm f ON s.CDSCode = f.CDSCode WHERE f."Enrollment (K-12)" = (SELECT MAX("Enrollment (K-12)") FROM frpm)`

Step-level Reward (Mean): $I_{\text{hard}} = 1.0$, $r_{\text{soft}} = 1.0$ **Outcome: SUCCESS**

Case 2: Negative Sample (Hallucination & Error Loop)

Step 1: Initial Hallucination
<reasoning> I will select `OpenDate` from `schools` and order by `Enrollment (K-12)`.
<tool_call> `SELECT OpenDate FROM schools ORDER BY "Enrollment (K-12)" DESC LIMIT 1;`
Observation: `Error: no such column: Enrollment (K-12)`

Step 2: Persistent Errors
<reasoning> Maybe the column is just named `Enrollment`... The backticks might be wrong...
<tool_call> `SELECT OpenDate FROM schools ORDER BY Enrollment DESC LIMIT 1;`
Observation: `Error: no such column: Enrollment`

Step 3: Suboptimal Fallback
<reasoning> I'll try joining `satscores` using `enroll12` as a fallback...
<tool_call> `SELECT s.OpenDate FROM schools s JOIN satscores sc ON s.CDSCode = sc.cds ORDER BY sc.enroll12 DESC LIMIT 1;`
Observation: `| OpenDate: [NULL] |`

Step 4: Final Suboptimal Answer
<reasoning> The query successfully executed by joining with `satscores`. I will provide this as the final answer.
<answer> `SELECT s.OpenDate FROM schools s JOIN satscores sc ON s.CDSCode = sc.cds ORDER BY sc.enroll12 DESC LIMIT 1`

Step-level Reward (Mean): $I_{\text{hard}} = 0.33$, $r_{\text{soft}} = 0.0$ **Outcome: FAILED**

Figure 9: Comparison of reasoning paths. The top block shows the task definition and Ground Truth; the middle and bottom blocks contrast a successful FineStep trajectory with a failed hallucination-prone trajectory.