DARES: DECISION-ALIGNED PROCESS SUPERVISION FOR MULTI-TURN TEXT-TO-SQL AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-turn Text-to-SQL agents must translate evolving user intents and database feedback into a sequence of tool-use and query decisions. Terminal execution rewards indicate whether a trajectory succeeds but not which decisions were responsible. Step-level rewards offer finer credit, yet their predefined step boundaries need not align with the semantic decisions that determine SQL correctness: one step may mix correct and faulty decisions, and one decision may span several steps. We propose DARES, which automatically discovers decision-aligned reward units from policy rollouts. Its Automatic Reward Unit Discovery (ARUD) constructs localized SQL contrasts, each pairing a correct and a faulty realization of one decision, verifies through controlled execution that the correct side indeed yields better outcomes, and maps the verified decisions back to the policy-generated token spans that express them. A decision-aligned process reward model learns from these contrasts and provides localized credit during GRPO, while execution rewards retain trajectory-level supervision. On interactive Text-to-SQL, DARES improves normalized reward over execution-only GRPO by up to 14.00 points. The results show that decision-aligned process supervision improves policy optimization across diverse multi-turn database tasks.

1 INTRODUCTION

Text-to-SQL aims to translate natural language into executable SQL queries, democratizing access to relational databases for non-expert users. In real-world data workflows, it supports natural-language analytics (Luo et al., 2020; 2022), business-intelligence question answering (Maddela et al., 2025; Sheinin et al., 2018), enterprise database operations (Chen et al., 2025; Xu et al., 2025; Zeng et al., 2020), and interactive SQL troubleshooting (Huo et al., 2026; Li et al., 2025b). In such settings, database tasks are rarely completed through a single, fully specified request. Therefore, effective database assistants must reason over an evolving interaction context rather than map a fixed question and schema to SQL in a single turn. They need to acquire missing information, inspect relevant database evidence, and adapt subsequent actions to both user feedback and changes in the database environment (Huo et al., 2026).

Such multi-turn interaction naturally motivates agentic reinforcement learning (RL) (Zhou et al., 2024), which optimizes policies through repeated action–observation cycles and task-level feedback. Recent work has demonstrated its effectiveness across interactive settings such as information retrieval (Wei et al., 2025; Qi et al., 2025; Yao et al., 2026), software engineering (Jin et al., 2026; Chen et al., 2026), and medical consultation (Bani-Harouni et al., 2026; Lai et al., 2026), where agents must actively gather evidence and adapt their behavior over multiple steps. Database environments provide a similarly favorable setting for agentic RL: SQL execution yields concrete and verifiable feedback about task outcomes. Yet this advantage also exposes a key limitation. Execution feedback is typically available only at the trajectory level, indicating whether the final interaction succeeds without identifying which intermediate decision was responsible. Consequently, a trajectory containing several correct decisions and one critical SQL error may receive the same terminal signal across all of its actions, obscuring the credit needed for effective policy learning.

Local credit assignment seeks to attribute trajectory-level outcomes to the intermediate decisions that produced them. Existing methods typically assign such credit at predefined action or step boundaries (Feng et al., 2025; Liu et al., 2026a). Yet these boundaries need not coincide with

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

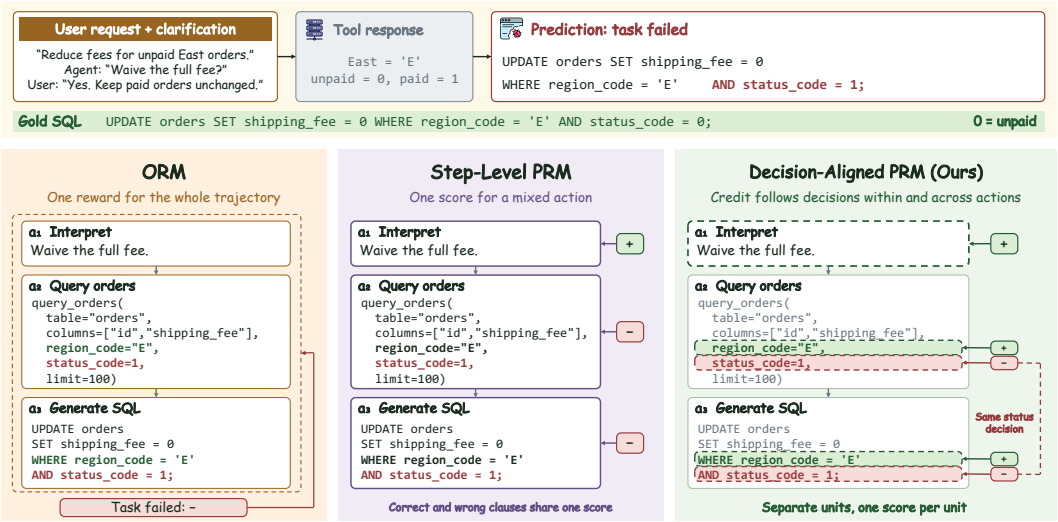


Figure 1: Credit assignment granularity of an outcome reward model (ORM), a step-level process reward model (PRM), and our decision-aligned PRM.

the semantic decisions that determine SQL correctness, as a single action may mix correct and faulty decisions while one decision may span multiple interaction steps (Figure 1). This motivates reward-unit discovery: identifying decision-aligned units from the trajectory itself and assigning credit directly to the policy spans that express them, where a policy span is a contiguous segment of policy-generated tokens.

Our key insight is that reward units need not be fixed in advance, but can instead be discovered from the semantic decisions expressed during interaction. In database agents, such decisions may be realized as a SQL clause or statement, a tool-call argument, or an intermediate reasoning span, depending on how the decision is expressed in a particular trajectory. Their consequences can often be grounded through database execution and environment feedback, enabling alternative realizations to be compared and validated. We use these validated semantic contrasts to identify decision-aligned policy spans and train a process reward model (PRM) to distinguish preferred from faulty realizations.

Based on this insight, we propose DARES, an agentic RL framework for multi-turn Text-to-SQL with automatically discovered process supervision. Its ♠ **Automatic Reward Unit Discovery** identifies semantic decisions from policy rollouts, validates their local causal effects through controlled execution, and aligns them with the policy spans that express them. The validated contrasts are used to train a ♣ **Decision-Aligned Process Reward Model** that scores each discovered unit, rather than a predefined step, under the interaction history in which it was produced. During GRPO optimization (Shao et al., 2024), execution rewards provide global trajectory credit, while the PRM assigns localized process credit to the discovered reward units.

Our contributions are threefold:

❶ We introduce **Automatic Reward Unit Discovery (ARUD)** for multi-turn Text-to-SQL, which discovers semantically meaningful reward units from interaction trajectories and grounds their credit through execution-validated local contrasts.

❷ We develop a **reward-unit-aligned learning framework** that trains a decision-aligned PRM over the discovered units and delivers its process feedback directly to the corresponding policy spans during reinforcement learning, enabling localized credit assignment beyond predefined action or step boundaries.

❸ On SQL generation, DARES improves over backbone-matched SFT and execution-only GRPO by up to 5.73 and 6.12 EX points, respectively. On interactive Text-to-SQL, it exceeds execution-only GRPO by up to 14.00 normalized-reward points.

2 PRELIMINARIES

Text-to-SQL. Text-to-SQL converts a natural-language question $\mathcal{Q}$ into an executable SQL query y over a database $D = \langle \mathcal{C}, \mathcal{T} \rangle$ (Li et al., 2023):

$$y = g_\theta(\mathcal{Q}, D), \tag{1}$$

where g_θ is the query-generation model, and $\mathcal{C}$ and $\mathcal{T}$ are the sets of columns and tables, respectively.

Agentic Reinforcement Learning. Agentic reinforcement learning (Agentic RL) models an agent as a policy that learns through multi-turn interaction with an external environment (Zhou et al., 2024; Zhang et al., 2026). Given a task x, at each step t, the agent selects an action $a_t \sim \pi_\theta(\cdot \mid h_t)$ based on its interaction history $h_t = (x, o_1, a_1, \ldots, o_t)$, where o_1 is the initial observation the environment exposes for x and each later o_t carries the environment response to a_{t-1}. This produces a trajectory

$$\tau = (o_1, a_1, \ldots, o_T, a_T), \tag{2}$$

in which an action may be an exploratory tool call against the environment or the final response that terminates the episode. Writing $R(\tau)$ for the scalar outcome assigned to a completed trajectory, the policy is optimized to maximize the expected return

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]\,. \tag{3}$$

Local Credit Assignment. Because $R(\tau)$ evaluates the trajectory as a whole, it does not indicate how individual agent behaviors contribute to the outcome. Local credit assignment combines feedback on intermediate decisions with this trajectory-level signal. Implicit step rewards (Liu et al., 2026a) instantiate this by assigning a separate credit to every action,

$$\widehat{A}_t = A^E(\tau) + \alpha\, c_t, \tag{4}$$

where $A^E(\tau)$ is the trajectory-level advantage, c_t is the step-level advantage derived for a_t, and α balances the two. The unit receiving credit is thus a single action, fixed in advance by the interaction protocol. We instead consider a setting in which the credit-bearing unit is not fixed in advance. A trajectory yields a selectively discovered set of variable-length semantic units, each aligned with a supported span of policy-generated tokens; these units need not cover the full trajectory. User utterances and tool observations serve only as context, while unsupported or unaligned policy tokens receive no localized process credit and are optimized only through the trajectory-level signal.

3 METHODOLOGY

In this section, we propose `DARES`, a decision-aligned agentic reinforcement learning framework for multi-turn Text-to-SQL, as illustrated in Figure 2. The policy model is first initialized through SFT to interact with database environments with tool-use capability. With policy rollouts, `DARES` automatically discovers causally validated reward units, learns a PRM from their counterfactual preferences, and aligns the resulting process rewards with their corresponding policy spans for subsequent GRPO optimization.

3.1 MULTI-TURN TOOL-USE POLICY INITIALIZATION

We instantiate the interaction as a partially observable Markov decision process (POMDP; Section A.1). The policy observes the database schema Σ_D and role-tagged events $e_j = (r_j, c_j)$, where $r_j \in \{\texttt{user}, \texttt{assistant}, \texttt{tool}\}$ and c_j is the event content. The trajectory and visible prefix are

$$\tau = (\Sigma_D, e_1, \ldots, e_L), \qquad h_t = (\Sigma_D, e_{<t}). \tag{5}$$

The policy samples $a_t \sim \pi_\theta(\cdot \mid h_t)$ at assistant events, generating textual responses, structured tool calls, or terminal SQL. User utterances and tool observations update the context but are not policy actions.

We initialize the policy on multi-turn tool-use demonstrations so that it learns both how to interact with the environment and how to produce the final query. Given $\mathcal{D}_{\mathrm{SFT}} = \{\tau_i^*\}_{i=1}^{N_{\mathrm{SFT}}}$, we minimize the masked autoregressive objective

$$\mathcal{L}_{\mathrm{SFT}} = -\frac{1}{N_{\mathrm{SFT}}} \sum_{i=1}^{N_{\mathrm{SFT}}} \sum_{n=1}^{|\tau_i^*|} m_{i,n} \log \pi_\theta\big(x_{i,n}^* \mid x_{i,<n}^*\big)\,, \tag{6}$$

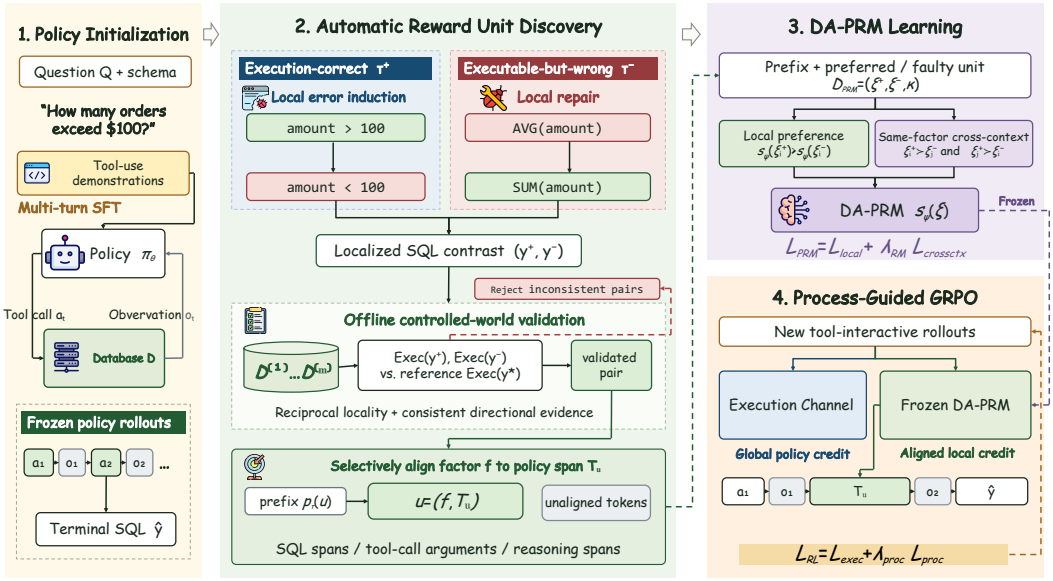


Figure 2: Overview of DARES for multi-turn Text-to-SQL.

where $x^*_{i,n}$ is the n-th token in the serialized trajectory and $m_{i,n} = 1$ only for policy-generated tokens. The objective therefore supervises textual responses, tool calls, and terminal SQL while using the schema, user utterances, and tool observations only as conditioning context.

3.2 Automatic Reward Unit Discovery

Automatic Reward Unit Discovery (ARUD) constructs causally grounded reward units from policy rollouts, using terminal SQL as an execution-grounded representation of localized decisions.

Reward Unit Formulation. Given a rollout trajectory τ, let $v(\tau)$ denote its policy-generated token sequence, and call any contiguous segment of $v(\tau)$ a *policy span*. Let $\mathcal{F}(\tau)$ be the set of supported semantic decisions recoverable from its structured actions and terminal SQL, and for each $f \in \mathcal{F}(\tau)$, let $\mathcal{A}_\tau(f)$ denote the set of policy spans that can be semantically aligned to f.

Definition 1 (Reward Unit)**.** A *reward unit* of τ is a pair

$$u = (f, T_u), \qquad f \in \mathcal{F}(\tau), \quad T_u \in \mathcal{A}_\tau(f), \tag{7}$$

namely a semantic decision together with a policy span aligned to it. Its *prefix* $p_\tau(u)$ is the online-visible history preceding T_u, including user utterances, earlier policy actions, and tool observations, and excludes all later events and the terminal execution outcome.

We write $\mathcal{U}(\tau) = \{(f, T) : f \in \mathcal{F}(\tau), T \in \mathcal{A}_\tau(f)\}$ for the set of reward units of τ. These units are selectively aligned rather than a partition of $v(\tau)$: their spans need not cover every policy token, unsupported or unaligned policy tokens receive no localized process credit, and non-policy events serve only as conditioning context. Since $\mathcal{A}_\tau(f)$ may contain several spans, one decision can be realized by multiple units in different actions, and one action can contain units of several decisions. In Figure 1, for example, the status filter forms two units, one in the tool-call argument and one in the terminal SQL.

Candidate Reward Unit Construction. We group frozen policy rollouts by interaction instance x, which specifies the database environment and user-side task, including benchmark-provided follow-up utterances. Terminal SQL execution partitions each group into execution-correct trajectories Γ_x^+ for local error induction and executable-but-wrong trajectories Γ_x^- for local repair.

For a trajectory $\tau^+ \in \Gamma_x^+$, its terminal SQL $\hat{y}(\tau^+)$ provides a trusted realization of the intermediate decisions made by the policy. ARUD selects an editable local decision and constructs a corrupted

counterpart by modifying only the corresponding SQL structure:

$$y = \hat{y}(\tau^+), \qquad \tilde{y} = \mathrm{Corrupt}(y). \tag{8}$$

The original realization is preferred and the corrupted one dispreferred.

For an executable-but-wrong trajectory $\tau^- \in \Gamma_x^-$, the generated SQL $y = \hat{y}(\tau^-)$ contains a naturally occurring policy error, and ARUD constructs a locally repaired counterpart while preserving the remaining query semantics:

$$\tilde{y} \in \mathrm{Repair}\big(y; \Gamma_x^+, y^\star\big), \tag{9}$$

where Γ_x^+ provides compatible correct sibling rollouts for the same question and $y^\star$ is used only as an offline reference for proposing local repairs.

In both cases, ARUD aligns the localized SQL contrast $(y, \tilde{y})$ to terminal-SQL spans, structured tool-call arguments, or intermediate reasoning spans expressing the same decision, instantiating Definition 1.

Causal Attribution and Validation. Semantic distinguishability alone does not establish that a reward unit contributes positively or negatively to the final task outcome. We therefore define the causal effect of a candidate unit u under its online-visible prefix $p_\tau(u)$. Let $s(u) \in \{\texttt{reasoning}, \texttt{tool}, \texttt{SQL}\}$ denote the surface containing T_u. The effect is

$$\Delta(u) = \mathbb{E}\big[R(\tau \mid do_{s(u)}(u = u^+), p_\tau(u))\big] - \mathbb{E}\big[R(\tau \mid do_{s(u)}(u = u^-), p_\tau(u))\big], \tag{10}$$

where a reasoning intervention regenerates subsequent policy outputs, a tool intervention re-executes the modified call and uses its new observation, and a terminal-SQL intervention has no suffix. In our offline construction, we approximate the direction of this effect through aligned terminal-SQL realizations y_u^+ and y_u^- in controlled database worlds. For each valid world $D^{(j)}$, we define

$$\Delta_j = \mathbf{1}\Big[\mathrm{Exec}(y_u^+, D^{(j)}) = \mathrm{Exec}(y^\star, D^{(j)})\Big] - \mathbf{1}\Big[\mathrm{Exec}(y_u^-, D^{(j)}) = \mathrm{Exec}(y^\star, D^{(j)})\Big]. \tag{11}$$

For terminal-SQL units, $y_u^\pm = u^\pm$; for reasoning and tool-call units, $y_u^\pm$ are SQL-level projections of the same semantic contrast, making Δ_j a proxy for the trajectory effect. A candidate is retained only when the intervention satisfies the reciprocal semantic-locality constraint and the preferred realization consistently receives positive directional evidence across valid worlds.

3.3 Reward-Unit-Aligned Credit Assignment

We train a Decision-Aligned Process Reward Model (DA-PRM) on ARUD preferences, which scores each reward unit under its interaction prefix, and use its scores to assign credit to the corresponding policy spans.

Process Reward Modeling. The preference dataset is

$$\mathcal{D}_{\mathrm{PRM}} = \big\{(\xi_i^+, \xi_i^-, \kappa_i)\big\}_{i=1}^{N_{\mathrm{PRM}}}, \qquad \xi_i^\pm = \mathrm{Format}\left(p_{\tau_i}(u_i), u_i^\pm\right), \tag{12}$$

where $\kappa_i = f(u_i)$ denotes the semantic factor of unit u_i, and u_i^+ and u_i^- are preferred and faulty realizations of the same decision under the shared online-visible prefix defined in Section 3.2.

We train a scalar PRM $s_\psi(\xi) \in \mathbb{R}$ to rank preferred realizations above faulty ones. For within-factor cross-context comparability, we additionally match pair i, whenever possible, with an independent pair j from a different source trajectory with $\kappa_i = \kappa_j$, enforcing $\xi_i^+ \succ \xi_j^-$ and $\xi_j^+ \succ \xi_i^-$.

The local objective is the Bradley–Terry loss $\mathcal{L}_{\mathrm{local}} = \mathrm{softplus}[-(s_\psi(\xi^+) - s_\psi(\xi^-))]$, and the PRM is trained on the combined objective

$$\mathcal{L}_{\mathrm{PRM}} = \mathcal{L}_{\mathrm{local}} + \lambda_{\mathrm{RM}} \mathcal{L}_{\mathrm{crossctx}}, \tag{13}$$

$$\mathcal{L}_{\mathrm{crossctx}}^{(i,j)} = \frac{1}{2}\Big[\mathrm{softplus}\big[-\big(s_\psi(\xi_i^+) - s_\psi(\xi_j^-)\big)\big] + \mathrm{softplus}\big[-\big(s_\psi(\xi_j^+) - s_\psi(\xi_i^-)\big)\big]\Big]. \tag{14}$$

Process-Guided GRPO. GRPO (Shao et al., 2024) optimizes the execution objective $\mathcal{L}_{\mathrm{exec}}$ using group-relative advantages from terminal SQL rewards: r_g^{exec} is $+1$ for an execution match, 0 for executable-but-incorrect SQL, and -1 for invalid or non-executable SQL.

The process-credit channel applies only to executable-but-incorrect trajectories. For each new rollout, ARUD's deterministic semantic extractor and span aligner identify supported units, excluding ambiguous or unaligned decisions. For each accepted unit, the frozen PRM takes $\xi_{g,u} = \mathrm{Format}(p_{\tau_g}(u), u)$ as input and produces

$$c_{g,u} = \tanh\left(\frac{s_\psi(\xi_{g,u}) - \mu_{f(u)}}{\max(\sigma_{f(u)}, \epsilon_{\mathrm{norm}})}\right), \tag{15}$$

where $\epsilon_{\mathrm{norm}} = 10^{-6}$ is a numerical-stability constant. We assign $c_{g,u}$ directly to the policy span T_u, rather than adding it to the trajectory reward. User utterances and tool observations provide context but receive no process credit.

To prevent trajectories with more factors, reward units, or longer spans from receiving disproportionately larger updates, we distribute a fixed process-credit budget as

$$w_{g,u,t} = \frac{1}{|\mathcal{F}_g|\, n_{g,f(u)}\, |T_u|}, \qquad t \in T_u, \tag{16}$$

where $\mathcal{F}_g$ is the set of factors with accepted units in rollout g, $n_{g,f(u)}$ is the number of units realizing factor $f(u)$, and $|T_u|$ is the span length, such that the total process-credit mass of each eligible trajectory is one. Using the token-level policy ratio $\rho_{g,t}$, the localized process objective is

$$\mathcal{L}_{\mathrm{proc}} = -\mathbb{E}\left[\sum_u \sum_{t \in T_u} w_{g,u,t} \min(\rho_{g,t} c_{g,u}, \mathrm{clip}(\rho_{g,t}, 1-\epsilon_{\mathrm{clip}}, 1+\epsilon_{\mathrm{clip}}) c_{g,u})\right]. \tag{17}$$

The policy is finally optimized with

$$\mathcal{L}_{\mathrm{RL}} = \mathcal{L}_{\mathrm{exec}} + \lambda_{\mathrm{proc}} \mathcal{L}_{\mathrm{proc}}, \tag{18}$$

combining global execution credit with localized process credit.

4 Experiments

This section is organized to answer the following research questions:

RQ1: Can causally grounded local supervision complement terminal execution feedback to improve multi-turn Text-to-SQL learning?

RQ2: How do different process reward modeling choices affect reward ranking and downstream policy performance?

RQ3: How effective is the learned PRM at localizing and repairing errors at inference time?

4.1 Setup

Datasets and Metrics. We evaluate three task families: SQL debugging on BIRD-CRITIC-SQLite (Li et al., 2025b) using success rate (SR); SQL generation on BIRD (Li et al., 2023) using execution accuracy (EX) and on SParC (Yu et al., 2019b) and CoSQL (Yu et al., 2019a) using question-level execution accuracy (QEX); and interactive Text-to-SQL on BIRD-INTERACT-Lite (Huo et al., 2026) using its official normalized reward, success-rate, and efficiency metrics. Section F.4 provides the dataset and evaluation details.

Baselines. For end-to-end evaluation, we compare against representative open- and closed-source Text-to-SQL LLMs, together with backbone-matched Qwen3.5-9B policies, including the supervised policy and GRPO variants under execution-only rewards and process rewards. Full baseline lists, hyperparameters, inference details, and hardware setup are provided in Section F.

4.2 Main Results

To answer RQ1, Table 1 compares `GRPO+DA-PRM` with external Text-to-SQL models across SQL generation, debugging, and interactive settings, while Table 2 fixes the Qwen3.5-9B backbone and compares the base model, SFT, execution-only GRPO, fixed-unit process supervision, and ours. This controlled ladder separates two properties that dense supervision conflates: how often credit arrives and whether the scored span isolates a single decision.

Table 1: Comparison with external models on SQL generation, debugging, and interactive Text-to-SQL. EX, QEX, SR, and NR are percentages; Turns and Coin are per-task averages left unranked, since early termination lowers both. Bold and underline mark the best and second-best arrow-marked results.

Model	BIRD	SParC	CoSQL	BIRD-CRITIC	BIRD-INTERACT c-Interact			BIRD-INTERACT a-Interact		
	EX ↑	QEX ↑	QEX ↑	SR ↑	NR ↑	SR ↑	Turns	NR ↑	SR ↑	Coin
Closed-source LLMs										
`GPT-5.6-Terra` (OpenAI, 2026)	40.09	44.47	50.61	43.37	<u>36.33</u>	<u>25.33</u>	3.75	24.67	12.33	18.23
`Grok 4.6` (xAI, 2026)	49.80	56.03	55.79	**49.10**	17.28	12.33	5.40	<u>28.10</u>	<u>16.67</u>	18.64
`Claude Sonnet 5` (Anthropic, 2026)	**58.74**	57.44	62.56	47.79	34.33	21.33	4.85	23.00	16.00	18.33
`Gemini 3.1 Pro Preview` (Google, 2026a)	39.57	41.56	46.47	<u>49.00</u>	**53.67**	**40.67**	5.18	**41.33**	**23.33**	16.79
`GLM-5.2` (Z.ai, 2026)	53.06	61.26	62.28	42.92	29.00	17.00	2.70	17.00	11.00	10.78
`DeepSeek-V4-Pro-0813` (DeepSeek-AI, 2026)	48.04	55.03	54.66	44.60	33.00	21.00	3.55	21.00	11.00	17.08
Open-source LLMs (≥27B)										
`Qwen3.8-27B` (Qwen Team, 2026)	46.81	27.76	41.11	39.36	33.00	22.33	3.53	25.00	<u>16.67</u>	14.13
`Gemma-4-31B-it` (Google, 2026b)	<u>56.13</u>	<u>64.26</u>	62.56	43.09	31.33	21.67	3.38	20.00	12.33	19.80
Open-source LLMs (≤14B)										
`MTSQL-R1-4B` (Guo et al., 2026)	32.79	45.22	42.24	28.31	2.33	0.33	0.90	3.00	1.00	6.33
`SQL-R1-7B` (MA et al., 2025)	44.32	59.02	59.64	30.20	2.00	0.33	4.76	4.00	0.33	18.77
`OmniSQL-7B` (Li et al., 2025a)	45.02	58.94	58.04	29.60	1.33	0.33	4.27	2.67	0.67	18.43
`SQL-R1-14B` (MA et al., 2025)	47.72	61.93	<u>62.75</u>	31.66	6.00	3.33	2.25	2.33	0.00	4.00
`XiyanSQL-14B` (Liu et al., 2026b)	45.05	56.40	56.53	21.79	4.00	1.67	1.26	1.67	0.00	3.65
Our Method										
`GRPO+DA-PRM (DARES)`	54.88	**66.75**	**63.19**	37.00	16.67	10.00	3.79	26.33	12.67	17.57

Table 2: Backbone-controlled comparison on Qwen3.5-9B, where all policies share one base model. The bottom row gives the largest point gain over any other listed baseline for each metric. The unranked Turns and Coin are omitted.

Training strategy	SQL Generation			BIRD-CRITIC	c-Interact		a-Interact	
	BIRD EX ↑	SParC QEX ↑	CoSQL QEX ↑	SR ↑	NR ↑	SR ↑	NR ↑	SR ↑
`Base model`	38.33	53.12	52.87	33.06	12.67	6.67	19.67	7.33
`SFT`	49.15	65.00	61.24	34.80	14.00	7.67	21.33	8.67
`GRPO+Exec-RM`	48.76	65.92	62.18	35.00	13.00	6.67	12.33	5.67
`GRPO+Step PRM`	52.62	65.84	61.52	36.40	15.00	8.33	24.00	11.00
`GRPO+DA-PRM (DARES)`	**54.88**	**66.75**	**63.19**	**37.00**	**16.67**	**10.00**	**26.33**	**12.67**
Max. gain	**+16.55**	**+13.63**	**+10.32**	**+3.94**	**+4.00**	**+3.33**	**+14.00**	**+7.00**

Observation ❶: Dense credit is most effective when aligned with individual decisions. On SQL generation, `GRPO+DA-PRM` outperforms every backbone-matched baseline, including execution-only GRPO, across all three benchmarks, with the largest gain on BIRD (up to $+16.55$ EX points). Fixed-unit supervision instead falls below execution-only rewards on SParC and CoSQL despite the same feedback frequency, suggesting that density alone is insufficient. Predefined boundaries can mix several decisions, whereas ARUD isolates decisions through context-matched substitutions that flip execution. SQL debugging shows a smaller gain over Step PRM ($+0.60$ SR points), consistent with the faulty query and issue description already providing localization cues. `DARES` leads the evaluated open models of at most 14B parameters on BIRD-CRITIC, but trails larger systems, as localized credit does not supply the schema- and dialect-specific knowledge needed for repair.

On interactive Text-to-SQL, execution-only GRPO barely improves c-Interact NR and degrades a-Interact NR from 19.67 to 12.33, highlighting the difficulty of crediting information-gathering actions through terminal feedback. Unlike the mixed pattern on SQL generation, Step PRM already outperforms Exec-RM on both interactive settings. `DARES` yields further gains by conditioning credit on the preceding interaction prefix, with larger NR gains on a-Interact than c-Interact ($+14.00$ vs. $+3.67$ points over Exec-RM), reaching 26.33 a-Interact NR and outperforming all evaluated open-source and most closed-source models. These results support assigning credit to localized decisions in their interaction context.

4.3 Ablation and Mechanism Analysis

To answer RQ2, we examine how PRM design choices affect both score comparability and downstream policy performance. Table 2 first establishes that ARUD-derived local credit improves policy performance. We then test whether the PRM makes these scores comparable across contexts, which matters because $c_{g,u}$ standardizes each score against per-factor statistics pooled across trajectories. Holding the ARUD units fixed, we ablate the terms of $\mathcal{L}_{\mathrm{PRM}}$ that enforce this comparability and evaluate each variant at two levels (Table 3): in score space, on held-out within-context pairs (ξ_i^+ vs. ξ_i^-) and same-factor pairs drawn from different trajectories (ξ_i^+ vs. ξ_j^-); and at the policy level, by retraining end to end with that variant as the process-credit channel under an identical GRPO recipe.

Table 3: Ablation of the cross-context objective with ARUD units fixed, at score level (left) and policy level (right). BIRD reports EX, SParC and CoSQL report QEX, and CRITIC reports SR. Bold and underline mark the best and second-best results in each column.

Variant	Score level			Policy level			
	Local	Cross-Ctx	Recip.	BIRD	SParC	CoSQL	CRITIC
Local-only (λ_{RM}=0)	**70.47**	66.89	44.39	42.89	55.11	55.60	34.80
+ comparison-matched	**70.47**	67.11	44.61	43.09	<u>55.69</u>	54.94	<u>35.00</u>
w/o reciprocity	69.43	68.20	47.77	42.83	55.44	<u>56.35</u>	34.20
Mismatched-factor ($\kappa_i \neq \kappa_j$)	<u>69.95</u>	70.45	**51.71**	<u>43.16</u>	54.84	55.69	33.20
Context-free (no $p_\tau(u)$)	68.91	**71.53**	49.26	42.76	54.53	55.88	34.80
DA-PRM (full)	**70.47**	<u>70.50</u>	<u>51.26</u>	**54.88**	**66.75**	**63.19**	**37.00**

Observation ❷: No single score-level diagnostic is sufficient; the complete objective yields the strongest policy. Local discrimination remains similar across variants (68.91%–70.47%), whereas the cross-trajectory objectives produce larger differences in Cross-Ctx and Recip. Removing cross-context training reduces these metrics from 70.50% and 51.26% to 66.89% and 44.39%, and matching the number of comparisons recovers only 0.22 points on each metric. Yet maximizing either diagnostic alone does not translate into a stronger policy: the context-free variant attains the highest Cross-Ctx score (71.53%), and mismatched-factor pairing attains the highest Recip. score (51.71%), but both remain below the full model on every policy-level benchmark. Across the three SQL-generation benchmarks, every incomplete variant also falls below both SFT and Exec-RM. The full DA-PRM objective combines factor-matched cross-trajectory comparisons, the online-visible prefix, and reciprocal supervision; the policy trained with it ranks first across all four policy columns, exceeding the strongest ablation by 11.72 EX points on BIRD, 11.06 on SParC, 6.84 on CoSQL, and 2.00 SR points on CRITIC. Its tied-best Local score and second-best Cross-Ctx and Recip. scores show that no single score-level diagnostic predicts the downstream policy result in this ablation.

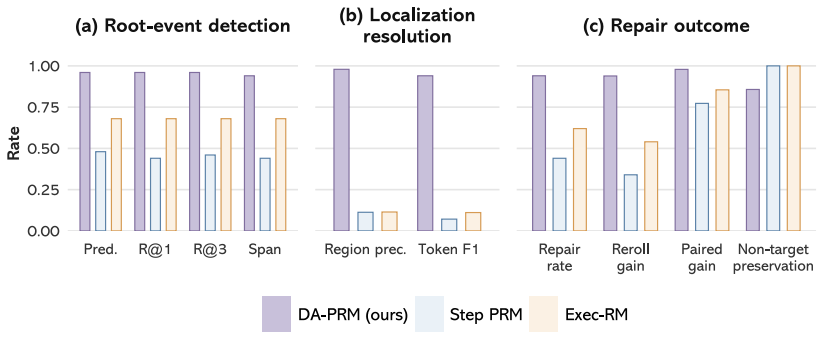


Figure 3: Controlled localization and repair. (a) Root-event detection (R@1/R@3) and gold-span coverage. (b) Localization metrics. (c) Repair outcomes.

4.4 PRM AS VERIFIER

To answer RQ3, we evaluate a verifier guided by the frozen DA-PRM on controlled error cases, comparing it with Step PRM and Exec-RM for fault detection, localization, and repair (Figure 3).

Observation ❸: Decision-aligned supervision supports precise localization and targeted repair on controlled errors. Figure 3(a) reports root-event hits, with the DA-PRM-based verifier reaching 0.96 at rank 1, compared with 0.44 for Step PRM and 0.68 for Exec-RM. Its R@3 is also 0.96, indicating that the recovered root events are already ranked first. Gold-span coverage reaches 0.94, showing that the predicted regions cover most of the annotated error spans.

Localization resolution further separates the methods (Figure 3(b)). The DA-PRM-based verifier reaches 0.94 end-to-end token F1, compared with 0.07 and 0.11 for Step PRM and Exec-RM, respectively. The high coverage and token F1 together show that its predicted spans capture the annotated error while remaining close to its boundaries. The baselines' much lower token F1 shows that finding a plausible event does not necessarily identify the text that needs correction.

For repair (Figure 3(c)), the DA-PRM-based verifier reaches a repair rate of 0.94, versus 0.44 for Step PRM and 0.62 for Exec-RM. Its paired reward gain conditional on intervention is 0.98, versus 0.77 and 0.86. These results connect precise localization with successful repair in the controlled setting.

5 RELATED WORK

Agentic RL. Reinforcement learning with verifiable rewards has strengthened language-model reasoning, with GRPO avoiding a learned critic through group-relative advantages (Shao et al., 2024). Agentic RL extends this setting from a single response to repeated action–observation cycles: Search-R1 learns when to retrieve (Jin et al., 2025), while WebRL and WebAgent-R1 optimize multi-turn web interaction from task outcomes (Qi et al., 2025; Wei et al., 2025). Sparse terminal rewards, however, do not identify which action contributed to success. Process reward models estimate intermediate promise (Choudhury, 2025), GiGPO derives step-level relative advantages from state-anchored groups (Feng et al., 2025), and Agent Lightning decomposes agent executions into trainable transitions (Luo et al., 2025). These approaches assign credit to different units—reasoning segments, tool calls, transitions, or turns—and no single granularity has become standard for agents whose tools return new evidence.

Text-to-SQL. Spider established cross-domain Text-to-SQL (Yu et al., 2018); subsequent systems improved schema grounding and constrained decoding (Wang et al., 2020; Scholak et al., 2021). LLM-based pipelines such as DIN-SQL and ReFoRCE instead combine decomposition, self-correction, consensus, and schema exploration at inference time (Pourreza & Rafiei, 2023; Deng et al., 2025). Recent benchmarks increase both scale and interactivity: BIRD emphasizes database values and external knowledge (Li et al., 2023), Spider 2.0 targets enterprise workflows (Lei et al., 2025), and BIRD-INTERACT includes user, knowledge, and database-state interaction (Huo et al., 2026). SDE-SQL further acquires evidence through iterative SQL probes (Xie et al., 2026). RL methods mostly optimize final-query correctness using execution-based or shaped rewards (MA et al., 2025; Pourreza et al., 2025; Yao et al., 2026; Zhu et al., 2026), whereas ReEx-SQL interleaves execution feedback with decoding (Dai et al., 2026). Thus, final-query RL and learned database interaction are converging, but the action interface, observation protocol, and temporal unit for credit assignment remain open design choices.

6 CONCLUSION

We introduced DARES, which discovers execution-validated semantic reward units for decision-aligned PRM training and local GRPO credit. Across SQL generation, debugging, and interactive Text-to-SQL, DARES outperforms backbone-matched baselines and all policy-level ablations. A verifier guided by the frozen DA-PRM localizes faulty decisions and aligned spans, enabling targeted repairs.

AI USE STATEMENT

Generative AI tools were used to aid and polish manuscript writing, support research ideation and execution, and generate synthetic data. The project code was implemented using generative AI tools. A frozen SFT policy generated the interaction trajectories from which ARUD constructs reward-unit preference pairs (Section 3.2 and Appendix E); the local SQL corruption and repair steps are primarily deterministic, and candidate pairs undergo execution-based validation. The authors manually reviewed the AI-assisted writing, research artifacts, synthetic data, and reported results. The authors take responsibility for the final content, including AI-assisted work.

ETHICS STATEMENT

This work uses publicly released database benchmarks and involves no new human annotations or collection of human-subject data. Dataset and model licenses are documented in Appendix H. SQL generation runs on isolated database copies, SQL debugging uses a fresh isolated database, and interactive evaluation retains the benchmark's isolation and read-only exploration rules (Appendix F.2). Granting a learned policy broader database privileges could expose sensitive records or alter data; deployment would require access controls, query validation, and sandboxing.

REPRODUCIBILITY STATEMENT

Section 3 specifies the training objectives and reward-unit construction; Appendices A, C, and D give the formal setting, algorithm, and span alignment. Appendix E records training configurations, while Appendices F.2, F.3, F.4, and F.5 specify evaluation protocols, prompts, datasets, and verifier metrics. Our code is available at `https://anonymous.4open.science/r/DARES-5D8B/`.

REFERENCES

Anthropic. Introducing Claude Sonnet 5, 2026. URL `https://www.anthropic.com/research/claude-sonnet-5`.

David Bani-Harouni, Chantal Pellegrini, Ege Özsoy, Nassir Navab, and Matthias Keicher. Language agents for hypothesis-driven clinical decision making with reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL `https://openreview.net/forum?id=7vHUQCMAzG`.

Yongchao Chen, Yueying Liu, Junwei Zhou, Yilun Hao, Jingquan Wang, Yang Zhang, Na Li, and Chuchu Fan. R1-code-interpreter: LLMs reason with code via supervised and multi-stage reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL `https://openreview.net/forum?id=FNlNH0iFOx`.

Yongnan Chen, Zhuo Chang, Shijia Gu, Yuanhang Zong, Mei Zhang, Shiyu Wang, Zixiang He, HongZhi Chen, Wei Jin, and Bin Cui. ADEPT-SQL: A high-performance text-to-SQL application for real-world enterprise-level databases. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pp. 275–283, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-253-4. doi: 10.18653/v1/2025.acl-demo.27. URL `https://aclanthology.org/2025.acl-demo.27/`.

Sanjiban Choudhury. Process reward models for llm agents: Practical framework and directions, 2025. URL `https://arxiv.org/abs/2502.10325`.

Yaxun Dai, Wenxuan Xie, Xialie Zhuang, Tianyu Yang, Ziyi Liu, Haiqin Yang, Yiying Yang, Yuhang Zhao, Pingfu Chao, and Wenhao Jiang. ReEx-SQL: Reasoning with execution-aware reinforcement learning for text-to-SQL. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 824–847, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.35. URL `https://aclanthology.org/2026.acl-long.35/`.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026. URL `https://huggingface.co/deepseek-ai/DeepSeek-V4-Pro-0813`.

Minghang Deng, Ashwin Ramachandran, Canwen Xu, Lanxiang Hu, Zhewei Yao, Anupam Datta, and Hao Zhang. RefoRCE: A text-to-SQL agent with self-refinement, format restriction, and column exploration. In *ICLR 2025 Workshop: VerifAI: AI Verification in the Wild*, 2025. URL https://openreview.net/forum?id=OuFIfDBwQd.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for LLM agent training. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=QXEhBMNrCW.

Google. Gemini 3.1 Pro: A smarter model for your most complex tasks, 2026a. URL https://blog.google/innovation-and-ai/models-and-research/gemini-models/gemini-3-1-pro/.

Google. Gemma 4: Byte for byte, the most capable open models, 2026b. URL https://blog.google/innovation-and-ai/technology/developers-tools/gemma-4/.

Taicheng Guo, Hai Wang, Chaochun Liu, Mohsen Golalikhani, Xin Chen, Xiangliang Zhang, and Chandan K. Reddy. MTSQL-r1: Towards long-horizon multi-turn text-to-SQL via agentic training. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 33905–33938, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1563. URL https://aclanthology.org/2026.acl-long.1563/.

Nan Huo, Xiaohan Xu, Jinyang Li, Per Jacobsson, Shipei Lin, Bowen Qin, Binyuan Hui, Xiaolong Li, Ge Qu, Shuzheng Si, Linheng Han, Edward Alexander, Xintong Zhu, Rui Qin, Ruihan Yu, Yiyao Jin, Feige Zhou, Weihao Zhong, Yun Chen, Hongyu Liu, Chenhao Ma, Fatma Ozcan, Yannis Papakonstantinou, and Reynold Cheng. BIRD-INTERACT: Re-imagining text-to-SQL evaluation via lens of dynamic interactions. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=nHrYBGujps.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan O Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training LLMs to reason and leverage search engines with reinforcement learning. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=Rwhi91ideu.

Yiyang Jin, Kunzhao Xu, Hang Li, Xueting Han, Yanmin Zhou, Cheng Li, and Jing Bai. Reveal: Self-evolving code agents via reliable self-verification. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=q56ZI1Co43.

Yunghwei Lai, Kaiming Liu, Ziyue Wang, Weizhi Ma, and Yang Liu. Doctor-r1: Mastering clinical inquiry with experiential agentic reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=vQGHTyL0Jw.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin SU, ZHAOQING SUO, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=XmProj9cPs.

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. Omnisql: Synthesizing high-quality text-to-sql data at scale. *Proc. VLDB Endow.*, 18(11):4695–4709, July 2025a. ISSN 2150-8097. doi: 10.14778/3749646.3749723. URL https://doi.org/10.14778/3749646.3749723.

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 Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*,

volume 36, pp. 42330–42357. Curran Associates, Inc., 2023. doi: 10.52202/075280-1835. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/83fc8fab1710363050bbd1d4b8cc0021-Paper-Datasets_and_Benchmarks.pdf.

Jinyang Li, Xiaolong Li, Ge Qu, Per Jacobsson, Bowen Qin, Binyuan Hui, Shuzheng Si, Nan Huo, Xiaohan Xu, Yue Zhang, Ziwei Tang, Yuanshuai Li, Florensia Widjaja, Xintong Zhu, Feige Zhou, Yongfeng Huang, Yannis Papakonstantinou, Fatma Ozcan, Chenhao Ma, and Reynold Cheng. SWE-SQL: Illuminating LLM pathways to solve user SQL issues in real-world applications. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025b. URL https://openreview.net/forum?id=yRxXTdElLv.

Xiaoqian Liu, Ke Wang, Yuchuan Wu, Fei Huang, Yongbin Li, Jianbin Jiao, and Junge Zhang. Agentic reinforcement learning with implicit step rewards. In *The Fourteenth International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=ooROvpmxMV.

Yifu Liu, Yin Zhu, Yingqi Gao, Zhiling Luo, Xiaoxia Li, Xiaorong Shi, Yuntao Hong, Jinyang Gao, Yu Li, Bolin Ding, and Jingren Zhou. Xiyan-sql: A novel multi-generator framework for text-to-sql. *IEEE Transactions on Knowledge and Data Engineering*, 38(4):2474–2487, 2026b. doi: 10.1109/TKDE.2026.3657851.

Xufang Luo, Yuge Zhang, Zhiyuan He, Zilong Wang, Siyun Zhao, Dongsheng Li, Luna K. Qiu, and Yuqing Yang. Agent lightning: Train any ai agents with reinforcement learning, 2025. URL https://arxiv.org/abs/2508.03680.

Yuyu Luo, Nan Tang, Guoliang Li, Wenbo Li, Tianyu Zhao, and Xiang Yu. Deepeye: A data science system for monitoring and exploring covid-19 data. *IEEE Data Eng. Bull.*, 43:121–132, 2020. URL https://api.semanticscholar.org/CorpusID:220651502.

Yuyu Luo, Xuedi Qin, Chengliang Chai, Nan Tang, Guoliang Li, and Wenbo Li. Steerable self-driving data visualization. *IEEE Transactions on Knowledge and Data Engineering*, 34(1):475–490, 2022. doi: 10.1109/TKDE.2020.2981464.

Peixian MA, Xialie Zhuang, Chengjin Xu, Xuhui Jiang, Ran Chen, and Jian Guo. SQL-r1: Training natural language to SQL reasoning model by reinforcement learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=hgJQcuDwm1.

Mounica Maddela, Lingjue Xie, Daniel Preotiuc-Pietro, and Mausam. STARQA: A question answering dataset for complex analytical reasoning over structured databases. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 34487–34499, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1749. URL https://aclanthology.org/2025.emnlp-main.1749/.

OpenAI. GPT-5.6: Frontier intelligence that scales with your ambition, 2026. URL https://openai.com/index/gpt-5-6/.

Mohammadreza Pourreza and Davood Rafiei. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. In *Advances in Neural Information Processing Systems*, volume 36, pp. 36339–36348. Curran Associates, Inc., 2023. doi: 10.52202/075280-1577. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/72223cc66f63ca1aa59edaec1b3670e6-Paper-Conference.pdf.

Mohammadreza Pourreza, Shayan Talaei, Ruoxi Sun, Xingchen Wan, Hailong Li, Azalia Mirhoseini, Amin Saberi, and Sercan O Arik. Reasoning-SQL: Reinforcement learning with SQL tailored partial rewards for reasoning-enhanced text-to-SQL. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=HbwkIDWQgN.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, Jie Tang, and Yuxiao Dong. WebRL: Training LLM web agents via

self-evolving online curriculum reinforcement learning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=oVKEAFjEqv.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, August 2026. URL https://qwen.ai/blog?id=qwen3.8.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 9895–9901, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.779. URL https://aclanthology.org/2021.emnlp-main.779/.

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

Vadim Sheinin, Elahe Khorashani, Hangu Yeo, Kun Xu, Ngoc Phuoc An Vo, and Octavian Popescu. QUEST: A natural language interface to relational databases. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA). URL https://aclanthology.org/L18-1469/.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 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*, pp. 7567–7578, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.677. URL https://aclanthology.org/2020.acl-main.677/.

Zhepei Wei, Wenlin Yao, Yao Liu, Weizhi Zhang, Qin Lu, Liang Qiu, Changlong Yu, Puyang Xu, Chao Zhang, Bing Yin, Hyokun Yun, and Lihong Li. WebAgent-r1: Training web agents via end-to-end multi-turn reinforcement learning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 7909–7928, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.401. URL https://aclanthology.org/2025.emnlp-main.401/.

xAI. Introducing Grok 4.6, 2026. URL https://x.ai/news/grok-4-6.

Wenxuan Xie, Yaxun Dai, and Wenhao Jiang. SDE-SQL: Enhancing text-to-SQL generation in large language models via self-driven exploration with SQL probes. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2507–2525, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.116. URL https://aclanthology.org/2026.acl-long.116/.

Keyan Xu, Dingzirui Wang, Xuanliang Zhang, Qingfu Zhu, and Wanxiang Che. Abacus-SQL: A text-to-SQL system empowering cross-domain and open-domain database retrieval. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pp. 118–128, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-253-4. doi: 10.18653/v1/2025.acl-demo.12. URL https://aclanthology.org/2025.acl-demo.12/.

Zhewei Yao, Guoheng Sun, Łukasz Borchmann, Zheyu Shen, Minghang Deng, Bohan Zhai, Hao Zhang, Ang Li, and Yuxiong He. Arctic-Text2SQL-r1: Simple rewards, strong reasoning in text-to-SQL. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 26966–26995, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1345. URL https://aclanthology.org/2026.findings-acl.1345/.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 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*, pp. 3911–3921, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1425. URL https://aclanthology.org/D18-1425/.

Tao Yu, Rui Zhang, Heyang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, Youxuan Jiang, Michihiro Yasunaga, Sungrok Shim, Tao Chen, Alexander Fabbri, Zifan Li, Luyao Chen, Yuwen Zhang, Shreya Dixit, Vincent Zhang, Caiming Xiong, Richard Socher, Walter Lasecki, and Dragomir Radev. CoSQL: A conversational text-to-SQL challenge towards cross-domain natural language interfaces to databases. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 1962–1979, Hong Kong, China, November 2019a. Association for Computational Linguistics. doi: 10.18653/v1/D19-1204. URL https://aclanthology.org/D19-1204/.

Tao Yu, Rui Zhang, Michihiro Yasunaga, Yi Chern Tan, Xi Victoria Lin, Suyi Li, Heyang Er, Irene Li, Bo Pang, Tao Chen, Emily Ji, Shreya Dixit, David Proctor, Sungrok Shim, Jonathan Kraft, Vincent Zhang, Caiming Xiong, Richard Socher, and Dragomir Radev. SParC: Cross-domain semantic parsing in context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4511–4523, Florence, Italy, July 2019b. Association for Computational Linguistics. doi: 10.18653/v1/P19-1443. URL https://aclanthology.org/P19-1443/.

Z.ai. GLM-5.2: Built for long-horizon tasks, 2026. URL https://z.ai/blog/glm-5.2.

Jichuan Zeng, Xi Victoria Lin, Steven C.H. Hoi, Richard Socher, Caiming Xiong, Michael Lyu, and Irwin King. Photon: A robust cross-domain text-to-SQL system. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 204–214, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-demos.24. URL https://aclanthology.org/2020.acl-demos.24/.

Guibin Zhang, Hejia Geng, Xiaohang Yu, Zhenfei Yin, Zaibin Zhang, Zelin Tan, Heng Zhou, Zhong-Zhi Li, Xiangyuan Xue, Yijiang Li, Yifan Zhou, Yang Chen, Chen Zhang, Yutao Fan, Zihu Wang, Songtao Huang, Francisco Piedrahita Velez, Yue Liao, Hongru WANG, Mengyue Yang, Heng Ji, Jun Wang, Shuicheng YAN, Philip Torr, and LEI BAI. The landscape of agentic reinforcement learning for LLMs: A survey. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=RY19y2RI1O. Survey Certification.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. ArCHer: Training language model agents via hierarchical multi-turn RL. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62178–62209. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/zhou24t.html.

Yuxuan Zhu, Tengjun Jin, Yoojin Choi, and Daniel Kang. Human-level text-to-SQL via reinforcement learning on verified data, without pipeline engineering. *arXiv preprint arXiv:2603.20004*, 2026. URL https://arxiv.org/abs/2603.20004.

A Formal Setting and Policy Initialization

A.1 POMDP Formulation

We formulate agentic Text-to-SQL as a partially observable Markov decision process

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{P}, \mathcal{R}, \Omega, \gamma), \tag{19}$$

where $\mathcal{S}$ is the latent environment state space, $\mathcal{A}$ is the agent action space, $\mathcal{O}$ is the observation space, $\mathcal{P}$ denotes the environment transition dynamics, Ω is the observation function, $\mathcal{R}$ is the reward function, and $\gamma \in (0, 1]$ is the discount factor. The latent state comprises the full database D together with the environment-side task specification, while the policy observes only the visible schema Σ_D and the role-tagged interaction events defined in Section 3. In our instantiation $\mathcal{R}$ is emitted only when an episode terminates, namely the execution reward of the terminal SQL, so it reduces to the trajectory-level $R(\tau)$ of Section 2 and we take $\gamma = 1$. Relative to the generic notation in Section 2, x comprises the initial user request and database environment, o_1 corresponds to Σ_D, and later observations are tool results or follow-up user utterances. We index the complete role-tagged event sequence by L; only assistant events are policy actions.

A.2 Trajectory Serialization and Supervision Masks

To instantiate the SFT objective in Equation (6), we serialize each demonstration in chronological order while preserving the schema and the role of every interaction event. The mask is set to $m_{i,n} = 1$ for tokens generated by the policy, including textual responses, structured tool calls, and the terminal SQL, and to $m_{i,n} = 0$ for the schema and all environment-provided tokens, including user utterances and tool observations. Masked tokens remain in $x^*_{i,<n}$ and therefore condition subsequent policy decisions without being prediction targets.

A.3 Notation and Terminology

The history h_t is the role-tagged serialization of the observation and action history in Section 2, including the schema and all events visible before action a_t. A *semantic decision* is a query choice, such as a predicate or projection, represented by a *factor* f. A *realization* is a concrete expression of that decision. A *reward unit* pairs one factor with one aligned policy span, so one factor may yield several units when the same decision appears in multiple policy actions. The sample counts N_{SFT} and N_{PRM} refer to the SFT trajectories and PRM preference pairs, respectively.

For PRM training and scoring, $\mathrm{Format}(p_\tau(u), u)$ serializes the role-tagged online-visible prefix and the selected unit realization into one model input. It preserves event order and role markers while excluding future events and the terminal execution outcome.

B Reliability of the Causal Preference Direction

The directional execution test in Section 3.2 evaluates a fixed SQL contrast, whereas a trajectory intervention regenerates downstream actions and observations. We relate these quantities through an explicit approximation assumption and derive a finite-sample condition for recovering the sign of the intervention effect.

B.1 Setting and Assumptions

Fix an interaction instance, an online-visible prefix $p = p_\tau(u)$, and a candidate pair (u^+, u^-) with aligned terminal queries (y^+, y^-). Let π_0 be the frozen continuation policy used to define the intervention. Write $S(\tau) \in \{0, 1\}$ for terminal execution success, assigning zero to unsuccessful or invalid outputs, and define

$$\begin{aligned} Q_p^{\pm} &= \mathbb{E}_{\pi_0}\left[S(\tau) \mid do(u = u^{\pm}), p\right], \\ \Delta_S(u) &= Q_p^{+} - Q_p^{-}. \end{aligned} \tag{20}$$

The expectation includes continuation randomness and the target environment's conditional state distribution given p. Thus Δ_S is the causal effect in Section 3.2 specialized to the success indicator. Its relation to the three-valued execution reward is given in Section B.2.

For a distribution $\mathcal{W}$ over valid controlled database worlds, define

$$Z^{\pm}(D) = \mathbf{1}\big[\mathrm{Exec}(y^{\pm}, D) = \mathrm{Exec}(y^{\star}, D)\big], \\ V^{\pm} = \mathbb{E}_{D\sim\mathcal{W}}[Z^{\pm}(D)], \qquad d_{\mathcal{W}} = V^{+} - V^{-}. \tag{21}$$

An execution failure has indicator zero. The reference query $y^{\star}$ represents the intended task semantics throughout the support of $\mathcal{W}$.

Assumption 1 (Well-defined local interventions)**.** Both realizations are valid replacements at the same prefix and express alternative values of one supported semantic decision. Their aligned SQL contrast changes that decision while preserving the remaining query semantics. The prefix, conditional initial environment distribution, continuation policy, environment transition rules, and user-response protocol are shared across the two interventions. After replacement, the suffix is generated anew under these shared mechanisms, including new tool observations. The resulting episodes terminate under a fixed evaluation budget.

This specifies an intervention distribution directly; identification from an observational comparison of successful and unsuccessful rollouts is unnecessary. It also permits downstream actions to correct or amplify the replaced decision.

Assumption 2 (Independent validation worlds)**.** The prefix, candidate pair, aligned queries, reference, world distribution, and sample size $m \geq 1$ are fixed before validation. Conditional on this construction, $D^{(1)}, \ldots, D^{(m)}$ are independent draws from $\mathcal{W}$. Validity constraints preserve the schema, required database constraints, and the task semantics expressed by $y^{\star}$, and are specified before sampling. Both queries are evaluated on each sampled world without filtering worlds by their observed directional evidence.

The two indicators within a world may be dependent. Independence is required across validation worlds, so one can evaluate both realizations on the same database.

Assumption 3 (Bounded execution-proxy error)**.** There exist nonnegative bounds ε_{+} and ε_{-} such that

$$|Q_p^{+} - V^{+}| \leq \varepsilon_{+}, \qquad |Q_p^{-} - V^{-}| \leq \varepsilon_{-}. \tag{22}$$

These bounds account for differences between fixed-query validation and regenerated trajectories, including continuation behavior and the discrepancy between controlled worlds and the target environment. Semantic locality alone does not imply that the bounds vanish. A sufficient condition for an arm's bound is a coupling of its intervention success and world-validation indicator whose disagreement probability is at most $\varepsilon_{\pm}$: under that coupling, $|\mathbb{E}S - \mathbb{E}Z^{\pm}| \leq \mathbb{E}|S - Z^{\pm}| = \Pr(S \neq Z^{\pm}) \leq \varepsilon_{\pm}$.

B.2 Finite-Sample Direction Guarantee

For each validation world, let $\Delta_j = Z^{+}(D^{(j)}) - Z^{-}(D^{(j)})$, as in Section 3.2, and define

$$\widehat{\Delta}_m = \frac{1}{m}\sum_{j=1}^{m} \Delta_j, \qquad b_m(\delta) = \sqrt{\frac{2\log(2/\delta)}{m}}, \quad \delta \in (0,1). \tag{23}$$

Theorem 1 (Reliability of the causal preference direction)**.** *Under Assumptions 1–3, with probability at least* $1 - \delta$ *over the validation worlds,*

$$\left|\Delta_S(u) - \widehat{\Delta}_m\right| \leq b_m(\delta) + \varepsilon_{+} + \varepsilon_{-}. \tag{24}$$

Consequently, on this same event,

$$\widehat{\Delta}_m > b_m(\delta) + \varepsilon_{+} + \varepsilon_{-} \quad \Longrightarrow \quad \Delta_S(u) > 0. \tag{25}$$

Proof. Conditional on the candidate construction, the variables Δ_j are independent, lie in $[-1, 1]$, and have common mean $d_{\mathcal{W}}$. Hoeffding's inequality gives, for every $a > 0$,

$$\Pr\Big(\Big|\widehat{\Delta}_m - d_{\mathcal{W}}\Big| \geq a\Big) \leq 2\exp\Big(-\frac{ma^2}{2}\Big). \tag{26}$$

Taking $a = b_m(\delta)$ bounds this probability by δ. Separately, Assumption 3 and the triangle inequality imply

$$\begin{aligned} |\Delta_S(u) - d_{\mathcal{W}}| &= |(Q_p^+ - V^+) - (Q_p^- - V^-)| \\ &\le |Q_p^+ - V^+| + |Q_p^- - V^-| \le \varepsilon_+ + \varepsilon_-. \end{aligned} \tag{27}$$

Combining the two inequalities proves Equation (24). Its lower endpoint gives $\Delta_S(u) \ge \widehat{\Delta}_m - b_m(\delta) - \varepsilon_+ - \varepsilon_-$, which is strictly positive under Equation (25). □

Screening multiple candidates. For $K \ge 1$ candidates fixed before validation, apply the theorem to each with failure probability δ/K. A union bound yields simultaneous coverage at least $1 - \delta$, replacing $b_m(\delta)$ by $\sqrt{2\log(2K/\delta)/m}$ when every candidate uses m worlds. Candidates may share validation worlds; independence across candidates is unnecessary. The simultaneous event remains valid when candidates are subsequently retained according to their validation scores. Adaptively constructed candidates require fresh validation worlds for a conditional per-candidate guarantee, with failure probabilities allocated across candidates for a joint guarantee.

Interpretation for ARUD. The bound separates finite-world sampling uncertainty from execution-proxy error. Increasing m reduces the former at rate $m^{-1/2}$ while leaving the latter unchanged. In particular, positive evidence on sampled worlds certifies a positive trajectory effect only when its margin exceeds both terms in Equation (25). The result is conditional on the stated proxy-error bounds; it supplies no numerical values for them. A deterministically designed or outcome-selected collection of worlds does not satisfy Assumption 2 automatically. Such a collection still provides execution evidence, but the sampling guarantee requires an independent validation stage with the specified world distribution.

Relation to the GRPO execution reward. Let $I(\tau)$ indicate invalid or non-executable terminal SQL. For the reward in Section 3, $R_{\text{exec}}(\tau) = S(\tau) - I(\tau)$. Writing $q_{\text{inv}}^{\pm} = \Pr(I(\tau) = 1 \mid do(u = u^{\pm}), p)$, its intervention effect is

$$\Delta_{\text{exec}}(u) = \Delta_S(u) - (q_{\text{inv}}^+ - q_{\text{inv}}^-). \tag{28}$$

If $|q_{\text{inv}}^+ - q_{\text{inv}}^-| \le \eta$, the same confidence event gives

$$\Delta_{\text{exec}}(u) \ge \widehat{\Delta}_m - b_m(\delta) - \varepsilon_+ - \varepsilon_- - \eta. \tag{29}$$

Thus the success-direction guarantee transfers directly when the two arms have equal invalid-output probabilities, including the case of always-executable continuations. Otherwise, the invalid-output difference contributes an additional term to the sufficient margin.

C End-to-End Training Algorithm and Complexity

Algorithm 1 summarizes the complete training procedure. The reward-unit dataset is constructed once from rollouts of the SFT-initialized policy. The resulting PRM parameters are then frozen during policy optimization, keeping the process-scoring function fixed while the policy is updated.

Time complexity. We separate neural computation from database execution because the latter depends on the database engine, query plan, and database size. Let

$$\Phi_\eta(\ell) = H_\eta\big(\ell^2 d_\eta + \ell d_\eta^2\big), \qquad \eta \in \{\theta, \psi\}, \tag{30}$$

denote the asymptotic cost of one dense-Transformer pass with H_η layers, hidden width d_η, and sequence length ℓ. A backward pass has the same order, and a cached autoregressive rollout also has aggregate cost $O(\Phi_\theta(\ell))$. For E_{SFT} epochs over demonstrations of lengths $\{\ell_i^{\text{SFT}}\}$, policy initialization costs

$$T_{\text{SFT}} = O\left(E_{\text{SFT}} \sum_i \Phi_\theta(\ell_i^{\text{SFT}})\right). \tag{31}$$

For ARUD, let N_Γ be the number of frozen rollouts, K the number of local contrast candidates, $m_{\max}$ the maximum number of controlled worlds per candidate, and C_{exec} the cost of one database

Algorithm 1 End-to-end training of DARES

Require: SFT demonstrations $\mathcal{D}_{\mathrm{SFT}}$; training instances $\mathcal{X}$ with references $y_x^\star$ and valid controlled worlds $\mathcal{W}_x$; rollout and optimization budgets
Ensure: Optimized policy π_θ and process reward model s_ψ

Train π_θ on $\mathcal{D}_{\mathrm{SFT}}$ with the masked objective in Equation (6).
Initialize $\mathcal{D}_{\mathrm{PRM}} \leftarrow \varnothing$.
for each interaction instance $x \in \mathcal{X}$ **do**
  Sample and freeze policy rollouts Γ_x; execute their terminal SQL.
  Partition Γ_x into correct Γ_x^+ and executable-but-wrong Γ_x^-.
  for each $\tau \in \Gamma_x^+ \cup \Gamma_x^-$ **do**
    if $\tau \in \Gamma_x^+$ **then**
      Form ordered local contrasts (y^+, y^-, f) by corrupting one editable decision in $\hat{y}(\tau)$.
    else
      Form (y^+, y^-, f) by locally repairing $\hat{y}(\tau)$ using compatible evidence from Γ_x^+ and $y_x^\star$.
    end if
    for each resulting contrast (y^+, y^-, f) **do**
      Align f to policy spans $T \in \mathcal{A}_\tau(f)$; discard ambiguous or unsupported alignments.
      if the contrast satisfies reciprocal semantic locality and has positive directional evidence over $\mathcal{W}_x$ **then**
        for each retained aligned span T **do**
          Set $u = (f, T)$ and recover its online-visible prefix $p_\tau(u)$.
          Add $(\mathrm{Format}(p_\tau(u), u^+), \mathrm{Format}(p_\tau(u), u^-), f)$ to $\mathcal{D}_{\mathrm{PRM}}$.
        end for
      end if
    end for
  end for
end for
Match each retained pair, when possible, to one different-trajectory pair with the same factor f.
Train s_ψ with $\mathcal{L}_{\mathrm{local}} + \lambda_{\mathrm{RM}}\mathcal{L}_{\mathrm{crossctx}}$; compute training-set factor statistics (μ_f, σ_f) and freeze s_ψ.
for each GRPO update **do**
  Sample a group of tool-interactive trajectories $\{\tau_g\}_{g=1}^G$ from π_θ and compute each terminal execution reward r_g^{exec}.
  for each trajectory τ_g **do**
    if τ_g is executable but incorrect **then**
      Extract supported units $\mathcal{U}_g$ and their aligned policy spans using the same extractor and aligner.
      for each $u \in \mathcal{U}_g$ **do**
        Score $\mathrm{Format}(p_{\tau_g}(u), u)$ with s_ψ; compute $c_{g,u}$ and $w_{g,u,t}$ as in Section 3.
      end for
    else
      Set $\mathcal{U}_g \leftarrow \varnothing$.
    end if
  end for
  Construct $\mathcal{L}_{\mathrm{exec}}$ from group-relative execution rewards and $\mathcal{L}_{\mathrm{proc}}$ from accepted unit spans.
  Update θ using $\mathcal{L}_{\mathrm{exec}} + \lambda_{\mathrm{proc}}\mathcal{L}_{\mathrm{proc}}$.
end for
return π_θ, s_ψ

execution. Let C_{pair} and C_{align} denote, respectively, the cost of constructing one local corruption or repair and aligning its semantic factor to policy tokens. With rollout length bounded by L and at most B tool or terminal-query executions per trajectory,

$$T_{\mathrm{ARUD}} = O\big(N_\Gamma[\Phi_\theta(L) + BC_{\mathrm{exec}}] + K[C_{\mathrm{pair}} + C_{\mathrm{align}} + m_{\max}C_{\mathrm{exec}}]\big). \tag{32}$$

The first execution term includes the tool interactions and terminal evaluation used to partition rollouts, while the $Km_{\max}$ term comes from controlled-world validation. Under a single-pass implementation, deterministic parsing, factor extraction, and span bookkeeping are linear in the inspected SQL and trajectory lengths; otherwise their cost remains explicit in C_{align}. If corruption or repair uses a learned generator, its inference cost is included in C_{pair}.

Let P be the number of retained local and reciprocal preference comparisons, L_{PRM} their maximum formatted length, and E_{PRM} the number of PRM training epochs. Indexing examples by factor makes cross-context matching $O(K + P)$ instead of a quadratic all-pairs search. Since each retained pair is

matched to at most one same-factor pair, $P = O(K)$, and PRM training costs

$$T_{\mathrm{PRM}} = O(K + E_{\mathrm{PRM}} P \Phi_\psi(L_{\mathrm{PRM}})) \,. \tag{33}$$

Finally, consider Q GRPO updates with group size G, at most U accepted reward units per eligible trajectory, and E_{pol} policy passes per update. Let C_{on} be the online extraction and alignment cost per trajectory. A conservative bound that scores each unit with its formatted prefix independently is

$$\begin{aligned} T_{\mathrm{RL}} = O\big(QG[(1 + E_{\mathrm{pol}})\Phi_\theta(L) + BC_{\mathrm{exec}} + C_{\mathrm{on}} \\ + U\Phi_\psi(L_{\mathrm{PRM}})]\big). \end{aligned} \tag{34}$$

Span weighting is linear in the credited token mass, which is at most QGL and is asymptotically dominated by the dense-Transformer policy passes above. The additional framework-specific costs are the one-time $Km_{\max}$ controlled executions used by ARUD and up to U frozen-PRM evaluations per eligible online trajectory. Batching or shared-prefix caching can reduce wall-clock time, but not the total-work bounds above. The complete training cost is $T_{\mathrm{SFT}} + T_{\mathrm{ARUD}} + T_{\mathrm{PRM}} + T_{\mathrm{RL}}$.

D Semantic Factor Extraction and Span Alignment

A candidate enters $\mathcal{U}(\tau)$ only if its query, semantic factor, AST anchor, character span, and policy span are determined unambiguously. Otherwise, it is recorded with an abstention reason and excluded from supervision and process credit.

Representation and factors. We use SQLGlot ($\geq 26, < 28$), SQLite dialect, and require exactly one nonempty relational query. Extraction is scope-local: CTEs, derived tables, expression subqueries, and set-operation branches receive deterministic child scopes; correlated references may bind outward without changing their factor's scope. Physical tables are resolved through each scope's selected sources. A factor stores its type, canonical value, resolved and unresolved references, AST paths, and metadata. Its structural identifier is

$$\mathrm{id}(f) = (\mathrm{scope}(f), \mathrm{type}(f), \mathrm{occ}(f)), \tag{35}$$

where occurrence indices follow clause order within each type and scope. Identity is invariant to aliases, case, whitespace, and redundant parentheses, but not insertion, deletion, or reordering of same-type siblings.

Canonicalization case-folds identifiers, removes aliases and redundant parentheses, uses resolved relation instances, sorts flattened homogeneous Boolean chains and symmetric comparison operands, and preserves literal types and values. Projection aliases map to projection slots and subqueries to scope identifiers. These signatures encode structure, not correctness. Qualified columns resolve from the current scope outward; unqualified columns use a projection alias, a sole relation, or a unique schema match. Unresolved factors remain auditable but cannot receive interventions or credit.

AST bindings and character spans. AST paths traverse sorted named arguments with list indices and must resolve uniquely. Coupled realizations may have multiple anchors. Local, complete, reversible bindings support columns, synchronized column sets, literals, integer literals, ordering, aggregation, and filter operators. Edits operate on the AST and are rendered and reparsed as a single supported query. For alignment, the anchored node is rendered without pretty printing and matched to the original SQL; source locations disambiguate repeated text, and join spans include `JOIN`. The exact preceding SQL prefix is retained as provenance. Terminal-SQL credit additionally requires a unique unit-string occurrence (exact, then case-insensitive) or a unique factor slot. Ambiguity causes abstention without falling back to a wider span.

Token alignment. All intervals are zero-based and half-open. SQL character offsets refer to terminal SQL, local offsets to the decoded final assistant span, and token offsets to the full response. Decode stored policy tokens without stripping special tokens or cleaning spaces, then re-encode without adding special tokens. Proceed only if token IDs match exactly and offsets are valid, monotone, and equal in count to the stored tokens. Locate SQL by its last exact occurrence, or its last case-insensitive occurrence when no exact match exists. With local SQL offset o_{sql}, factor span $[c_s, c_e)$ maps to $[a_s, a_e) = [o_{\mathrm{sql}} + c_s, o_{\mathrm{sql}} + c_e)$. Select tokenizer offsets $[l_j, r_j)$ satisfying

$$l_j < a_e, \qquad r_j > a_s, \qquad r_j > l_j. \tag{36}$$

Table 4: Sizes of the training and development sets used for SFT.

Dataset	Training size	Development size
BIRD	9,418	1,534
SParC	3,032	422
CoSQL	2,164	297
SIX-GYM-SQLITE	3,705	1,294
Total	18,319	3,547

The selected tokens must form a nonempty contiguous cover of both character boundaries and contain only policy-mask positions after conversion to global coordinates. Boundary subwords may include adjacent characters. Fuzzy matching, guessed offsets, decode–re-encode mismatches, and wider-span fallbacks are rejected.

Overlap and abstention. For valid candidates, first remove intervals that strictly contain another interval. Build a conflict graph on the remainder using

$$A \sim B \iff \max(A_s, B_s) < \min(A_e, B_e). \tag{37}$$

Abstain every candidate in a component with more than one interval, including identical and partially overlapping spans; adjacent spans remain separate. No score-based selection or merging is used. Accepted spans therefore satisfy

$$T_u \cap T_{u'} = \varnothing \quad (u \neq u'). \tag{38}$$

Disjoint realizations of one factor form separate units sharing its process-credit budget. Unresolved references, invalid anchors, ambiguous occurrences, or invalid policy-token covers retain an abstention reason; an empty $\mathcal{A}_\tau(f)$ receives no credit.

E Training Details

E.1 Supervised Fine-Tuning

The multi-turn SQL SFT stage for the Qwen3.5-9B policy uses the official training splits of BIRD, SParC, CoSQL, and SIX-GYM-SQLITE. After episode-preserving serialization and length filtering, the training split contains 9,418 BIRD, 3,032 SParC, 2,164 CoSQL, and 3,705 SIX-GYM-SQLITE records, for 18,319 benchmark-derived training records in total. The corresponding development split contains 1,534 BIRD, 422 SParC, 297 CoSQL, and 1,294 SIX-GYM-SQLITE records. This development split is not used for checkpoint or hyperparameter selection. SParC and CoSQL dialogues retain their multi-turn structure, and conversations are split only at SQL-turn boundaries rather than token-truncated. We fine-tune the text-only Qwen3.5-9B initialization for three epochs with AdamW, a learning rate of 2×10^{-5}, cosine decay, a warmup ratio of 0.05, weight decay 0.01, and gradient clipping at 1.0.

E.2 Process Reward Model

The DA-PRM starts from a fresh text-only Qwen3.5-9B backbone and adds a rank-8 LoRA adapter and a single scalar reward head for process scoring. The source-disjoint split contains 26,243 training pairs, 3,287 development pairs, and 3,285 test pairs; the training sampler draws from 13,882 unique pair identities, with 26,243 pair presentations in each nominal epoch. Training runs for five epochs, 206 optimizer steps per epoch, and 1,030 optimizer steps in total. We use AdamW with learning rate 2×10^{-5}, linear decay, warmup ratio 0.05, weight decay 0.01, and maximum gradient norm 1.0. The global pair batch size is 128 and the per-device micro batch contains one pair. The maximum model input is 4,096 tokens.

The PRM objective uses unit weight for the local term and $\lambda_{\mathrm{RM}} = 1$ for cross-context ranking. Each nominal epoch contains 26,243 local comparisons and 26,238 directed same-factor, cross-unit comparisons, which is approximately a 1:1 ratio. Cross-factor training comparisons are excluded.

Table 5: Training costs measured in GPU hours. Failed runs, evaluation, data generation, model merging, and duplicated steps from checkpoint resumption are excluded.

Training stage	Wall-clock time (h)	#GPUs	GPU hours
SFT	1.95	4	7.80
RL	27.47	4	109.86
DA-PRM	3.47	2	6.93
Total	32.88	–	124.59

Verifier output head. For verifier evaluation, we retain the trained DA-PRM model and replace only its scalar reward head with a head that produces structured diagnoses. Thus, the verifier and DA-PRM share all components except the output head; the scalar process score is not the reported verdict or confidence field.

E.3 Process-Guided GRPO

The multi-turn GRPO training split contains 22,477 prompts; this is a separate prompt pool from the 18,319 benchmark-derived SFT records above. Training uses one epoch and 2,809 policy updates, with four sibling rollouts per prompt ($G = 4$). The global prompt batch, PPO mini batch, and per-device PPO micro batch are 8, 8, and 1, respectively. The actor uses AdamW with learning rate 1×10^{-6}, a constant scheduler without warmup, weight decay 0.01, and maximum gradient norm 1.0. PPO uses clipping $\epsilon_{\text{clip}} = 0.2$, one policy-update epoch after each rollout batch, KL loss coefficient 0.001 with the low-variance KL estimator, and entropy coefficient zero. KL is applied as an actor loss rather than added directly to the reward. The process-reward coefficient is $\lambda_{\text{proc}} = 0.5$. Rollouts use temperature 0.8, top-$p = 0.95$, and maximum prompt and response lengths of 6,144 and 4,096 tokens.

E.4 Automatic Reward Unit Discovery

ARUD consumes a frozen lineage containing 44,954 rollouts from 22,477 source examples, or two rollouts per source example. The rollout policy is the SFT Qwen3.5-9B; sampling uses seed 20260824, temperature 0.8, top-$p = 0.95$, a 6,144-token prompt limit, a 4,096-token response limit, and a 10,240-token model context. Corruption and repair are constructed primarily by the deterministic semantic/AST compiler, supplemented by deterministic, gold-guided-local, and natural-sibling construction paths. The full acquisition issues no new external or closed-model generation requests.

F Experimental Setup Details

F.1 Hardware and Software

All experiments were conducted on a server equipped with four NVIDIA A100 GPUs with 80 GB of memory per GPU, 256 GB of system RAM, and two AMD EPYC 7643 CPUs. The software stack used vLLM version 0.26.0 and verl version 0.9.0.

F.2 Evaluation Protocol

Scope and splits. We evaluate SQL generation on the official public-development sets of BIRD, SParC, and CoSQL; BIRD contributes 1,534 public-development questions. We do not infer unavailable hidden labels or present public proxies as hidden-test results. Missing databases or scoring assets, unsupported constructs, malformed dialogues, and unavailable references are recorded as coverage exclusions; execution errors, timeouts, invalid generations, and missing submissions remain in the evaluation denominator and receive zero task credit.

Common inputs and decoding. Comparable methods share the system prompt, chat template, schema renderer, dialogue history, database interface, tool serialization, context limit, and evaluator.

The schema exposes table and column names, types, primary keys, and foreign keys; gold SQL, execution results, reward scores, root-cause labels, and evaluator-only metadata are hidden. Final benchmark evaluation uses greedy decoding (temperature 0, top-$p = 1$, one generation). Verifier generation is greedy with a 1,024-token limit. Verifier-guided repair uses temperature 0.8, top-$p = 0.95$, at most 2,048 new tokens per suffix reroll, and two matched rerolls per intervention.

SQL generation. For BIRD, the model receives the question, public evidence when available, and the schema. For SParC and CoSQL, it receives the complete history with the model's preceding predictions unchanged. The local agent can call `execute_sql(sql)` and terminate with `submit_solution(sql_list)`. It uses one tool call per turn, at most five assistant turns and eight attempted tool calls, and a 500-second trajectory limit. Each trajectory has an isolated SQLite copy; invalid SQL, execution errors, timeouts, and missing submissions are incorrect under the official evaluator.

SQL debugging. The debugging model receives the issue description, faulty SQL, schema, and benchmark database context, but no reference repair or hidden test outcomes. It uses the same tool interface, submits within five turns, and is successful only when the repair passes all released curated tests on a fresh isolated database within the 30-second evaluator timeout. Partial test success and all malformed or incomplete trajectories are failures.

Interactive Text-to-SQL. We use the official BIRD-Interact-ADK implementation on BIRD-INTERACT-Lite (300 tasks over 18 PostgreSQL databases) for c-Interact and a-Interact. We retain the benchmark's user simulator, action spaces, clarification and follow-up workflow, debugging opportunity, database state transitions, read-only exploration, BIRD-coin budgets, isolation, termination rules, and official reward aggregation. Normalized reward, success, Turns, and Coin are reported as returned by the official evaluator without rescaling.

Step PRM baseline. Step PRM uses the same preference data and process-credit injection as DA-PRM. Its only change is the credit granularity: it scores and credits each policy-generated action as one unit, rather than the decision-aligned spans within an action. The GRPO rollout and update budget and $\lambda_{\text{proc}} = 0.5$ are shared with DA-PRM.

F.3 Inference Prompts

The following system messages and user-message templates specify the local SQL-generation and SQL-debugging interfaces. Their tool-use format is adapted from the BIRD-RL repository. Braced placeholders are filled from each task instance; earlier turns and tool responses are supplied as conversation history when present.

F.3.1 SQL-Generation Prompt

System message

```
You are an expert SQL query writer. Your task is to translate natural
language questions into correct SQL queries.

You will be provided with:
1. A database schema with sample data
2. Column descriptions
3. A natural language question
4. Evidence/hints about domain-specific terms

## Your Objective

Generate the correct SQL query by:
- Analyzing the question to understand what data is needed
- Using tools to explore the database schema and sample data
- Testing candidate queries to validate your approach
- Submitting the final correct SQL query
```

```
## Available Tools

<tools>
[
  {
    "type": "function",
    "function": {
      "name": "execute_sql",
      "description": "Execute a SQL query against the database for
      exploration and testing. Returns execution results but does NOT
      evaluate correctness. Use this to test and refine your SQL before
      submitting the final solution.",
      "parameters": {
        "type": "object",
        "properties": {
          "sql": {
            "type": "string",
            "description": "A single SQL query to execute for exploration
            and testing."
          }
        },
        "required": ["sql"]
      }
    }
  },
  {
    "type": "function",
    "function": {
      "name": "submit_solution",
      "description": "Submit your final SQL solution. This ends the current
      task. The SQL will be executed and evaluated against test cases. Only
      call this when you are confident your solution is correct, as this is
      your final submission.",
      "parameters": {
        "type": "object",
        "properties": {
          "sql_list": {
            "type": "array",
            "items": {
              "type": "string"
            },
            "description": "List of SQL statements for the final solution.
            Should contain the complete, correct solution to the problem."
          }
        },
        "required": ["sql_list"]
      }
    }
  }
]
</tools>

## How to Call Tools

To invoke a tool, the output must be in the following format:

1. Open the tool call tag: <tool_call>
2. Provide a valid JSON object with exactly two keys:
   - "name": string - the name of the function to call
     ("execute_sql" or "submit_solution")
   - "arguments": object - a JSON object containing the parameters
     for that tool
3. Close the tool call tag: </tool_call>

**Format examples:**

For testing/exploration:
<tool_call>{"name": "execute_sql",
"arguments": {"sql": "SELECT * FROM table LIMIT 5"}}</tool_call>

For final submission:
<tool_call>{"name": "submit_solution",
"arguments": {"sql_list":
["SELECT col FROM table WHERE condition"]}}</tool_call>

## Critical Format Requirements

Your trajectory MUST follow this exact structure for EVERY turn:
```

```
1. **<think> tags**: Concise, high-level reasoning about what you're
   doing and why.

2. **Action**: After thinking, take ONE action by calling a tool:
   - Use execute_sql for exploration and testing
   - Use submit_solution for final submission (ends trajectory)

3. **Turn limit**: Maximum {max_turns} turns. Each turn:
   <think> + <tool_call>. **CRITICAL**: If you don't call
   submit_solution in the first {prev_turns} turns, the last turn
   MUST be submit_solution (forced submission).
```

User message template

```
## Database Schema
{schema}

## Column Descriptions
{column_descriptions}

## Question
{question}

## Evidence
{evidence}

**Your task**: Generate the next single turn of the SQL generation
trajectory. This means:
1. One <think> block with concise reasoning (no SQL code in thoughts)
2. Followed by ONE <tool_call>: either execute_sql to explore/test OR
   submit_solution for the final answer

Generate only the next turn. Maximum {max_turns} turns allowed. Review
the conversation history to determine your current turn. **If this is
the final turn, you MUST call submit_solution.**

**Note**: If conversation history is provided below, continue from
where the exploration left off. If this is the first turn, start by
analyzing the question and exploring the database.
```

F.3.2 SQL-DEBUGGING PROMPT

System message

```
You are an expert SQL debugger with deep knowledge of database systems
and SQL.

Your task is to debug and fix problematic SQL queries. You will be
provided with:
1. A natural language query describing what the SQL should do
2. A database schema
3. A problematic SQL query that contains errors

## Your Objective

Debug the problematic SQL by:
- Identifying issues through systematic exploration
- Using tools to test hypotheses and validate your understanding
- Building toward the correct solution incrementally
- Demonstrating sound reasoning at each step

Your debugging process should be thorough and realistic, showing how
an expert would approach the problem from first principles.

## Available Tools

The SQL debugger has access to the following tools:

<tools>
[
  {
    "type": "function",
    "function": {
      "name": "execute_sql",
      "description": "Execute SQL queries against the database to test
      and debug them. Use this for exploration and testing. Returns
      execution results but does NOT evaluate correctness. Use this to
      iterate and refine your SQL before submitting the final solution.",
      "parameters": {
        "type": "object",
        "properties": {
          "sql": {
            "type": "string",
            "description": "A single SQL query to execute. Use this for
            testing individual queries during exploration."
          }
        },
        "required": ["sql"]
      }
    }
  },
  {
    "type": "function",
    "function": {
      "name": "submit_solution",
      "description": "Submit your final SQL solution. This ends the
      current task. The SQL will be executed and evaluated against test
      cases, and you will receive immediate pass/fail feedback with
      details. Only call this when you are confident your solution is
      correct, as this is your final submission.",
      "parameters": {
        "type": "object",
        "properties": {
          "sql_list": {
            "type": "array",
            "items": {
              "type": "string"
            },
            "description": "List of SQL statements for the final
            solution. Should contain the complete, correct solution to
            the problem."
          }
        },
        "required": ["sql_list"]
      }
    }
```

```
  }
]
</tools>

## How to Call Tools

To invoke a tool, the output must be in the following format:

1. Open the tool call tag: <tool_call>
2. Provide a valid JSON object with exactly two keys:
   - "name": string - the name of the function to call
     ("execute_sql" or "submit_solution")
   - "arguments": object - a JSON object containing the parameters
     for that tool
3. Close the tool call tag: </tool_call>

**Format examples:**

For testing/exploration (single query):
<tool_call>{"name": "execute_sql",
"arguments": {"sql": "SQL statement"}}</tool_call>

For final submission (can be multiple statements):
<tool_call>{"name": "submit_solution",
"arguments": {"sql_list":
["SQL statement 1", "SQL statement 2", ...]}}</tool_call>

**Requirements**:
- The JSON must be valid and parseable
- For execute_sql, "arguments" must contain "sql" as a string
- For submit_solution, "arguments" must contain "sql_list" as an array of strings
- Each string in "sql_list" is one SQL statement
- Do NOT use markdown formatting inside tool_call tags

## Critical Format Requirements

Your generated trajectory MUST follow this exact structure for EVERY
turn:

1. **<think> tags**: Provides a concise, high-level summary of the
   model's action for the current turn. It is intended for decision
   tracing rather than exposing internal reasoning.

2. **Action**: After thinking, take ONE action by calling a tool:
   - Use execute_sql for testing and exploration
     (single query, no evaluation)
   - Use submit_solution for final submission
     (can be multiple statements, runs test cases, gives immediate
     feedback, ends trajectory)

3. **Turn limit**: Maximum {max_turns} turns. Each turn:
   <think> + <tool_call>. **CRITICAL**: If you don't call
   submit_solution in the first {prev_turns} turns, the last turn
   MUST be submit_solution (forced submission).
```

User message template

```
## Problem Description
{query}

## Database Schema
{schema}

## Problematic SQL
The following SQL query contains errors and does not correctly satisfy
the user's requirements:

'''sql
{issue_sql}
'''

**Your task**: Provide the next single turn of the debugging
trajectory. This means:
```

```
1. One <think> block explaining your reasoning for the next action
2. Followed by ONE <tool_call>: either execute_sql to explore/test OR
   submit_solution for the final answer

Generate only the next turn. Maximum {max_turns} turns allowed. Review
the conversation history to determine your current turn. **If this is
the final turn, you MUST call submit_solution.**

**Note**: If conversation history is provided below (previous thinking,
tool calls, and tool responses), continue from where the debugging left
off. Analyze the previous results and determine the next logical action.
If this is the first turn (no history), start by analyzing the problem
and deciding on an initial exploration strategy.
```

F.4 Datasets and Metrics

SQL generation. We use BIRD (Li et al., 2023) for single-turn SQL generation and the context-dependent SParC (Yu et al., 2019b) and CoSQL (Yu et al., 2019a) benchmarks for multi-turn SQL generation. We follow the official evaluation scripts: BIRD is evaluated by execution accuracy (EX), while question-level and interaction-level execution accuracy (QEX and IEX) are defined for both SParC and CoSQL. Tables 1–3 report QEX; IEX requires separate interaction-level results and is not inferred from QEX.

SQL debugging. We evaluate SQL debugging on BIRD-CRITIC-SQLite (Li et al., 2025b), which contains 500 user-reported SQL issues from real-world SQLite database applications. Each instance provides a problem description, a faulty SQL query, and the target database environment. A prediction is successful only if the repaired SQL passes all curated test cases; we therefore report the benchmark's task-resolution Success Rate (SR).

Interactive Text-to-SQL. We use BIRD-INTERACT-Lite (Huo et al., 2026), which contains 300 PostgreSQL tasks evaluated through conversational (c-Interact) and agentic (a-Interact) interaction protocols. A trajectory can involve clarification, knowledge retrieval or environment exploration, SQL generation, and debugging after execution failures. We report the official normalized reward, task-level SR , as well as interaction efficiency, measured by conversational turns for c-Interact and BIRD-coin cost for a-Interact.

Metric definitions. Let $\hat{y}_i$ denote a predicted SQL query, $y_i^\star$ its reference query, D_i the corresponding database, and $\mathrm{Exec}(y, D)$ the execution result of y on D. Execution accuracy is

$$\mathrm{EX} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}[\mathrm{Exec}(\hat{y}_i, D_i) = \mathrm{Exec}(y_i^\star, D_i)]\,. \tag{39}$$

For question-level and interaction-level reporting, we use

$$\begin{aligned} \mathrm{QEX} &= \frac{1}{N_Q}\sum_{q=1}^{N_Q} \mathbf{1}[q \text{ is execution-correct}], \\ \mathrm{IEX} &= \frac{1}{N_I}\sum_{j=1}^{N_I} \mathbf{1}[\text{all questions in interaction } j \text{ are correct}]. \end{aligned} \tag{40}$$

For SQL debugging, let $\mathcal{C}_i$ be the curated test cases for instance i. The task-resolution success rate is

$$\mathrm{SR} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}[\forall c \in \mathcal{C}_i,\ \mathrm{Pass}(\hat{y}_i, c) = 1]\,. \tag{41}$$

For interactive Text-to-SQL, r_i^{norm} is the official normalized reward returned for trajectory i, and s_i is its official task-success indicator:

$$\mathrm{NR} = \frac{1}{N}\sum_{i=1}^{N} r_i^{\mathrm{norm}}, \qquad \mathrm{SR}_{\mathrm{task}} = \frac{1}{N}\sum_{i=1}^{N} s_i. \tag{42}$$

The efficiency measures are the per-task averages

$$\text{Turns} = \frac{1}{N}\sum_{i=1}^{N} T_i, \qquad \text{Coin} = \frac{1}{N}\sum_{i=1}^{N} C_i, \tag{43}$$

where T_i is the number of conversational turns and C_i is the BIRD-coin cost. Higher is better for EX, QEX, IEX, SR, and NR; lower is better for Turns and Coin.

F.5 PRM Verifier Metrics

We evaluate the DA-PRM-based verifier at root-event detection, localization resolution, and repair outcome. Let v_i^* and $\hat{v}_i$ denote the gold and predicted verdicts, respectively, and let z_i indicate that the output passes the verifier's JSON schema. Invalid outputs are treated as having no prediction and contribute zero to all metrics below. With $\mathcal{I}^- = \{i : v_i^* = \text{incorrect}\}$, the prediction rate is

$$\text{Pred} = \frac{1}{|\mathcal{I}^-|}\sum_{i\in\mathcal{I}^-} \mathbf{1}[z_i = 1 \wedge \hat{v}_i = \text{incorrect}]. \tag{44}$$

For each trajectory, let $e_{i,r}$ be the r-th predicted suspected-error event, $\mathcal{E}_i^*$ the set of gold root events, and $\mathcal{G}_i(e)$ the gold root spans in event e. Root-event detection at rank k measures whether any of the top-k suspected errors identifies a gold root event, averaged over the $N_{\text{err}} = |\mathcal{I}^-|$ gold-incorrect trajectories:

$$\text{R@}\,k = \frac{1}{N_{\text{err}}}\sum_{i\in\mathcal{I}^-} \mathbf{1}[z_i = 1 \wedge \exists r \le k : e_{i,r} \in \mathcal{E}_i^*], \qquad k \in \{1, 3\}. \tag{45}$$

Missing or invalid predictions count as misses.

For the span metrics, character offsets are half-open intervals. On every $i \in \mathcal{I}^-$, Span IoU evaluates only the first verifier localization `suspected_errors[0]`. Span resolution uses trusted character offsets when available; otherwise, the predicted span text must occur exactly once in the predicted event's content, and zero or multiple occurrences make the localization invalid. Let the resolved predicted character interval be $P = [p_s, p_e)$, and let a gold root span in the same event be $G = [g_s, g_e)$. Their intersection and union lengths are

$$|P\cap G| = \max(0, \min(p_e, g_e) - \max(p_s, g_s)), \qquad |P\cup G| = (p_e - p_s) + (g_e - g_s) - |P\cap G|, \tag{46}$$

and

$$\text{SpanIoU}(P, G) = \frac{|P \cap G|}{|P \cup G|}. \tag{47}$$

If multiple gold root spans exist in that event, the score uses the maximum IoU over them; if the predicted event is not a gold root event, or if the first localization is missing or invalid, the score is zero. Gold-span coverage (the figure's "Span" metric) is the corresponding character recall, $|P \cap G|/|G|$, averaged over $\mathcal{I}^-$. Given a correct root event, region precision is $|P \cap G|/|P|$. Token Span F1 maps the two character intervals to token sets $T(P)$ and $T(G)$ and computes

$$\text{TokenF1}(P, G) = \frac{2|T(P) \cap T(G)|}{|T(P)| + |T(G)|}, \tag{48}$$

with zero for an invalid or empty prediction.

For repair outcome, let $\mathcal{I}_{\text{rep}}$ be the eligible repaired instances, and let R_i^{before} and R_i^{after} be their pre- and post-repair rewards. The figure reports

$$\text{RepairRate} = \frac{1}{|\mathcal{I}_{\text{rep}}|}\sum_{i\in\mathcal{I}_{\text{rep}}} \mathbf{1}[R_i^{\text{after}} > 0], \qquad \text{RerollGain} = \frac{1}{|\mathcal{I}_{\text{rep}}|}\sum_{i\in\mathcal{I}_{\text{rep}}} (R_i^{\text{after}} - R_i^{\text{before}}). \tag{49}$$

For matched intervention and no-intervention rerolls, let R_i^+ and R_i^- denote their respective rewards. Paired gain is the mean difference $R_i^+ - R_i^-$. Non-target preservation is the fraction of eligible repairs whose non-target semantic-unit set is unchanged. The figure displays rates on the $[0, 1]$ scale.

G Case Study: PRM as Verifier for Semantic Error Localization

We hold the full trajectory fixed and compare Exec-RM, Step PRM, and the DA-PRM-based verifier. The latter differs from DA-PRM only in its output head: the verifier head produces structured diagnoses in place of the scalar process score. Figure 4 illustrates this pattern through a denominator error.

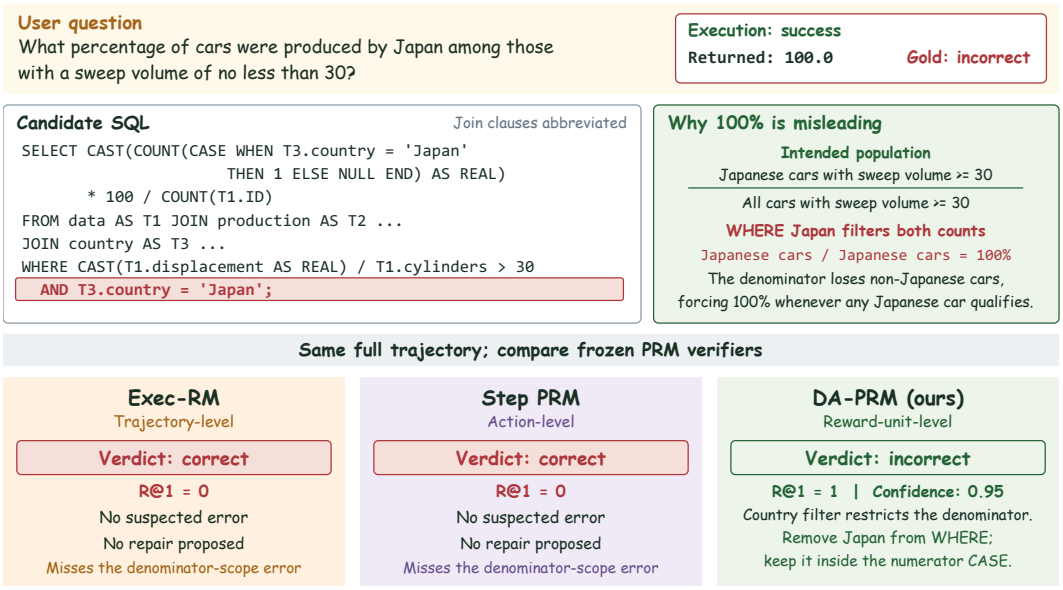


Figure 4: Semantic error localization with PRM verifiers. An executable query returns `100.0` because a country predicate restricts both counts to Japanese cars. Only the DA-PRM-based verifier rejects the query and identifies the gold root event. R@1 is the per-example root-event hit indicator (Section F.5).

An executable query with the wrong denominator. The question asks for the percentage of Japanese cars among cars with a sweep volume of no less than 30, so the denominator must include all countries. The candidate counts Japan conditionally in the numerator but also applies `T3.country = 'Japan'` in `WHERE`. This removes non-Japanese rows before both aggregates, forcing 100% whenever qualifying Japanese cars exist. The observed result, `100.0`, is executable and plausible in range despite the incorrect population.

Localization and proposed repair. Both baseline verifiers return `correct`, an empty suspected-error list, and no repair. The DA-PRM-based verifier returns `incorrect`, ranks the gold root event first, and reports 0.95 in its confidence field. Its explanation connects the country predicate to the aggregation scope: applying the condition before aggregation incorrectly restricts the denominator. The proposed repair removes the country predicate from `WHERE` while retaining it inside the numerator's `CASE` expression. This restores the denominator's coverage of all countries. The recorded repair retains the original sweep-volume comparison `> 30`, whereas "no less than 30" requires `>= 30`; it thus demonstrates correction of the denominator scope, with the boundary condition still unresolved.

Related semantic interactions. Two further cases show the same verdict and Rank-1 localization pattern. For a request for movies rated between 3 and 5, DA-PRM proposes removing `T1.stars`, an extra projection inherited from the previous turn. For "Of those three, who is the oldest?" it proposes replacing `T3.Age` with `T3.Name` while preserving age-based ordering. These cases connect a local SQL decision to its intended answer semantics.

H ETHICS AND RESOURCE LICENSES

This study reuses released benchmark data and involves no new human annotations or collection of human-subject data. Tables 6 and 7 record the licenses declared on official project pages and publisher-hosted Hugging Face cards.

Datasets. All six benchmark and training datasets listed below declare CC BY-SA 4.0. SIX-GYM-SQLITE is included because it supplies SFT training and development records, even though it is not a separate main-results benchmark.

Table 6: Declared licenses of benchmark and training datasets. Links point to official project pages or publisher dataset cards.

Dataset	Declared license	Official source
BIRD	CC BY-SA 4.0	BIRD project page
SParC	CC BY-SA 4.0	SParC project page
CoSQL	CC BY-SA 4.0	CoSQL project page
BIRD-CRITIC-SQLite	CC BY-SA 4.0	birdsql / bird-critic-1.0-sqlite
BIRD-INTERACT-Lite	CC BY-SA 4.0	birdsql / bird-interact-lite
SIX-GYM-SQLITE	CC BY-SA 4.0	birdsql / six-gym-sqlite

Model weights and backbone-controlled baselines. The released-weight baselines and Qwen3.5-9B backbone declare Apache 2.0. The backbone-controlled policies and PRMs are trained from Qwen3.5-9B.

Table 7: Declared licenses of released model weights and the shared backbone.

Model	Declared license	Official source
Qwen3.5-9B (shared backbone)	Apache 2.0	Qwen model card
MTSQL-R1-4B	Apache 2.0	taicheng model card
SQL-R1-7B	Apache 2.0	SQL-R1-7B model card
SQL-R1-14B	Apache 2.0	SQL-R1-14B model card
OmniSQL-7B	Apache 2.0	seeklhy model card
XiYanSQL-QwenCoder-14B	Apache 2.0	2504 card
Qwen3.8-27B	Apache 2.0	Qwen model card
Gemma-4-31B-it	Apache 2.0	Google model card

Google's Gemma 4 license page also specifies Apache 2.0. The Apache 2.0 terms require a license copy, retention of applicable notices, and identification of modified files when redistributing covered material.

Evaluation software. The official BIRD-CRITIC-1, BIRD-Interact, and BIRD-RL repositories carry MIT license files. This software license requires retention of the copyright and permission notice in redistributed copies and is distinct from the benchmarks' CC BY-SA 4.0 data license.

Service-access baselines. API access to `GPT-5.6-Terra`, `Grok 4.6`, `Claude Sonnet 5`, `Gemini 3.1 Pro Preview`, `GLM-5.2`, and `DeepSeek-V4-Pro-0813` follows provider and hosting-service terms, separate from weight licenses. Official sources include Anthropic's commercial terms, OpenAI's services agreement, xAI's enterprise terms, Gemini API terms, Z.ai's terms, and DeepSeek's terms.