CAPER: Clause-Aligned Process Supervision for Text-to-SQL

Lujie Ban
School of Data Science
The Chinese University of Hong Kong, Shenzhen
Shenzhen, China
lujieban@link.cuhk.edu.cn

Jiasheng Shi
School of Data Science
The Chinese University of Hong Kong, Shenzhen
Shenzhen, China
shijiasheng@cuhk.edu.cn

Jinyang Li
Department of Computer Science
The University of Hong Kong
Pok Fu Lam, Hong Kong
jl0725@connect.hku.hk

Xiaolin Han
School of Computer Science
Northwestern Polytechnical University
Xi'an, Shaanxi
xiaolinh@nwpu.edu.cn

Tsz Nam Chan
College of Computer Science and Software Engineering
Shenzhen University
Shenzhen, China
edisonchan@szu.edu.cn

Chenhao Ma*
School of Data Science
The Chinese University of Hong Kong, Shenzhen
Shenzhen, China
machenhao@cuhk.edu.cn

Abstract

Text-to-SQL systems are typically evaluated by query-level execution correctness, but this terminal signal provides little guidance about which intermediate SQL decision caused success or failure. Token-level dense supervision is also ill-suited: SQL tokens do not align with complete semantic decisions, can penalize execution-equivalent queries, and are difficult to label reliably at scale. We therefore propose `CAPER`, which automatically derives clause-level supervision via counterfactual intervention on the SQL abstract syntax tree, enabling root-cause error localization for reward modeling; the resulting data is used to train `CAPER-9B`, a lightweight Clause-PRM that provides clause-boundary feedback for policy optimization and candidate verification. Experiments on BIRD and Spider show that clause-aligned supervision improves BIRD MV@8 execution accuracy by 7.42 percentage points over the backbone-matched `GRPO-Sparse` baseline, while reaching 84.53% failure-localization accuracy and 90.60% MRR on held-out failures. Our project page is at https://github.com/banrichard/RL-NL2SQL.

1 Introduction

Text-to-SQL aims to translate natural-language questions into executable SQL queries, providing a direct interface between non-expert users and structured databases [1, 2]. In real-world data workflows, it supports natural-language analytics [3, 4], business-intelligence question answering [5, 6], enterprise database operations [7–9], and interactive SQL troubleshooting [10, 11]. Large language models (LLMs) have further expanded the design space through schema linking [12, 13], database retrieval [14, 15], and generation correction [16, 17]. Despite this progress, Text-to-SQL remains a high-precision structured generation problem: a generated query must be syntactically valid, schema-grounded, and semantically equivalent to the intended database operation, and a single wrong join key, aggregation, or predicate can still lead to the wrong execution result.

*Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

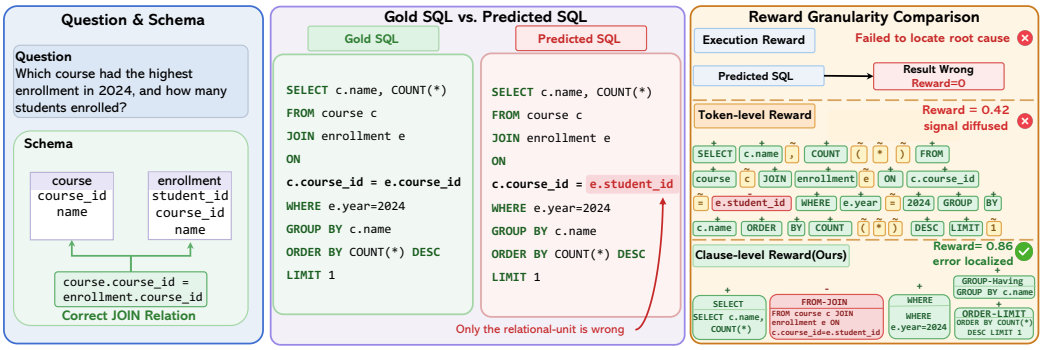

Figure 1: A near-miss Text-to-SQL case illustrating the granularity gap between execution-level outcomes, token-level feedback, and clause-level SQL decisions.

One way to improve Text-to-SQL is to scale supervised fine-tuning (SFT) [18] with larger or richer demonstration corpora. But this shifts the bottleneck to data construction: executable Text-to-SQL annotations are expensive to produce, require SQL expertise, and remain costly to scale [10, 11, 19–23]. In the LLM era, strong SFT pipelines often also depend on richer supervision such as reasoning traces [24]. Query-level execution outcomes offer a more scalable source of supervision because they are already used to judge whether a generated SQL query retrieves the correct answer [25–28]. However, an execution outcome is only a terminal label: it says whether the full query succeeds, but not which intermediate SQL decision made it succeed or fail.

This creates a credit assignment problem for process supervision. As illustrated in Figure 1, a predicted SQL query can be largely correct yet fail because of one decisive semantic error, such as an incorrect relational condition or join path. Execution-level feedback treats this near-miss the same as a completely wrong query: once the final execution result is incorrect, the signal provides little guidance about which intermediate SQL decision should be reinforced, repaired, or discouraged. In other words, query-level outcomes provide supervision without root-cause localization: they tell us that a query is wrong, but not which SQL decision caused the failure.

A naïve remedy is to move from execution-level feedback to denser token-level feedback. However, token-level supervision is often too fine-grained for SQL semantics. Individual tokens such as aliases, commas, operators, or column names rarely form complete decisions on their own, while many SQL decisions are expressed by multi-token structures such as selected column sets, predicates, aggregations, or join paths. As a result, token-level feedback can diffuse credit across low-level symbols and require unreliable fine-grained labels.

These limitations suggest that the right unit of process supervision should lie between the two extremes: denser than execution-level feedback but more semantically grounded than token-level feedback. We therefore focus on clause-level SQL units, which naturally group tokens into meaningful intermediate decisions such as `SELECT`, `FROM/JOIN`, `WHERE`, and `GROUP BY/HAVING`. Clause-level supervision preserves the scalability of outcome-derived feedback while assigning credit at a granularity closer to how SQL errors affect execution. The key missing step is to localize the root cause of failure at this semantic granularity before converting it into process supervision. This raises question Q : *how can we derive clause-aligned supervision for Text-to-SQL from query-level execution outcomes, without requiring costly manual annotations while preserving execution correctness as the task objective?*

Our Solution. To solve question Q , we propose ❶ `CAPER`, an automatic framework that uses counterfactual intervention to localize root-cause SQL decisions and construct clause-level preference supervision, and ❷ `CAPER-9B`, a lightweight clause-level process reward model trained for process supervision. Methodologically, `CAPER` leverages ♣ **counterfactual intervention**, systematically perturbing the abstract syntax tree (AST) to identify the decisive error step responsible for failure. To enhance data diversity and ensure annotation precision, it further adopts ♠ **fault injection**, synthetically generating failure instances by perturbing successful ASTs through targeted corruption.

Further, we fine-tune `Qwen3.5-9B` on the annotated dataset to obtain `CAPER-9B` and deploy it as a process reward model in reinforcement learning (RL) for Text-to-SQL, providing fine-grained

clause-level rewards for intermediate SQL decisions. A **multi-granular reward** is designed to supervise RL training, emphasizing both *clause-level* and *execution-level* feedback. During policy optimization, this multi-granular reward design provides the policy model with both clause-level process feedback and execution-level outcome feedback, yielding denser supervision while preserving final execution accuracy as the task objective.

In summary, our contributions are:

❶ **Automated Pipeline.** We propose a counterfactual SQL-AST pipeline for root-cause localization and clause-level credit assignment without manual annotation, which yields over 90,000 clause-annotated preference tuples across three datasets.

❷ **Clause-level PRM.** We develop `CAPER-9B`, a lightweight clause-level process reward model for Text-to-SQL RL optimization through clause-boundary dense rewards, combining intermediate process feedback with final execution correctness.

❸ **Empirical Evaluation.** RL with `CAPER-9B` improves BIRD MV@8 execution accuracy by 7.42 percentage points over the backbone-matched `GRPO-Sparse` baseline. `CAPER-9B` also reaches 84.53% top-1 failure-localization accuracy and 90.60% MRR on held-out failures.

2 Preliminary

In this section, we introduce the definition of Text-to-SQL and the objective of a process reward model.

Text-to-SQL. Text-to-SQL is a task that converts a natural language question $\mathcal{Q}$ into a SQL query y capable of retrieving the correct answer from a database [19]. Given a database $\mathcal{D} = \langle \mathcal{C}, \mathcal{T} \rangle$, where $\mathcal{C}$ and $\mathcal{T}$ denote the sets of columns and tables, respectively, the Text-to-SQL task can be formulated as:

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

where $f(\cdot \mid \theta)$ is the text-to-SQL model parameterized by θ.

Process Reward Model. The process reward model (PRM) evaluates the correctness of each intermediate reasoning step, providing step-level supervision beyond the final answer [29]. Given an input x and a step-by-step solution $z = (z_1, \ldots, z_K)$, the reward of the k-th step can be formulated as:

$$f_\psi(x, z_{\le k}) \in \mathbb{R}, \quad q_\psi(\ell_k = \text{positive} \mid x, z_{\le k}) = \sigma(f_\psi(x, z_{\le k})), \tag{2}$$

where f_ψ is the real-valued PRM score parameterized by ψ, q_ψ is the probability induced by the sigmoid link function, $z_{\le k}$ is the solution prefix up to step k, and ℓ_k is the process label of step z_k. A conventional PRM can rank a complete solution by multiplying step probabilities,

$$R^{\text{rank}}_{\text{PRM}}(x, z) = \prod_{k=1}^{K} q_\psi(\ell_k = \text{positive} \mid x, z_{\le k}), \tag{3}$$

which estimates the probability that all intermediate steps are correct. In our RL formulation, we instead use the real-valued step scores as additive reward signals under the MDP return objective.

3 Methodology

In this section, we propose `CAPER`, which first constructs clause-level preference supervision from successful and failed SQL trajectories, then trains a Clause-Level Process Reward Model (Clause-PRM) on the resulting annotations, and finally uses it to provide clause-boundary rewards during Text-to-SQL policy optimization, as demonstrated in Figure 2.

3.1 Clause-Level Text-to-SQL Formulation

For policy optimization, we model SQL generation as an episodic MDP so that clause-aligned process scores can be incorporated into the trajectory return. At token step t, the state is the input and current SQL prefix $s_t = (x, a_{<t})$ with $x = (\mathcal{Q}, \mathcal{D})$, and the action a_t appends the next SQL token to the prefix. Under a policy $\pi_\theta(a_t \mid s_t)$, a rollout trajectory is

$$\tau = (s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_T, a_T, r_T, s_{T+1}). \tag{4}$$

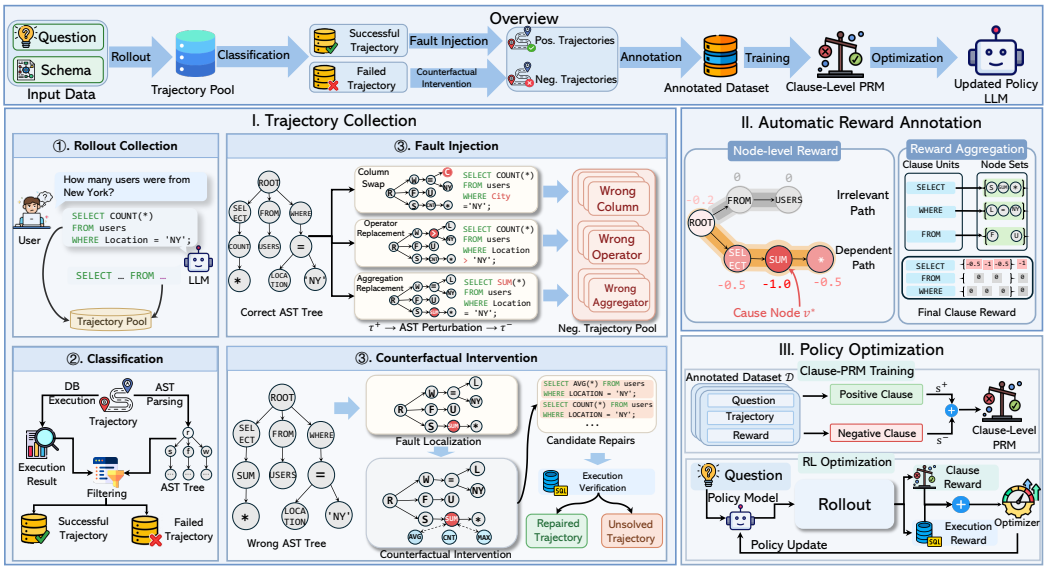

Figure 2: Overview of the CAPER framework.

Let $G(\tau) = \sum_{t=1}^{T} \gamma^{t-1} r_t$ denote its return. The policy objective is

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[G(\tau)]. \tag{5}$$

Although rollouts are token-level, intermediate feedback should operate at the clause level, since SQL errors usually arise from a small number of incorrect top-level clause decisions. We therefore instantiate the generic PRM step in Equation (2) as a top-level SQL clause unit.

Definition 1 (Clause Unit)**.** A clause unit is a semantically coherent top-level SQL component. We denote the k-th clause unit by $u_k^{\kappa_k}$, where u_k is its content and $\kappa_k \in \mathcal{K}$ is its type:

$$\begin{aligned}\mathcal{K} = \{&\textsc{With}, \textsc{Select}, \textsc{From-Join}, \textsc{Where},\\ &\textsc{Group-Having}, \textsc{Window}, \textsc{Order-Limit}, \textsc{Set-Op}\}.\end{aligned} \tag{6}$$

Here, WITH covers CTEs; FROM-JOIN groups `FROM/JOIN/ON`; GROUP-HAVING groups `GROUP BY/HAVING`; WINDOW covers window functions and specifications; ORDER-LIMIT covers `ORDER BY/LIMIT/OFFSET`; and SET-OP covers `UNION/INTERSECT/EXCEPT`. We decompose only the top-level SQL structure; clauses inside nested subqueries are included in their enclosing clause unit.

At the clause level, a generated SQL query is decomposed as

$$y = (u_1^{\kappa_1}, u_2^{\kappa_2}, \ldots, u_K^{\kappa_K}). \tag{7}$$

Evaluating the k-th clause requires the preceding clause context, which we denote as

$$p_k = (u_1^{\kappa_1}, \ldots, u_{k-1}^{\kappa_{k-1}}). \tag{8}$$

We then instantiate the PRM in Equation (2) as a real-valued clause preference scorer:

$$f_\psi(x, p_k, u_k^{\kappa_k}) \in \mathbb{R}, \tag{9}$$

where higher scores indicate clause units that are more consistent with the correct SQL continuation under (x, p_k).

3.2 Automatic Clause-Level Reward Annotation

The annotation stage of CAPER converts coarse trajectory-level outcomes into fine-grained clause-level reward annotations. We begin by collecting a trajectory pool from a supervised fine-tuned policy, denoted by π_{SFT}. Let $\mathcal{X} = \{(x_i, y_i)\}_{i=1}^{N}$ be the set of training queries, where x_i is the

input question–schema pair and y_i is the corresponding gold SQL query. For each input x_i, the SFT model generates one rollout trajectory $\tau_i \sim \pi_{\mathrm{SFT}}(\cdot \mid x_i)$, which induces a terminal SQL query $\hat{y}(\tau_i)$. This trajectory collection is separate from tuple construction: a single retained rollout can yield multiple clause-level preference tuples because fault injection may perturb several editable nodes and counterfactual repair may proceed across multiple divergent clauses. The collected trajectories are partitioned by their execution outcome. Let

$$\Omega(\tau_i) = \begin{cases} 1, & \text{if } \hat{y}(\tau_i) \text{ is execution-equivalent to } y_i, \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

be the indicator function for trajectory classification. Accordingly, we define

$$\mathfrak{T}^+ = \{\tau_i : \Omega(\tau_i) = 1\}, \text{ and } \mathfrak{T}^- = \{\tau_i : \Omega(\tau_i) = 0\}. \tag{11}$$

The successful set $\mathfrak{T}^+$ provides trusted positive trajectories for fault injection, whereas the failed set $\mathfrak{T}^-$ provides naturally occurring error trajectories for counterfactual intervention. Our goal is to construct clause-level reward annotations by either inducing a root-cause error on $\mathfrak{T}^+$ or localizing a root-cause error on $\mathfrak{T}^-$.

For each trajectory τ, we parse its terminal SQL $y(\tau)$ into an abstract syntax tree (AST) $G(\tau) = (V, E)$. According to Definition 1, each generated unit is associated with a subset of AST nodes through a clause-to-node mapping $\phi(u_k^{\kappa_k}) = V_k \subseteq V$. This mapping enables us to translate node-level structural errors on the AST into clause-level reward supervision for intermediate decisions along the trajectory.

Fault Injection for Successful Trajectories. For $\tau_i \in \mathfrak{T}^+$, the generated SQL is execution-correct and therefore provides a trusted positive reference. Using its AST, we sample one or more editable key nodes $v^\star \in V$ and apply a type-preserving mutation operator μ to obtain corrupted trees $G' = \mu(G, v^\star)$. Each corrupted sample is retained only if its execution result differs from that of the original query, ensuring that the mutation induces a genuine semantic error.

Counterfactual Intervention for Failed Trajectories. For $\tau_i \in \mathfrak{T}^-$, we recover clause-level supervision by comparing the predicted SQL against its gold counterpart. Let $u_k^{\kappa_k}$ and $\tilde{u}_k^{\kappa_k}$ denote the predicted and gold clause units. We first identify the earliest divergent clause index $k^\star = \min\{k : u_k^{\kappa_k} \neq \tilde{u}_k^{\kappa_k}\}$ and then construct a counterfactual corrected query through an AST-level repair under the same prefix:

$$y^{\mathrm{cf}} = \mathrm{Intervene}(\hat{y},\, p_{k^\star},\, u_{k^\star}^{\kappa_{k^\star}} \mapsto \tilde{u}_{k^\star}^{\kappa_{k^\star}}), \tag{12}$$

where $\mathrm{Intervene}(\cdot)$ reparses the whole SQL after repair, canonicalizes aliases introduced by the repaired clause, and retains the counterfactual only when the resulting query is syntactically valid and executable. If a single repaired clause does not yield a valid counterfactual, we repeat on the earliest remaining divergence; each retained repair contributes a clause-level tuple. Figure 3 illustrates this process.

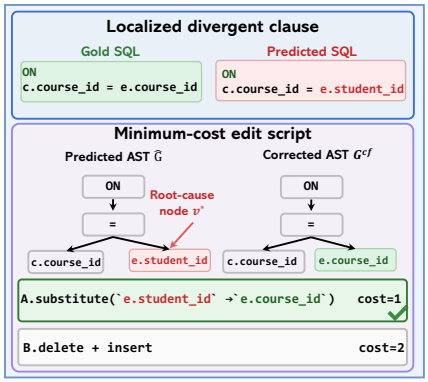

Figure 3: A counterfactual intervention example.

We then identify the node-level source of this mismatch. Let $\hat{G} = \mathrm{Parse}(\hat{y})$ and $G^{\mathrm{cf}} = \mathrm{Parse}(y^{\mathrm{cf}})$ denote the ASTs of the predicted and counterfactually corrected queries, and let $\Xi_{k^\star}(\hat{G}, G^{\mathrm{cf}})$ be the set of valid edit scripts that transform the divergent predicted clause into its corrected counterpart. We select the minimum-cost script and define the root-cause node $v^\star$ as the anchor of its first edit:

$$\xi^\star = \arg\min_{\xi \in \Xi_{k^\star}(\hat{G}, G^{\mathrm{cf}})} \sum_{m=1}^{|\xi|} c(\xi_m), \quad v^\star = \mathrm{Anchor}(\xi_1^\star), \tag{13}$$

where $c(\xi_m)$ is the cost of the m-th node or edge edit, and $\mathrm{Anchor}(\xi_1^\star)$ returns the touched node in $\hat{G}$, or its parent attachment node for an insertion. Thus, $\xi^\star$ is the smallest local transformation from the predicted clause to its counterfactual correction.

Topology-Aware Node-Level Reward. Once the root-cause node $v^\star$ is identified or induced, we assign a continuous penalty to each AST node $\hat{v}_i \in V$. Equation (13) localizes the clause-level mismatch anchor, while the distance below measures shortest-path distance to that anchor within $\hat{G}$:

$$R_{\text{node}}(\hat{v}_i; v^\star) = -\alpha\, M(\hat{v}_i, v^\star) \exp\left(-\frac{d_{\hat{G}}(\hat{v}_i, v^\star)^2}{2\sigma_d^2}\right), \tag{14}$$

where $M(\hat{v}_i, v^\star) \in \{0, 1\}$ is a causal mask, and the graph distance $d_{\hat{G}}(\hat{v}_i, v^\star) = |\eta_{\hat{G}}(\hat{v}_i, v^\star)|$ is the shortest-path distance on $\hat{G}$, where $\eta_{\hat{G}}(\hat{v}_i, v^\star)$ is the unique simple AST path connecting $\hat{v}_i$ and $v^\star$, and $|\eta_{\hat{G}}|$ is the number of edges on that path. We set $M(\hat{v}_i, v^\star) = 1$ only when $\hat{v}_i$ is $v^\star$ or an ancestor of $v^\star$ within the enclosing clause-level AST region, and 0 otherwise. Here $\alpha > 0$ is the maximum penalty magnitude, and σ_d controls the distance-decay bandwidth. The root-cause node itself receives the strongest penalty $-\alpha$, while structurally upstream nodes on the same dependency path receive Gaussian-decayed negative reward. Unrelated branches are masked out and receive zero reward.

Clause-Level Aggregation and Supervision Construction. To obtain supervision at the clause-unit level, we aggregate node-level penalties within each dispreferred unit:

$$R^-_{\text{clause}}(u_k^{\kappa_k}; v^\star) = \min_{\hat{v}_i \in \phi(u_k^{\kappa_k})} R_{\text{node}}(\hat{v}_i; v^\star). \tag{15}$$

The minimum operator propagates the strongest node-level penalty inside a clause unit to the clause level, ensuring that a clause containing the decisive error inherits a strong non-positive label.

For each supervised clause pair (u_k^+, u_k^-), where u_k^+ is the preferred clause unit and u_k^- is the dispreferred clause unit, we use the preferred clause as a zero-penalty reference and define the margin by the dispreferred clause penalty:

$$\delta_k = -R^-_{\text{clause}}(u_k^-; v^\star), \tag{16}$$

Since $R^-_{\text{clause}}(u_k^-; v^\star) \le 0$ by construction, we have $\delta_k \ge 0$, with larger values indicating a more decisive structural error in the dispreferred clause.

Based on these clause-level rewards, we construct local supervision tuples under a shared input–prefix context. For successful trajectories, the original clause unit is preferred over its injected counterpart; for failed trajectories, the counterfactually corrected clause is preferred over the original failed clause. We therefore define

$$\mathcal{D}^+ = \left\{ (x, p_k, (u_k^{\kappa_k})^+, (u_k^{\kappa_k})^-, \kappa_k, \delta_k) \;\middle|\; \tau \in \mathfrak{T}^+ \right\}, \tag{17}$$

and

$$\mathcal{D}^- = \left\{ (x, p_{k^\star}, (\tilde{u}_{k^\star}^{\kappa_{k^\star}})^+, (u_{k^\star}^{\kappa_{k^\star}})^-, \kappa_{k^\star}, \delta_{k^\star}) \;\middle|\; \tau \in \mathfrak{T}^- \right\}. \tag{18}$$

The final annotated dataset is $\mathcal{D} = \mathcal{D}^+ \cup \mathcal{D}^-$, where each tuple carries both a discrete preference label and a continuous margin δ_k reflecting the structural severity of the clause-level error.

3.3 Clause-Level Process Reward Model

Using the clause-level PRM in Equation (9), we train Clause-PRM on $\mathcal{D}$. For each tuple $(x, p_k, u_k^+, u_k^-, \kappa_k, \delta_k) \in \mathcal{D}$, we optimize the margin-weighted preference objective

$$\mathcal{L}_{\text{PRM}} = -\frac{1}{|\mathcal{D}|} \sum_{(x, p_k, u_k^+, u_k^-, \kappa_k, \delta_k) \in \mathcal{D}} \left(1 + \frac{\delta_k}{\alpha}\right) \log \sigma\big(f_\psi(x, p_k, u_k^+) - f_\psi(x, p_k, u_k^-)\big), \tag{19}$$

where $\sigma(\cdot)$ is the sigmoid function and α is the maximum penalty magnitude in Equation (14). Thus, clause pairs with larger structural margins contribute stronger preference supervision.

For policy optimization, the old policy samples a group of structured responses $\{z^{(g)}\}_{g=1}^B$ for the same input x. Each response yields a SQL query $\hat{y}^{(g)} = \text{ExtractSQL}(z^{(g)})$ and a clause sequence $(\hat{u}_{g,1}^{\kappa_{g,1}}, \ldots, \hat{u}_{g,K_g}^{\kappa_{g,K_g}})$. At the k-th clause boundary, we assign

$$r_{g,k} = I_{\text{format}}(z^{(g)}) \left[\lambda f_\psi(x, \hat{p}_{g,k}, \hat{u}_{g,k}^{\kappa_{g,k}}) + (1 - \lambda) \mathbb{1}[k = K_g] r_{\text{exec}}^{(g)}\right], \tag{20}$$

where $r_{\text{exec}}^{(g)} = \mathbb{1}[\text{Exec}(\hat{y}^{(g)}) = \text{Exec}(y)]$. Thus, the execution reward is added only once at the terminal clause boundary, while intermediate boundaries receive learned clause-level process rewards. The components are:

- **Format reward [18]:** $I_{\text{format}}(z^{(g)}) = \mathbb{1}[\text{ValidFormat}(z^{(g)})]$ checks the response structure rather than SQL syntax; it equals 1 only when $z^{(g)}$ follows the required `<think>` $\cdots$ `</think>` and `<answer>` $\cdots$ `</answer>` format.
- **Clause reward:** $f_\psi(x, \hat{p}_{g,k}, \hat{u}_{g,k}^{\kappa_{g,k}})$ scores the current clause under the predicted prefix.
- **Execution reward:** $r_{\text{exec}}^{(g)}$ checks whether the predicted and gold SQL produce the same execution result.

Instead of collapsing all clause rewards into a single trajectory-level advantage, we compute a clause-level return-to-go $G_{g,k} = \sum_{j=k}^{K_g} \gamma_c^{j-k} r_{g,j}$ where γ_c is a clause-level discount factor. We use GRPO-style normalization at each clause position. Let $\mathcal{B}_k = \{g : K_g \geq k\}$ be the valid responses that contain a k-th clause. The clause-level group advantage is

$$A_{g,k} = \frac{G_{g,k} - \text{mean}(\{G_{h,k} : h \in \mathcal{B}_k\})}{\text{std}(\{G_{h,k} : h \in \mathcal{B}_k\}) + \epsilon_{\text{std}}}, \tag{21}$$

for $g \in \mathcal{B}_k$. Let $c_g(t) \in \{1, \ldots, K_g\}$ map token position t to the clause span whose boundary reward supervises that token. The clipped GRPO objective becomes

$$\mathcal{L}_{\text{GRPO}}(\theta) = -\mathbb{E}_{x,\{z^{(g)}\}} \left[\frac{1}{B} \sum_{g=1}^{B} \frac{1}{|z^{(g)}|} \sum_{t=1}^{|z^{(g)}|} \min\big(\rho_{g,t} A_{g,c_g(t)},\ \bar{\rho}_{g,t} A_{g,c_g(t)}\big) \right], \tag{22}$$

where $\rho_{g,t} = \pi_\theta(z_t^{(g)} \mid x, z_{<t}^{(g)}) / \pi_{\theta_{\text{old}}}(z_t^{(g)} \mid x, z_{<t}^{(g)})$ and $\bar{\rho}_{g,t} = \text{clip}(\rho_{g,t}, 1 - \epsilon_{\text{clip}}, 1 + \epsilon_{\text{clip}})$. This clause-level return-to-go preserves temporal credit assignment: tokens in a clause are optimized with the advantage of the remaining trajectory after that clause boundary, rather than a single trajectory-level advantage shared by all tokens.

4 Experiments

4.1 Experiment Setup

Data and Models. We evaluate `CAPER` on BIRD [19] and Spider [20]. We first run `Qwen3.5-9B` on the training splits and then apply counterfactual intervention and fault injection to build over $90{,}000$ clause-annotated preference tuples from BIRD-train, Spider-dev and SYNSQL-5k-train. We fine-tune `Qwen3.5-9B` on this data to obtain `CAPER-9B` as our clause-level PRM. For policy optimization, we start from a `Qwen3.5-9B-SFT` policy model (fine-tuned on BIRD-train and SYNSQL-5k [26]) and compare GRPO under sparse execution reward, token-level reward, and the clause-level reward from `CAPER-9B`.

Evaluation Protocol. For end-to-end Text-to-SQL, we report execution accuracy (EX) [19] on the development set of BIRD, and both development and test sets of Spider under greedy decoding and Majority Vote@8 [18]. For failure localization, we hold out 10% of the annotated data and evaluate only failed trajectories. Each method ranks the clause units in a trajectory, and we report top-1 localization accuracy (Acc_{loc}), Hit@3, and mean reciprocal rank (MRR).

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 sparse, token, or clause-level rewards. For failure localization, we compare against heuristic selectors, prompted self-debugging, an execution-only reward model, and ablations of our annotation pipeline. For DeepEye-SQL, we use GPT-5.4 as the base model due to limited computational resources. Full baseline lists, hyperparameters, inference details, and hardware setup are provided in Appendix D.

4.2 Main Results

Our main question is the impact of clause-level supervision on RL itself: *does a more semantically aligned reward improve policy optimization over sparse execution feedback or token-level shaping?* Since Text-to-SQL is ultimately judged by end-to-end execution accuracy, we first evaluate Clause-PRM by its effect on the final GRPO-trained policy on BIRD and Spider.

Table 1: Main execution accuracy (EX, %) on BIRD and Spider under greedy decoding (P@1) and Majority Vote@8 (MV@8), where the backbone-matched block compares `Qwen3.5-9B-SFT` with GRPO variants trained under sparse, token-level, and clause-level rewards; the best and second-best results are highlighted in bold and underlined, respectively. GPT-5.4 is evaluated zero-shot and serves as a contextual reference across different training settings.

Model	BIRD		Spider Dev		Spider Test		Avg.	
	P@1	MV@8	P@1	MV@8	P@1	MV@8	P@1	MV@8
Closed-source LLMs								
`Gemini-2.5-Pro` [30]	<u>62.32</u>	63.77	**82.78**	**84.13**	78.04	<u>83.65</u>	<u>74.38</u>	<u>77.18</u>
`Claude Sonnet 4` [31]	55.34	66.10	76.40	78.52	75.04	79.97	68.93	74.86
`GPT-5.4` [32]	59.58	60.36	77.85	83.26	78.24	81.73	71.89	75.12
`Claude Sonnet 4.6` [33]	54.62	67.13	76.59	79.01	74.84	78.59	68.68	74.91
`DeepEye-SQL` [34]	/	<u>68.56</u>	/	73.89	/	80.44	/	74.30
Open-source LLMs (7B–9B)								
`OmniSQL-7B` [18]	36.83	56.38	76.78	79.11	77.55	79.69	63.72	71.73
`SQL-R1-7B` [26]	36.57	56.58	78.52	79.21	80.34	80.48	65.14	72.09
`XiyanSQL-7B` [35]	30.64	35.00	55.03	79.49	53.88	76.99	46.52	63.83
`AlphaSQL` [36]	/	58.51	/	73.59	/	77.96	/	70.02
`Qwen3.5-9B` [37]	51.90	58.91	75.44	76.59	78.43	79.92	68.59	71.81
Open-source LLMs ($\geq$14B)								
`OmniSQL-14B` [18]	45.96	65.71	76.98	79.20	78.34	79.83	67.09	74.91
`XiyanSQL-14B` [35]	37.15	42.11	58.46	80.27	73.87	78.44	56.49	66.94
Backbone-Matched Policies								
`Qwen3.5-9B-SFT` [37]	57.56	61.34	78.23	79.69	78.71	80.48	71.50	73.84
`GRPO-Sparse`	60.21	62.19	80.15	81.22	<u>81.56</u>	82.09	73.97	75.17
`GRPO-Clause`	**63.56**	**69.61**	<u>81.63</u>	<u>83.55</u>	**82.63**	**83.87**	**75.94**	**79.01**
`GRPO-Token`	57.36	60.49	78.59	79.11	79.17	79.62	71.71	73.07

Within the backbone-matched setting in Table 1, `GRPO-Clause` achieves the strongest results under both greedy decoding and Majority Vote@8. On BIRD, it increases MV@8 execution accuracy from 62.19% with `GRPO-Sparse` to 69.61%, a gain of 7.42 percentage points (11.9% relative). This controlled comparison supports the benefit of clause-aligned supervision for policy optimization. The gains are larger under majority voting, while token-level rewards fail to match the clause-level variant. Zero-shot `GPT-5.4` provides additional context on overall performance across training settings: `GRPO-Clause` is fine-tuned on BIRD-train, whereas `GPT-5.4` is evaluated zero-shot.

4.3 Reward Injection and Sensitivity

We run a controlled ablation using 128 examples from BIRD-train to examine reward injection and sensitivity to λ. The trajectory baseline uses the same clause labels and rewards but collapses them into one trajectory-level advantage, isolating the per-clause credit assignment in Equations (21) and (22).

Table 2: Reward injection and λ sensitivity with 128 BIRD-train examples. Execution accuracy (EX) is reported in percentages on the development sets.

Credit assignment	λ	BIRD Dev EX	Spider Dev EX
Per-clause	0.25	60.43	76.50
Per-clause	0.50	**61.02**	**78.43**
Per-clause	0.75	59.84	77.66
Trajectory	0.50	59.65	77.08

As shown in Table 2, $\lambda = 0.50$ yields the highest execution accuracy on both datasets among the tested settings. At this value, per-clause injection improves over the trajectory baseline by 1.37 EX points on BIRD Dev and 1.35 points on Spider Dev. On Spider Test, the corresponding gains are 1.82 points under greedy decoding and 0.47 points under MV@8.

4.4 Failure Localization

We next evaluate whether Clause-PRM learns error attribution rather than only trajectory-level preference by asking each method to rank faulty clauses in held-out failed trajectories. As shown in Table 3, `CAPER-9B` achieves the best overall results across all three metrics. The strong SELF-DEBUG baseline suggests that prompted LLMs can often identify suspicious regions, but Clause-PRM provides more reliable clause rankings through explicit clause-level supervision. The ablations further clarify where this gain comes from. Removing topology causes a large drop in top-1 accuracy while keeping Hit@3 relatively high, which suggests that the model can still place the faulty clause near the top of the list but struggles to rank it precisely without structural propagation. Removing fault injection or counterfactual supervision is even more damaging, pushing performance much closer to heuristic and execution-only baselines. Together, these trends indicate that strong failure localization does not come from generic reward modeling alone; it depends on combining clause-level supervision with topology-aware propagation and counterfactual error construction.

Table 3: Failure localization on held-out failures. All metrics are reported in percentages.

Method	$\mathrm{Acc}_{\mathrm{loc}}(\%)\uparrow$	Hit@3 (%) ↑	MRR (%) ↑
RANDOM-CLAUSE	25.90	76.34	52.51
LAST-CLAUSE	9.82	53.49	38.60
`Qwen3.5-9B` (SELF-DEBUG)	<u>82.78</u>	<u>93.56</u>	<u>86.35</u>
`Exec-RM-9B`	25.93	71.50	50.82
`w/o Topology`	45.50	89.96	67.34
`w/o Fault Injection`	22.20	80.17	51.24
`w/o Counterfactual`	26.02	83.06	54.31
`CAPER-9B`	**84.53**	**96.58**	**90.60**

4.5 Clause-PRM as a Candidate Verifier

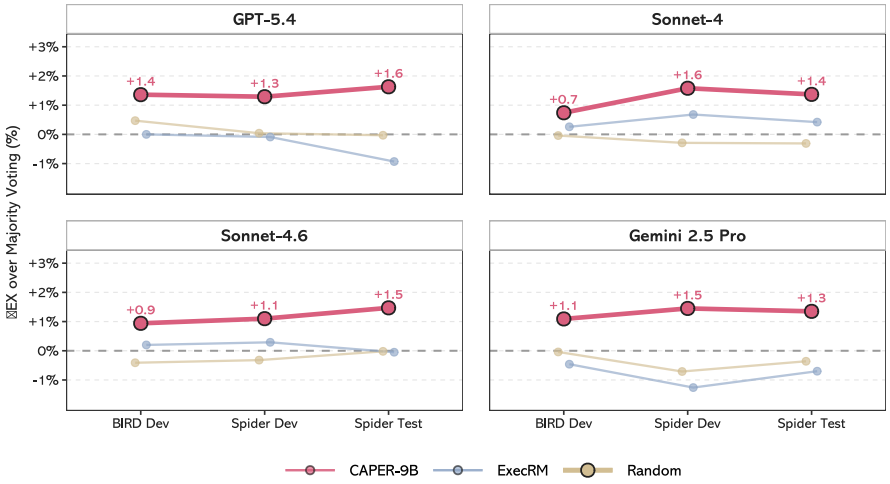

Figure 4: Candidate verification gains over Majority Vote@8, where each selector ranks the same eight sampled SQL candidates and ΔEX measures the absolute execution-accuracy change over majority voting.

Beyond using Clause-PRM as a reward model for policy optimization, we further evaluate whether it can serve as a plug-and-play verifier for candidates generated by closed-source LLMs. This setting is motivated by black-box Text-to-SQL deployment, where practitioners often cannot update the generator but can sample multiple candidates and rerank them with an external verifier. For each question, all non-greedy selectors operate on the same eight sampled SQL candidates, so the comparison isolates the candidate selection rule rather than generation quality. As shown in Figure 4,

CAPER-9B consistently improves over Majority Vote@8 across four closed-source generators and three evaluation splits, with gains ranging from $+0.7$ to $+1.6$ EX points. In contrast, Exec-RM and random selection are less stable and often stay near or below majority voting. This suggests that majority voting can be limited by execution-result frequency, while terminal-label reward models provide only coarse candidate-level signals; clause-level process supervision offers a more transferable semantic quality signal for identifying candidates with reliable intermediate SQL decisions.

4.6 Efficiency Analysis

We measure policy generation and verifier reranking separately on the same stratified set of 100 BIRD Dev questions. Generation latency is measured under greedy decoding, while reranking latency measures the time required to score the same eight frozen SQL candidates per question, excluding candidate generation.

Table 4: Inference latency on 100 BIRD Dev questions. Greedy decoding and eight-candidate reranking are timed separately; all values are in seconds per question.

Stage	Method	Latency (s/question)
Greedy decoding	GRPO-Sparse	1.88
	GRPO-Token	3.66
	GRPO-Clause	2.80
Reranking eight candidates	Exec-RM	0.835 ± 0.004
	Token-PRM	4.358 ± 0.014
	Clause-PRM	4.072 ± 1.018

As shown in Table 4, GRPO-Clause takes 2.80 seconds per question, reducing generation latency by 23.5% relative to GRPO-Token, while remaining slower than GRPO-Sparse. For reranking, Clause-PRM has a similar average latency to Token-PRM and a higher cost than Exec-RM. These measurements separate policy generation from the additional cost of optional candidate verification.

5 Related Work

Text-to-SQL. Text-to-SQL has progressed from early semantic parsing systems such as Seq2SQL [25] to cross-domain benchmarks like Spider [20], with later work improving structural modeling through intermediate representations, relation-aware schema encoding, and constrained decoding [38–40]. Recent studies further extend evaluation to more realistic settings, including BIRD and Spider 2.0, and increasingly rely on LLM-based decomposition and self-correction [19, 21, 41, 42, 24]. Despite this progress, surveys consistently highlight persistent brittleness under schema shift and spurious language-SQL correlations [43, 1]. Our work is most related to this robustness direction, focusing on counterfactual supervision rather than new parser architectures or prompting pipelines.

Credit Assignment. Credit assignment remains difficult when rewards are sparse or delayed. Prior work addresses this problem through counterfactual attribution in logged feedback and RL, including Counterfactual Risk Minimization and its variants, COMA, and counterfactual off-policy evaluation [44–48]. Process supervision similarly shows that intermediate feedback can be more informative than pure outcome signals for long-horizon reasoning [49, 29]. In Text-to-SQL, recent work explores execution rewards, graph-based shaping, stepwise PRM guidance, and clause-wise critics [25–28, 50, 51]. Our method follows this credit-assignment perspective, but derives semantically grounded clause-level supervision from counterfactual AST perturbations for reward modeling and policy optimization.

6 Conclusion

CAPER derives clause-aligned process supervision for Text-to-SQL by attributing query-level execution outcomes to SQL decisions. The resulting Clause-PRM improves failure localization, candidate verification, and GRPO-based policy optimization on BIRD and Spider, suggesting that clause-aligned supervision is a scalable direction for structured generation.

Acknowledgments and Disclosure of Funding

Funding. Chenhao Ma was partially supported by NSFC under Grant 62672430, Basic and Applied Basic Research Fund in Guangdong Province under Grant 2025A1515010439, Ant Group through CCF-Ant Research Fund, 1+1+1 CUHK-CUHK(SZ)-GDSTC Joint Collaboration Fund - Grant 2025A0505000079, and the Guangdong Provincial Key Laboratory of Big Data Computing, The Chinese University of Hong Kong, Shenzhen. Jiasheng Shi was supported by the Youth Science Fund Project (Youth Science Fund (Category C)) of the National Natural Science Foundation of China (grant number 12501403). Xiaolin Han was supported by NSFC under Grants 62672417 and 62302397, Young Talent Fund of Xi'an Association for Science and Technology (0959202613046), the China Postdoctoral Science Foundation under Grant Number 2025M774364, and Shaanxi Postdoctoral Research Project (2025BSHSDZZ106). Tsz Nam Chan was supported by the Natural Science Foundation of China under grants 23IAA00610 and 62572326.

Competing interests. The authors declare no competing interests.

References

[1] Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Nan Tang, and Yuyu Luo. A survey of text-to-sql in the era of llms: Where are we, and where are we going? *IEEE Transactions on Knowledge and Data Engineering*, 37(10):5735–5754, 2025. doi: 10.1109/TKDE.2025.3592032. URL `https://dblp.org/rec/journals/tkde/LiuSLMJZFLTL25`.

[2] Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. Next-Generation Database Interfaces: A Survey of LLM-Based Text-to-SQL . *IEEE Transactions on Knowledge & Data Engineering*, 37(12):7328–7345, December 2025. ISSN 1558-2191. doi: 10.1109/TKDE.2025.3609486. URL `https://doi.ieeecomputersociety.org/10.1109/TKDE.2025.3609486`.

[3] 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(2): 121–132, 2020.

[4] 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.

[5] 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*, pages 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/`.

[6] 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)*, 2018.

[7] 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 Pushkar Mishra, Smaranda Muresan, and Tao Yu, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 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/`.

[8] 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)*, pages 118–128, 2025.

[9] 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 Asli Celikyilmaz and Tsung-Hsien Wen, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 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/.

[10] 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*, 2025. URL https://openreview.net/forum?id=yRxXTdElLv.

[11] 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.

[12] Yihan Wang, Peiyu Liu, and Xin Yang. Linkalign: Scalable schema linking for real-world large-scale multi-database text-to-sql. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 977–991, 2025.

[13] Yujian Gan, Xinyun Chen, and Matthew Purver. Re-appraising the schema linking for text-to-SQL. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 835–852, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.53. URL https://aclanthology.org/2023.findings-acl.53/.

[14] Mayank Kothyari, Dhruva Dhingra, Sunita Sarawagi, and Soumen Chakrabarti. CRUSH4SQL: Collective retrieval using schema hallucination for Text2SQL. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 14054–14066, Singapore, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.868. URL https://aclanthology.org/2023.emnlp-main.868/.

[15] Zhenhe Wu, Zhongqiu Li, Jie Zhang, Zhongjiang He, Jian Yang, Yu Zhao, Ruiyu Fang, Bing Wang, Hongyan Xie, Shuangyong Song, et al. Ucs-sql: uniting content and structure for enhanced semantic bridging in text-to-sql. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 8156–8168, 2025.

[16] Wenbo Xu, Haifeng Zhu, Liang Yan, Chuanyi Liu, Peiyi Han, Shaoming Duan, and Jeff Z Pan. Ts-sql: Test-driven self-refinement for text-to-sql. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 2864–2889, 2025.

[17] Ge Qu, Jinyang Li, Bowen Qin, Xiaolong Li, Nan Huo, Chenhao Ma, and Reynold Cheng. Share: An slm-based hierarchical action correction assistant for text-to-sql. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11268–11292, 2025.

[18] Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, et al. Omnisql: Synthesizing high-quality text-to-sql data at scale. *Proceedings of the VLDB Endowment*, 18(11):4695–4709, 2025.

[19] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chen-Chuan Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. In *Advances in Neural Information Processing Systems 36*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/83fc8fab1710363050bbd1d4b8cc0021-Abstract-Datasets_and_Benchmarks.html.

[20] 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*, pages 3911–3921, Brussels, Belgium, 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1425. URL https://aclanthology.org/D18-1425/.

[21] 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.

[22] Tomer Wolfson, Daniel Deutch, and Jonathan Berant. Weakly supervised text-to-SQL parsing through question decomposition. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 2528–2542, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-naacl.193. URL https://aclanthology.org/2022.findings-naacl.193/.

[23] Jipeng Zhang, Haolin Yang, Kehao Miao, Ruiyuan Zhang, Renjie Pi, Jiahui Gao, and Xiaofang Zhou. ExeSQL: Self-taught text-to-SQL models with execution-driven bootstrapping for SQL dialects. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 24305–24326, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.1320. URL https://aclanthology.org/2025.findings-emnlp.1320/.

[24] Mingqian He, Yongliang Shen, Wenqi Zhang, Qiuying Peng, Jun Wang, and Weiming Lu. Star-sql: Self-taught reasoner for text-to-sql. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 24365–24375, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.1187. URL https://aclanthology.org/2025.acl-long.1187/.

[25] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103, 2017. doi: 10.48550/arXiv.1709.00103. URL https://arxiv.org/abs/1709.00103.

[26] 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.

[27] Zhewei Yao, Guoheng Sun, Lukasz Borchmann, Gaurav Nuti, Zheyu Shen, Minghang Deng, Bohan Zhai, Hao Zhang, Ang Li, and Yuxiong He. Arctic-text2sql-r1: Simple rewards, strong reasoning in text-to-sql, 2025. URL https://arxiv.org/abs/2505.20315.

[28] Han Weng, Puzhen Wu, Cui Longjie, Yi Zhan, Boyi Liu, Yuanfeng Song, Dun Zeng, Yingxiang Yang, Qianru Zhang, Dong Huang, Xiaoming Yin, Yang Sun, and Xing Chen. Graph-reward-SQL: Execution-free reinforcement learning for text-to-SQL via graph matching and stepwise reward. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 12917–12943, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.694. URL https://aclanthology.org/2025.findings-emnlp.694/.

[29] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun, editors, *International Conference on Learning Representations*, volume 2024, pages 39578–39601, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/aca97732e30bcf1303bc22ac3924fd16-Paper-Conference.pdf.

[30] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the

frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

[31] Anthropic. Claude Sonnet 4, 2025. URL https://www.anthropic.com/news/claude-4.

[32] OpenAI. Introducing GPT-5.4, March 2026. URL https://openai.com/index/introducing-gpt-5-4/.

[33] Anthropic. Introducing Claude Sonnet 4.6, February 2026. URL https://www.anthropic.com/research/claude-sonnet-4-6.

[34] Boyan Li, Chong Chen, Zhujun Xue, Yinan Mei, and Yuyu Luo. Deepeye-sql: A software-engineering-inspired text-to-sql framework. *Proc. ACM Manag. Data*, 4(3), May 2026. doi: 10.1145/3802035. URL https://doi.org/10.1145/3802035.

[35] 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 & Data Engineering*, 38(04):2474–2487, April 2026. ISSN 1558-2191. URL https://doi.ieeecomputersociety.org/10.1109/TKDE.2026.3657851.

[36] Boyan Li, Jiayi Zhang, Ju Fan, Yanwei Xu, Chong Chen, Nan Tang, and Yuyu Luo. Alpha-SQL: Zero-shot text-to-SQL using Monte Carlo tree search. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 36810–36830. PMLR, 2025. URL https://proceedings.mlr.press/v267/li25dt.html.

[37] QwenTeam. Qwen3.5: Towards native multimodal agents, 2026. URL https://qwen.ai/blog?id=qwen3.5.

[38] Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. Towards complex text-to-sql in cross-domain database with intermediate representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4524–4535, Florence, Italy, 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1444. URL https://aclanthology.org/P19-1444/.

[39] 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*, pages 7567–7578, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.677. URL https://aclanthology.org/2020.acl-main.677/.

[40] 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*, pages 9895–9901, Online and Punta Cana, Dominican Republic, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.779. URL https://aclanthology.org/2021.emnlp-main.779/.

[41] 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, pages 36339–36348, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/72223cc66f63ca1aa59edaec1b3670e6-Paper-Conference.pdf.

[42] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-sql empowered by large language models: A benchmark evaluation. *Proceedings of the VLDB Endowment*, 17(5):1132–1145, 2024. doi: 10.14778/3641204.3641221. URL https://dblp.org/rec/journals/pvldb/GaoWLSQDZ24.

[43] Naihao Deng, Yulong Chen, and Yue Zhang. Recent advances in text-to-sql: A survey of what we have and what we expect. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 2166–2187, Gyeongju, Republic of Korea, 2022. International Committee on Computational Linguistics. URL https://aclanthology.org/2022.coling-1.190/.

[44] Adith Swaminathan and Thorsten Joachims. Counterfactual risk minimization: Learning from logged bandit feedback. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 814–823, Lille, France, 2015. PMLR. URL https://proceedings.mlr.press/v37/swaminathan15.html.

[45] Ben London and Ted Sandler. Bayesian counterfactual risk minimization. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4125–4133. PMLR, 2019. URL https://proceedings.mlr.press/v97/london19a.html.

[46] Houssam Zenati, Eustache Diemert, Matthieu Martin, Julien Mairal, and Pierre Gaillard. Sequential counterfactual risk minimization. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 40681–40706. PMLR, 2023. URL https://proceedings.mlr.press/v202/zenati23a.html.

[47] Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, pages 2974–2982. AAAI Press, 2018. doi: 10.1609/AAAI.V32I1.11794. URL https://dblp.org/rec/conf/aaai/FoersterFANW18.html.

[48] Michael Oberst and David Sontag. Counterfactual off-policy evaluation with gumbel-max structural causal models. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4881–4890. PMLR, 2019. URL https://proceedings.mlr.press/v97/oberst19a.html.

[49] Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback. 2022. doi: 10.48550/ARXIV.2211.14275. URL https://arxiv.org/abs/2211.14275.

[50] Yuxin Zhang, Meihao Fan, Ju Fan, Mingyang Yi, Yuyu Luo, Jian Tan, and Guoliang Li. Reward-SQL: Boosting text-to-SQL via stepwise reasoning and process-supervised rewards, 2025. URL https://arxiv.org/abs/2505.04671.

[51] Jikai Chen, Leilei Gan, Ziyu Zhao, Zechuan Wang, Dong Wang, and Chenyi Zhuang. SQLCritic: Correcting text-to-SQL generation via clause-wise critic, 2025. URL https://arxiv.org/abs/2503.07996.

A Limitations

While our proposed method demonstrates significant improvements in Text-to-SQL reinforcement learning, there are several limitations to consider. First, the clause-level reward model is trained on a specific set of datasets and may not generalize well to other domains or SQL dialects without further fine-tuning. Second, we focus on SQL Generation rather than Chain of Thought (CoT) reasoning, and our method may not directly apply to other structured generation tasks that require different forms of intermediate supervision. Third, `CAPER` currently decomposes SQL only at the top-level clause granularity, which can still be coarse when the decisive error lies inside a clause, such as one predicate in a conjunctive `WHERE` condition or an internal subquery; although the same annotation procedure can in principle be applied recursively to obtain hierarchical rewards at the clause, sub-clause, and token levels, we restrict this work to the first level to keep the experimental cost manageable. Finally, the automated annotation pipeline introduces additional computational and engineering overhead because it requires SFT rollouts, AST parsing, counterfactual repair or fault injection, and repeated execution checks; this cost is higher than using sparse terminal execution rewards alone, although the resulting annotations can be reused for PRM training.

B Theoretical Analysis of Reward Propagation

This section gives deterministic bounds for the topology-aware reward in Equation (14) and the clause-level aggregation in Equation (15). These results do not require any distributional assumption; they follow directly from the bounded mask, nonnegative AST distance, and the monotonicity of the Gaussian kernel.

Lemma 1 (Bounds of Gaussian node propagation)**.** *Let $\alpha > 0$, $\sigma_d > 0$, $M_i = M(\hat{v}_i, v^\star) \in \{0, 1\}$, and $d_i = d_{\hat{G}}(\hat{v}_i, v^\star) \geq 0$. The node-level reward*

$$R_{\text{node}}(\hat{v}_i; v^\star) = -\alpha M_i \exp\left(-\frac{d_i^2}{2\sigma_d^2}\right)$$

satisfies

$$-\alpha \leq R_{\text{node}}(\hat{v}_i; v^\star) \leq 0.$$

If $M_i = 0$, then $R_{\text{node}}(\hat{v}_i; v^\star) = 0$. If $M_i = 1$ and $d_i \leq D$, then

$$-\alpha \leq R_{\text{node}}(\hat{v}_i; v^\star) \leq -\alpha \exp\left(-\frac{D^2}{2\sigma_d^2}\right).$$

Moreover, for unmasked nodes, R_{node} is monotone nondecreasing in d_i, equivalently the error severity $-R_{\text{node}}$ is monotone nonincreasing in d_i.

Proof. Since $d_i \geq 0$ and $\sigma_d > 0$,

$$0 < \exp\left(-\frac{d_i^2}{2\sigma_d^2}\right) \leq 1.$$

Together with $M_i \in \{0, 1\}$, this gives

$$0 \leq M_i \exp\left(-\frac{d_i^2}{2\sigma_d^2}\right) \leq 1.$$

Multiplying by $-\alpha$ yields $-\alpha \leq R_{\text{node}}(\hat{v}_i; v^\star) \leq 0$. If $M_i = 0$, the reward is exactly zero. If $M_i = 1$ and $d_i \leq D$, the Gaussian term is decreasing in d_i, so

$$\exp\left(-\frac{D^2}{2\sigma_d^2}\right) \leq \exp\left(-\frac{d_i^2}{2\sigma_d^2}\right) \leq 1.$$

Multiplying by $-\alpha$ gives the stated local bound. Finally, when $M_i = 1$,

$$\frac{\partial R_{\text{node}}}{\partial d_i} = \frac{\alpha d_i}{\sigma_d^2} \exp\left(-\frac{d_i^2}{2\sigma_d^2}\right) \geq 0,$$

so the negative reward becomes weaker as the node moves farther from the anchor, while the corresponding severity $-R_{\text{node}}$ decreases with distance. □

Lemma 2 (Bounds of minimum clause aggregation)**.** *For a nonempty clause unit u, define the active node set*

$$\mathcal{A}(u; v^\star) = \{\hat{v}_i \in \phi(u) : M(\hat{v}_i, v^\star) = 1\}.$$

The clause-level reward in Equation (15) *satisfies*

$$-\alpha \le R^-_{\text{clause}}(u; v^\star) \le 0.$$

If $\mathcal{A}(u; v^\star) = \emptyset$, then $R^-_{\text{clause}}(u; v^\star) = 0$. Otherwise, let

$$d_{\min}(u) = \min_{\hat{v}_i \in \mathcal{A}(u;v^\star)} d_{\hat{G}}(\hat{v}_i, v^\star), \qquad D_u = \max_{\hat{v}_i \in \mathcal{A}(u;v^\star)} d_{\hat{G}}(\hat{v}_i, v^\star).$$

Then the minimum aggregation has the exact form

$$R^-_{\text{clause}}(u; v^\star) = -\alpha \exp\left(-\frac{d_{\min}(u)^2}{2\sigma_d^2}\right),$$

and therefore

$$-\alpha \le R^-_{\text{clause}}(u; v^\star) \le -\alpha \exp\left(-\frac{D_u^2}{2\sigma_d^2}\right) \le 0.$$

Proof. By Lemma 1, every node reward lies in $[-\alpha, 0]$. The minimum over the nonempty set $\phi(u)$ must therefore also lie in $[-\alpha, 0]$, proving the global bound.

If $\mathcal{A}(u; v^\star) = \emptyset$, then every node in the clause has mask value zero, hence every node reward is zero and the minimum is zero. If $\mathcal{A}(u; v^\star) \neq \emptyset$, inactive nodes have reward zero and cannot determine the minimum because active nodes have nonpositive rewards. Thus

$$R^-_{\text{clause}}(u; v^\star) = \min_{\hat{v}_i \in \mathcal{A}(u;v^\star)} \left[-\alpha \exp\left(-\frac{d_{\hat{G}}(\hat{v}_i, v^\star)^2}{2\sigma_d^2}\right)\right].$$

The Gaussian term is decreasing in distance, and the leading negative sign means the most negative reward is achieved by the active node closest to $v^\star$. Therefore the minimum equals

$$-\alpha \exp\left(-\frac{d_{\min}(u)^2}{2\sigma_d^2}\right).$$

Since $d_{\min}(u) \le D_u$, the same monotonicity gives the stated upper bound. Equivalently, if $s_i = -R_{\text{node}}(\hat{v}_i; v^\star) \ge 0$ denotes node-level error severity, then

$$-R^-_{\text{clause}}(u; v^\star) = \max_{\hat{v}_i \in \phi(u)} s_i,$$

so minimum reward aggregation is exactly maximum severity aggregation over nodes in the clause. This preserves the decisive local error rather than diluting it by clause length. □

C Algorithm Details

The details of the annotation pipeline are shown in Algorithm 1.

Complexity Analysis. Let $M = |\mathfrak{T}^+| + |\mathfrak{T}^-|$ be the number of trajectories to annotate, and let N upper-bound the SQL length, number of clause units, AST size, and tokenized Clause-PRM input length for any trajectory. Clause splitting, AST traversal, fault injection, and clause-to-node reward aggregation are linear in N. The only super-linear algorithmic component is FIRSTDIVERGENCE, which performs clause-restricted edit-script computation; using a standard tree-edit dynamic program, this step costs $O(N^3)$ in the worst case. For database execution, let D upper-bound the number of rows in any table and let J be the maximum number of joined tables in a generated query. Without assuming a particular index or query plan, one execution has worst-case cost $O(D^J)$. Therefore, the worst-case offline annotation complexity is

$$O\big(M(N^3 + D^J)\big).$$

The storage complexity is $O(MN)$ when storing the retained clause preference tuples explicitly, or $O(M)$ if the tuples store references to the original SQL strings. During policy optimization, a rollout

Algorithm 1 Automatic Clause-Level Reward Annotation

Input: $\mathfrak{T}^-,\ \mathfrak{T}^+,\ \mu,\ \phi$
Output: $\mathcal{D}$

$\mathcal{D}^- \leftarrow \emptyset, \quad \mathcal{D}^+ \leftarrow \emptyset$
/* Part 1: Fault Injection on $\mathfrak{T}^+$ */
for $\tau \in \mathfrak{T}^+$ **do**
  $G \leftarrow \mathrm{Parse}(\hat{y}), \quad \mathcal{V}_{\mathrm{edit}} \leftarrow \mathrm{SampleEditable}(G)$
  for $v^\star \in \mathcal{V}_{\mathrm{edit}}$ **do**
    $G' \leftarrow \mu(G, v^\star)$
    if $\mathrm{Exec}(\mathrm{Decode}(G')) \neq \mathrm{Exec}(\mathrm{Decode}(G))$ **then**
      $k \leftarrow \mathrm{ClauseIndex}(v^\star; \phi), \quad u_k^+ \leftarrow [\mathrm{ClauseSplit}(\hat{y})]_k, \quad u_k^- \leftarrow [\mathrm{ClauseSplit}(\mathrm{Decode}(G'))]_k$
      $\delta_k \leftarrow -R^-_{\mathrm{clause}}(u_k^-; v^\star)$
      $\mathcal{D}^+ \leftarrow \mathcal{D}^+ \cup \{(x, p_k, u_k^+, u_k^-, \kappa_k, \delta_k)\}$
    end if
  end for
end for
/* Part 2: Counterfactual Intervention on $\mathfrak{T}^-$ */
for $\tau \in \mathfrak{T}^-$ **do**
  $x \leftarrow x(\tau),\ \hat{y} \leftarrow y(\tau),\ \tilde{y} \leftarrow y^{\mathrm{gt}}(\tau)$
  $\mathcal{K}_{\mathrm{div}} \leftarrow \{k :\ u_k \neq \tilde{u}_k\}$
  for $k^\star \in \mathcal{K}_{\mathrm{div}}$ **do**
    $y^{\mathrm{cf}} \leftarrow \mathrm{Repair}(\hat{y}, \tilde{y}, p_{k^\star})$
    if $\mathrm{Valid}(y^{\mathrm{cf}})$ **then**
      $v^\star \leftarrow \mathrm{FirstDivergence}(\mathrm{Parse}(\hat{y}), \mathrm{Parse}(y^{\mathrm{cf}}))$
      $\tilde{u}^+_{k^\star} \leftarrow [\mathrm{ClauseSplit}(y^{\mathrm{cf}})]_{k^\star}, \quad u^-_{k^\star} \leftarrow [\mathrm{ClauseSplit}(\hat{y})]_{k^\star}$
      $\delta_{k^\star} \leftarrow -R^-_{\mathrm{clause}}(u^-_{k^\star}; v^\star)$
      $\mathcal{D}^- \leftarrow \mathcal{D}^- \cup \{(x, p_{k^\star}, \tilde{u}^+_{k^\star}, u^-_{k^\star}, \kappa_{k^\star}, \delta_{k^\star})\}$
    end if
  end for
end for
return $\mathcal{D} \leftarrow \mathcal{D}^- \cup \mathcal{D}^+$

group of size B contains at most BN clause boundaries. A Transformer Clause-PRM forward pass over an $O(N)$-length input costs $O(N^2)$ when the model architecture is fixed, so reward scoring and terminal execution together add

$$O\big(B(N^3 + D^J)\big),$$

where the BN^3 term comes from BN clause-boundary PRM calls and the BD^J term comes from one terminal execution per rollout. In practice, clause-boundary PRM scores are batched, which improves wall-clock efficiency without changing the asymptotic bound.

D Experimental Setup Details

`CAPER-9B` **Data Split.** We train `CAPER-9B` on the final clause-annotated training split and use a held-out evaluation split for model selection and failure-localization evaluation. As shown in Table 5, the final split contains 82,081 training rows and 9,166 held-out evaluation rows, drawn from BIRD-train, SYNSQL-5k-train, and Spider-dev.

Table 5: `CAPER-9B` training and held-out evaluation data statistics.

Split	Total Rows	BIRD-Train	SYNSQL-5k-Train	Spider-Dev
Train	82,081	58,090	19,668	4,323
Eval	9,166	6,488	2,197	481

Training Configuration. We fine-tune `Qwen3.5-9B` on the clause-annotated dataset to obtain `CAPER-9B`. For policy optimization, we first train a supervised `Qwen3.5-9B-SFT` policy and then initialize GRPO from this checkpoint. We set the maximum penalty magnitude α in Equation (14) to 1.0, the Gaussian distance bandwidth σ_d in Equation (14) to 1.0, and the balance factor λ in Equation (20) to 0.5. RL training uses batch size 16, 8 rollouts, and learning rate 1×10^{-6}.

`GRPO-Token` **Baseline.** `GRPO-Token` uses the same initialization checkpoint, GRPO objective, rollout number, batch size, learning rate, and terminal execution reward as `GRPO-Clause`, but replaces the learned clause-level process reward with heuristic token-level dense shaping. Specifically, after extracting the SQL from the `<answer>` field, we canonicalize keyword casing and whitespace, tokenize the generated and gold SQL strings, and assign a token process reward $\mathbb{1}[\hat{s}_t = s_t^\star]$ at each SQL-token position t when both positions exist, and 0 otherwise. The terminal execution reward is still added only once at the final token. This baseline isolates the effect of making rewards dense at the token level without training an additional token-level PRM.

Inference and Metrics. For end-to-end evaluation, we report execution accuracy (EX) [19]. Given N evaluation examples with predicted SQL $\hat{y}_i$ and gold SQL y_i,

$$\mathrm{EX} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}[\mathrm{Exec}(\hat{y}_i) = \mathrm{Exec}(y_i)]\,. \tag{23}$$

Greedy decoding with temperature 0 corresponds to P@1. For Majority Vote@8, we sample candidates $\{\hat{y}_i^{(m)}\}_{m=1}^{8}$ with temperature 0.8, group them by execution result, and choose a candidate from the largest group:

$$r_i^{\mathrm{MV}} = \arg\max_r \sum_{m=1}^{8} \mathbb{1}\left[\mathrm{Exec}(\hat{y}_i^{(m)}) = r\right], \qquad \hat{y}_i^{\mathrm{MV@8}} \in \{\hat{y}_i^{(m)} : \mathrm{Exec}(\hat{y}_i^{(m)}) = r_i^{\mathrm{MV}}\}. \tag{24}$$

The reported MV@8 is EX computed with $\hat{y}_i = \hat{y}_i^{\mathrm{MV@8}}$. For candidate verification, we report the absolute gain over majority voting,

$$\Delta\mathrm{EX} = \mathrm{EX}(\hat{y}^{\mathrm{selector}}) - \mathrm{EX}(\hat{y}^{\mathrm{MV@8}}). \tag{25}$$

For failure localization, each method ranks the clause units in a failed trajectory. Let π_i denote the predicted ranking from most to least likely faulty for held-out failure i, $k_i^\star$ denote the annotated faulty clause, and $\rho_i = \mathrm{rank}_{\pi_i}(k_i^\star)$. For M held-out failures, we compute

$$\mathrm{Acc}_{\mathrm{loc}} = \frac{1}{M}\sum_{i=1}^{M}\mathbb{1}[\rho_i = 1], \qquad \mathrm{Hit@3} = \frac{1}{M}\sum_{i=1}^{M}\mathbb{1}[\rho_i \le 3], \qquad \mathrm{MRR} = \frac{1}{M}\sum_{i=1}^{M}\frac{1}{\rho_i}. \tag{26}$$

Full Baseline List. For end-to-end evaluation, we compare against representative Text-to-SQL models of varying scales, including `Gemini-2.5-Pro` [30], `Claude Sonnet 4` [31], `GPT-5.4` [32], `Claude Sonnet 4.6` [33], `DeepEye-SQL` [34], `OmniSQL-7B` [18], `SQL-R1-7B` [26], `XiyanSQL-7B` [35], `AlphaSQL` [36], `Qwen3.5-9B` [37], `OmniSQL-14B` [18], and `XiyanSQL-14B` [35]. Within the backbone-matched policy block, we additionally compare the supervised `Qwen3.5-9B-SFT` policy and GRPO variants trained under sparse, clause-level, and token-level rewards, as listed in Table 1. For failure localization, we compare `CAPER-9B` against four classes of baselines: (1) heuristic selectors, including RANDOM-CLAUSE and LAST-CLAUSE; (2) prompted self-debugging with `Qwen3.5-9B`, which directly predicts the faulty clause from the failed trajectory; (3) an execution-only reward model `Exec-RM-9B` trained from terminal execution labels without clause-level supervision; and (4) ablated variants of our approach, including `w/o Topology`, `w/o Fault Injection`, and `w/o Counterfactual Intervention`.

Environment. All experiments are conducted on a server with 8 NVIDIA A100 (80GB) GPUs. RL training is implemented with OpenRLHF[2]. We summarize the measured compute for the main local training and evaluation stages in Table 6; GPU-hours are computed as the number of GPUs multiplied by wall-clock hours.

Existing Assets and Licenses. We use public datasets, model checkpoints, software, and API services under their stated licenses or access terms, summarized in Table 7. For derived artifacts released with this submission, the supplemental code package includes a README with asset links, preprocessing steps, release notes, and applicable terms.

[2]https://github.com/openrlhf/openrlhf

Table 6: Compute resources for the main local training and evaluation stages.

Stage	Hardware	Wall-clock Time	GPU-hours
Clause-PRM training	8 A100 (80GB)	4 h 39 m	37.2
GRPO policy optimization	8 A100 (80GB)	19 h 1 m 46 s	152.2
Evaluation	2 A100 (80GB)	2 h	4.0

Table 7: Existing assets used in this work and their licenses or access terms.

Asset	Use in this work	License / access terms
BIRD [19]	Training and evaluation data	CC BY-SA 4.0
Spider [20]	Training and evaluation data	Apache-2.0
SynSQL-Complex-5K [26]	Synthetic training data	Apache-2.0
`Qwen3.5-9B` [37]	Base policy and PRM backbone	Apache-2.0
`OmniSQL-7B/14B` [18]	Open-source Text-to-SQL baselines	Apache-2.0 Hugging Face releases: 7B, 14B
`SQL-R1-7B` [26]	Open-source Text-to-SQL baseline	Apache-2.0
`XiYanSQL-7B/14B` [35]	Open-source Text-to-SQL baselines	Apache-2.0 Hugging Face releases: 7B, 14B
`AlphaSQL` [36]	Baseline system and reported comparison	MIT
`DeepEye-SQL` [34]	Baseline system and reported comparison	MIT
OpenRLHF	RL training framework	Apache-2.0
GPT, Claude, and Gemini APIs [32, 31, 33, 30]	Closed-source LLM baselines and candidate generation	Provider terms: OpenAI Service Terms, Anthropic Commercial Terms, Gemini API Additional Terms

E Candidate Verification Case Study

We provide a qualitative example from BIRD Dev to illustrate why clause-level verification can be more reliable than selecting candidates by sampling frequency or terminal-only reward scores. As shown in Figure 5, the question asks for the Italian flavor text of the card `Ancestor's Chosen` in the `card_games` database. The gold query must join `cards` with `foreign_data` through `uuid`, because the card name is stored in `cards`, whereas the localized flavor text and language are stored in `foreign_data`.

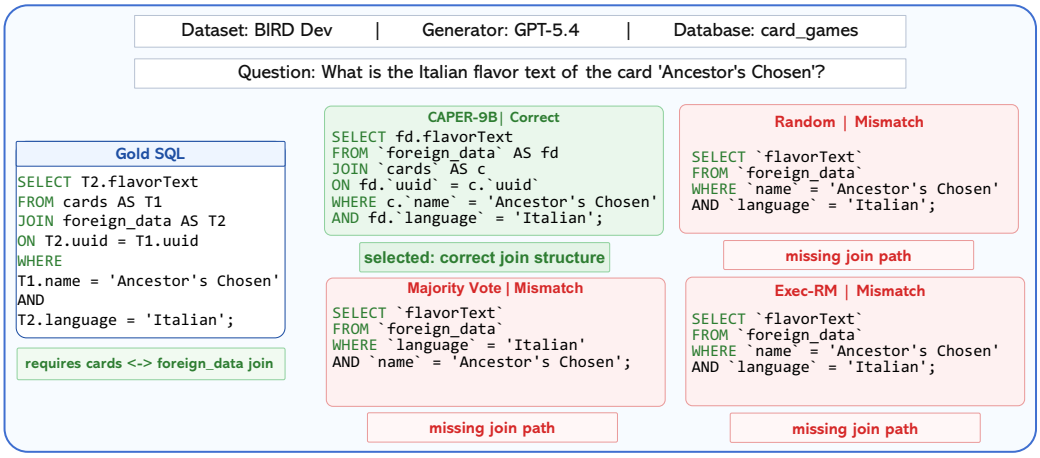

Figure 5: Candidate verification case study on BIRD Dev with GPT-5.4 candidates. Random, Majority Vote@8, and `Exec-RM` select candidates that directly filter `foreign_data` by card name and therefore miss the required `cards`–`foreign_data` join. `CAPER-9B` selects the only candidate that matches the gold join structure.

This example exposes a common failure mode for simpler selectors. The incorrect candidates mention the requested output column and language condition, but they omit the schema-level dependency between the English card name and its localized foreign record. As a result, Random, Majority Vote@8, and `Exec-RM` all choose candidates that fail execution correctness. In contrast, `CAPER-9B` selects the candidate that preserves the required clause structure: the join between `cards` and `foreign_data`, the name filter on `cards`, and the language filter on `foreign_data`. This case suggests that clause-level process supervision provides a transferable structural quality signal when terminal-only or frequency-based selectors prefer superficially plausible SQL.

F Evaluation Prompt Example

Example Evaluation Prompt

```
Task Overview:
You are a data science expert. Below, you are provided with a database
schema and a natural language question. Your task is to understand the
schema and generate a valid SQL query to answer the question.

Database Engine:
SQLite

Database ID:
formula_1

Database Schema:
CREATE TABLE ‘circuits‘ (
  ‘circuitId‘ INTEGER,
  ‘circuitRef‘ TEXT,
  ‘name‘ TEXT,
  ‘location‘ TEXT,
  ‘country‘ TEXT,
  ‘lat‘ REAL,
  ‘lng‘ REAL,
  ‘alt‘ INTEGER,
  ‘url‘ TEXT,
  PRIMARY KEY (‘circuitId‘)
);
CREATE TABLE ‘constructors‘ (
  ‘constructorId‘ INTEGER,
  ‘constructorRef‘ TEXT,
  ‘name‘ TEXT,
  ‘nationality‘ TEXT,
  ‘url‘ TEXT,
  PRIMARY KEY (‘constructorId‘)
);
CREATE TABLE ‘drivers‘ (
  ‘driverId‘ INTEGER,
  ‘driverRef‘ TEXT,
  ‘number‘ INTEGER,
  ‘code‘ TEXT,
  ‘forename‘ TEXT,
  ‘surname‘ TEXT,
  ‘dob‘ DATE,
  ‘nationality‘ TEXT,
  ‘url‘ TEXT,
  PRIMARY KEY (‘driverId‘)
);
CREATE TABLE ‘seasons‘ (
  ‘year‘ INTEGER,
  ‘url‘ TEXT,
  PRIMARY KEY (‘year‘)
);
CREATE TABLE ‘races‘ (
  ‘raceId‘ INTEGER,
  ‘year‘ INTEGER,
  ‘round‘ INTEGER,
  ‘circuitId‘ INTEGER,
  ‘name‘ TEXT,
  ‘date‘ DATE,
  ‘time‘ TEXT,
  ‘url‘ TEXT,
  PRIMARY KEY (‘raceId‘)
);
CREATE TABLE ‘constructorResults‘ (
  ‘constructorResultsId‘ INTEGER,
  ‘raceId‘ INTEGER,
  ‘constructorId‘ INTEGER,
  ‘points‘ REAL,
  ‘status‘ TEXT,
  PRIMARY KEY (‘constructorResultsId‘)
);
CREATE TABLE ‘constructorStandings‘ (
  ‘constructorStandingsId‘ INTEGER,
  ‘raceId‘ INTEGER,
```

```
  `constructorId` INTEGER,
  `points` REAL,
  `position` INTEGER,
  `positionText` TEXT,
  `wins` INTEGER,
  PRIMARY KEY (`constructorStandingsId`)
);
CREATE TABLE `driverStandings` (
  `driverStandingsId` INTEGER,
  `raceId` INTEGER,
  `driverId` INTEGER,
  `points` REAL,
  `position` INTEGER,
  `positionText` TEXT,
  `wins` INTEGER,
  PRIMARY KEY (`driverStandingsId`)
);
CREATE TABLE `lapTimes` (
  `raceId` INTEGER,
  `driverId` INTEGER,
  `lap` INTEGER,
  `position` INTEGER,
  `time` TEXT,
  `milliseconds` INTEGER,
  PRIMARY KEY (`raceId`, `driverId`, `lap`)
);
CREATE TABLE `pitStops` (
  `raceId` INTEGER,
  `driverId` INTEGER,
  `stop` INTEGER,
  `lap` INTEGER,
  `time` TEXT,
  `duration` TEXT,
  `milliseconds` INTEGER,
  PRIMARY KEY (`raceId`, `driverId`, `stop`)
);
CREATE TABLE `qualifying` (
  `qualifyId` INTEGER,
  `raceId` INTEGER,
  `driverId` INTEGER,
  `constructorId` INTEGER,
  `number` INTEGER,
  `position` INTEGER,
  `q1` TEXT,
  `q2` TEXT,
  `q3` TEXT,
  PRIMARY KEY (`qualifyId`)
);
CREATE TABLE `status` (
  `statusId` INTEGER,
  `status` TEXT,
  PRIMARY KEY (`statusId`)
);
CREATE TABLE `results` (
  `resultId` INTEGER,
  `raceId` INTEGER,
  `driverId` INTEGER,
  `constructorId` INTEGER,
  `number` INTEGER,
  `grid` INTEGER,
  `position` INTEGER,
  `positionText` TEXT,
  `positionOrder` INTEGER,
  `points` REAL,
  `laps` INTEGER,
  `time` TEXT,
  `milliseconds` INTEGER,
  `fastestLap` INTEGER,
  `rank` INTEGER,
  `fastestLapTime` TEXT,
  `fastestLapSpeed` TEXT,
  `statusId` INTEGER,
  PRIMARY KEY (`resultId`)
);
-- Foreign Keys:
-- `races`.`circuitId` can be joined with `circuits`.`circuitId`
-- `races`.`year` can be joined with `seasons`.`year`
-- `constructorResults`.`constructorId` can be joined with `constructors`.`constructorId`
```

```
-- ‘constructorResults‘.‘raceId‘ can be joined with ‘races‘.‘raceId‘
-- ‘constructorStandings‘.‘constructorId‘ can be joined with ‘constructors‘.‘constructorId‘
-- ‘constructorStandings‘.‘raceId‘ can be joined with ‘races‘.‘raceId‘
-- ‘driverStandings‘.‘driverId‘ can be joined with ‘drivers‘.‘driverId‘
-- ‘driverStandings‘.‘raceId‘ can be joined with ‘races‘.‘raceId‘
-- ‘lapTimes‘.‘driverId‘ can be joined with ‘drivers‘.‘driverId‘
-- ‘lapTimes‘.‘raceId‘ can be joined with ‘races‘.‘raceId‘
-- ‘pitStops‘.‘driverId‘ can be joined with ‘drivers‘.‘driverId‘
-- ‘pitStops‘.‘raceId‘ can be joined with ‘races‘.‘raceId‘
-- ‘qualifying‘.‘constructorId‘ can be joined with ‘constructors‘.‘constructorId‘
-- ‘qualifying‘.‘driverId‘ can be joined with ‘drivers‘.‘driverId‘
-- ‘qualifying‘.‘raceId‘ can be joined with ‘races‘.‘raceId‘
-- ‘results‘.‘statusId‘ can be joined with ‘status‘.‘statusId‘
-- ‘results‘.‘constructorId‘ can be joined with ‘constructors‘.‘constructorId‘
-- ‘results‘.‘driverId‘ can be joined with ‘drivers‘.‘driverId‘
-- ‘results‘.‘raceId‘ can be joined with ‘races‘.‘raceId‘
This schema describes the database’s structure, including tables, columns,
primary keys, foreign keys, and any relevant relationships or constraints.

Question:
Please list the location coordinates of the US circuits.

Output Rules:
1. You must output exactly one <think>...</think> block followed by one
   <answer>...</answer> block.
2. Inside <answer>, include exactly one ‘‘‘sql ... ‘‘‘ block containing
   runnable SQLite SQL.
3. Do not output any text before <think> or after </answer>.
4. Do not use <sql>...</sql>.
5. Keep <think> concise (<=120 words) so the final <answer> is never dropped.

Evidence:
location coordinates refers to (lat, lng); the US refers to country = ’USA’;

Instructions:
- Make sure you only output the information that is asked in the question. If
  the question asks for a specific column, make sure to only include that column
  in the SELECT clause, nothing more.
- The generated query should return all of the information asked in the question
  without any missing or extra information.
- Note that while the reasoning process and SQL query need to be enclosed within
  <think> </think> and <answer> </answer> tags respectively, this should not
  affect the quality of the SQL generation.
- The answer must contain the SQL query within ‘‘‘sql ‘‘‘ tags.

Output Format:
In your answer, please enclose the generated SQL query in a code block:
"""sql
-- Your SQL query
"""

Take a deep breath and think step by step to find the correct SQL query.
```

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We claimed in the abstract and introduction that we proposed an automatic annotation pipeline for clause-level reward annotation, developed a lightweight clause-level process reward model, improved Text-to-SQL execution accuracy, and improved failure localization.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss limitations in Appendix A, including dataset and SQL-dialect scope, clause-granularity limits, and the computational overhead of annotation.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: We provide the lemmas and corresponding proofs in Appendix B.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper provides detailed descriptions of the dataset curation process, model and parameter configuration, benchmarks and evaluation metrics, baselines, and the environment used for experiments in the main text. Additional details on data splits, hyperparameters, and other experimental settings are provided in the supplemental material.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in

some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We provide anonymized supplemental materials for data construction, model training, and evaluation, including scripts and instructions for reproducing the main experimental results.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper specifies the dataset curation process, model and parameter configuration, benchmarks, evaluation metrics, baselines, and experimental environment in the main text and appendix. Additional data splits, hyperparameters, and inference details are provided in Appendix D.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: We do not report error bars, confidence intervals, or statistical significance tests because full repeated RL training with large language models is computationally expensive. We instead report controlled comparisons, ablations, and results across multiple datasets and decoding settings.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper reports the GPU type and memory, the RL framework, key training hyperparameters, and measured wall-clock time and GPU-hours for Clause-PRM training, GRPO policy optimization, and evaluation in Appendix D.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work uses public Text-to-SQL benchmarks, model checkpoints, software, and API services under their stated licenses or access terms; does not involve human subjects, crowdsourcing, or newly collected personal data; and documents limitations, potential negative societal impacts, compute resources, release documentation, and asset licenses in the paper, appendix, checklist, and supplementary material.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.

- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The paper discusses positive impacts from improving the efficiency and accessibility of Text-to-SQL systems, as well as potential negative impacts from incorrect SQL generation, over-reliance on automated database access, and misuse on sensitive databases.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release data or models that have a high risk for misuse; the released artifacts are limited to Text-to-SQL annotation, training, and evaluation assets.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We cite the original papers for datasets, models, and software, summarize their licenses or provider access terms in Table 7, and include asset links and release notes in the supplemental README.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: We release a new clause-annotated dataset and a clause-level process reward model; the README in the supplemental code package documents data construction, splits, training configuration, evaluation protocol, and limitations.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.

- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: The paper describes the use of LLMs as the Text-to-SQL policy, clause-level process reward model, and evaluation baselines in the methodology and experimental setup. LLM assistance used only for writing and editing did not affect the core methodology, scientific rigor, or originality of the work.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.