

JUDGE SQL: Reasoning over SQL Candidates with Weighted Consensus Tournament

Jiayuan Bai^{1*} Xuanguang Pan^{1,2*} Chongyang Tao^{1†} Shuai Ma^{1†}

¹SKLCCSE Lab, Beihang University ²Beihang University

{baijiayuan, panxg, chongyang, shuaima}@buaa.edu.cn

Abstract

Text-to-SQL is a pivotal task that bridges natural language understanding and structured data access. Recent advances in large language models, through prompting, fine-tuning, and reinforcement learning, have greatly increased the likelihood of generating correct SQL queries. However, selecting the correct SQL from multiple candidates remains challenging. Existing approaches, such as self-consistency, Best-of-N decoding, and learned scalar ranking models, rely on shallow heuristics or lack explicit reasoning, making them prone to unstable judgments and unable to capture fine-grained semantic differences between similar queries. We propose JudgeSQL, a framework that combines reasoning-based SQL evaluation with a weighted consensus tournament. JudgeSQL develops a reasoning-based SQL judge model that distills reasoning traces with reinforcement learning guided by verifiable rewards, enabling accurate and interpretable judgments. Building on this, a weighted consensus tournament integrates explicit reasoning preferences with the generator’s execution-consistency confidence, improving both reliability and efficiency. Experiments on the BIRD benchmark demonstrate that JudgeSQL consistently outperforms prior methods, with strong generalization and robustness across model scales. The code is available at <https://github.com/Jiayuan-Bai/JudgeSQL>.

1 Introduction

Text-to-SQL has emerged as a pivotal task connecting natural language understanding with structured data access (Katsogiannis-Meimarakis and Koutrika, 2023; Liu et al., 2025b). By converting human language questions into executable SQL queries, Text-to-SQL empowers non-expert users to interact with complex databases. Despite recent

progress in LLMs, the task remains challenging due to the need for precise semantic understanding, schema reasoning, and handling of complex relational compositions (Liu et al., 2025b).

Recent progress has focused on the *generation* side based on LLMs, including better prompt strategies (Pourreza and Rafiei, 2023), supervised fine-tuning (Pourreza and Rafiei, 2024), multi-step generation, reinforcement learning (Pourreza et al., 2025b; Ma et al., 2025; Yao et al., 2025), or generative reasoning strategies (Zhang et al., 2025; Guo et al., 2025; Mao et al., 2026; Liu et al., 2025a; Pourreza et al., 2025a). While these approaches improve single-pass generation, they face a limitation: the variability and uncertainty of the sampling space, arising from the probabilistic decoding of language models and the complex, compositional reasoning required to map natural language to SQL. Beyond enhancing individual predictions, a promising direction is Best-of-N (BoN) sampling, a widely adopted strategy for test-time scaling (TTS) in LLMs (Lightman et al., 2024; Wang et al., 2024; Snell et al., 2024). This approach generates multiple candidate SQLs using a single model or an ensemble, and performs selection at inference time, which expands the hypothesis space and leads to improved accuracy and robustness. While TTS demonstrates clear empirical benefits, it shifts the inference paradigm from generating plausible queries to selecting among candidates based on semantic alignment.

Existing selection strategies remain lagging behind, despite the growing complexity of the candidate space. Current Best-of-N approaches for SQL selection typically rely on self-consistency (SC) (Wang et al., 2023), which selects the final output via majority voting over execution results (Dong et al., 2023; Gao et al., 2024; Talaei et al., 2025). While simple, such methods primarily depend on frequency and often fail to capture subtle semantic differences between candidates.

*Equal contribution.

†Corresponding author.

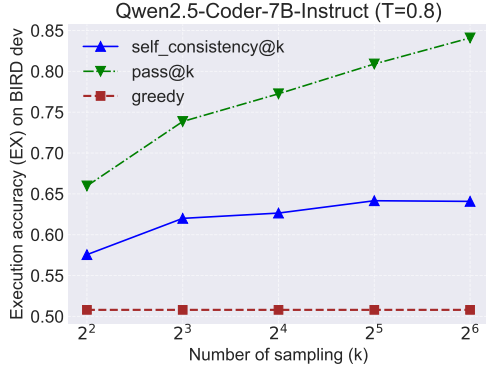


Figure 1: Trends of EX scores for greedy decoding, self-consistency, and Pass@k methods across different sampling sizes on the BIRD dev set.

More recently, a few studies have explored learned scalar-based judgment models obtained through direct fine-tuning of LLMs (Pourreza et al., 2025a). While often effective, they inherently lack mechanisms to generate intermediate reasoning steps, remain susceptible to inconsistent scoring and fragile reasoning chains, and struggle to distinguish semantically different queries with similar results. This limitation is especially pronounced in realistic scenarios, where semantically different SQL queries may yield comparable outputs. Empirically, the performance gap between greedy decoding, heuristic-based selection (self-consistency), and oracle selection (pass@k) can exceed 10 to 20 points in execution accuracy, as shown in Figure 1, demonstrating that current selection strategies leave a substantial portion of the model’s potential untapped. Consequently, selecting the correct candidates has emerged as the central bottleneck in fully harnessing the reasoning capabilities of LLMs for Text-to-SQL, motivating the need for principled selection mechanisms.

To address these challenges, we propose JudgeSQL, a novel framework that combines structured reasoning with weighted consensus mechanisms for SQL candidate judgment. To explore the reasoning capabilities of LLMs in SQL candidate evaluation, we first investigate the zero-shot performance of current LLMs for SQL selection by designing structured execution-guided prompts. Surprisingly, even without task-specific fine-tuning, LLMs demonstrate remarkably impressive performance compared to the widely-used SC. To further strengthen LLMs’ reasoning ability in SQL candidate judgment, we develop a reasoning-based SQL judge model that distills reasoning traces and employs reinforcement learning guided by verifiable rewards, enabling more accurate and interpretable

selection of candidate queries. Finally, based on the outputs of the reasoning-based judge, we propose a weighted consensus tournament mechanism, which combines the reasoning-based judge scores with the generator’s execution-consistency confidence to efficiently select the most reliable SQL queries.

We evaluate the proposed selection framework on the BIRD benchmark using sampled SQL candidates generated by representative models of varying sizes and architectures. We compare both the prompted reasoning model and the reinforced reasoning model using several tournaments. Results show that RL-trained reasoning models outperform direct prompting by promoting structured reasoning. The weighted consensus tournament combines selector preferences with generator confidence and achieves more accurate and efficient selections than the traditional double round-robin. Moreover, performance scales positively with the number of sampled candidates. Our contributions are fourfold:

- We investigate the capabilities of current LLMs for zero-shot reasoning in SQL selection by designing *structured execution-guided prompts* and find their performance already strong compared to the commonly used SC.
- We propose a *reasoning-based SQL judge model* that combines reasoning trace distillation with reinforcement learning guided by verifiable rewards, resulting in more accurate and interpretable SQL candidates selection.
- We introduce a *weighted consensus tournament* that combines reasoning-based SQL judgment and generator-derived execution consistency, achieving higher efficiency and performance than existing round-robin tournament method.
- Our model achieves superior performance across various settings and demonstrates strong cross-scale generalization and robustness to generator capacity, highlighting its potential as a reusable module for guiding SQL selection.

2 Related Works

2.1 Text-to-SQL Generation

Text-to-SQL generation has advanced from early rule-based and neural seq2seq models (Popescu et al., 2004; Li and Jagadish, 2014; Saha et al., 2016; Basik et al., 2018; Sun et al., 2018; Wang et al., 2020) to LLM-based paradigms characterized by semantic reasoning (Gao et al., 2024; Nan et al., 2023; Zhang et al., 2024; Liu et al., 2023).

To optimize single-pass generation quality, contemporary research explores advanced prompting strategies (Pourreza and Rafiei, 2023; Qu et al., 2024; Chi et al., 2025) and supervised fine-tuning (SFT) (Pourreza and Rafiei, 2024; Qin et al., 2025; He et al., 2025; Li et al., 2025b). More recently, preference optimization and reinforcement learning have been introduced to enhance compositionality. This includes Direct Preference Optimization (DPO) for improving semantic faithfulness (Rafailov et al., 2023; Yang et al., 2024; Liu et al., 2025a; Zhai et al., 2025) and reinforcement learning approaches such as GRPO that leverage rule-based rewards to guide structured reasoning paths (Shao et al., 2024; Pourreza et al., 2025b; Ma et al., 2025; Yao et al., 2025). However, these methodologies focus on altering the generator’s distribution, leaving the variability and uncertainty of the test-time sampling space unaddressed.

2.2 Post-processing on Text-to-SQL

While most Text-to-SQL research focuses on SQL generation, post-processing methods also play an important role in improving the reliability and correctness of generated SQL queries.

Self-correction methods improve LLM-generated SQL reliability through iterative refinement; single-model approaches (Talaie et al., 2025; Wang et al., 2025) use execution feedback or self-reflection, while multi-agent frameworks (Cen et al., 2025; Askari et al., 2025; Qu et al., 2025) employ collaborative reasoning among multiple agents to improve SQL correctness.

Beyond refining individual outputs, test-time scaling (TTS) (Snell et al., 2025) improves reliability by aggregating multiple candidate queries during inference. A common TTS strategy is output consistency (Wang et al., 2023), which selects results consistent across reasoning paths. Techniques such as self-consistency voting (Dong et al., 2023; Gao et al., 2024; Talaie et al., 2025; Li et al., 2025a) and cross-model consistency (Li et al., 2024) have been widely applied in text-to-SQL. Since consistency alone does not ensure correctness, N-best rankers reorder candidates using model-derived features or LLM-driven semantic preferences. Feature-enhanced rerankers (Hui et al., 2021) exploit hidden representations and external knowledge to rank SQL candidates, while ZeroNL2SQL (Fan et al., 2024) ranks PLM-generated SQL sketches based on semantic alignment with the input question. More recently, to distinguish fine-grained seman-

tic differences, recent studies leverage LLM-based judgment frameworks for candidate selection, employing either tournament-style pairwise comparison (e.g., CHASE-SQL (Pourreza et al., 2025a)) or listwise reranking (e.g., Mcs-SQL (Lee et al., 2025), XiYan-SQL (Liu et al., 2026)).

However, existing LLM-based judgment approaches lack explicit reasoning supervision and underutilize consensus signals. Scalar-based judgment models produce uninterpretable preference scores (e.g., CHASE-SQL), while listwise frameworks impose substantial cognitive burden as candidate sets grow (e.g., Mcs-SQL and XiYan-SQL). Moreover, prior tournament-based methods treat all candidates uniformly, ignoring implicit confidence signals from generation frequency. JudgeSQL addresses these limitations through a weighted consensus tournament that integrates explicit pairwise reasoning with implicit generator confidence for more robust SQL selection.

3 Problem Formalization

In the Text-to-SQL task, given a natural language question $x \in \mathcal{X}$ and a database schema d , the goal is to generate a SQL query $y \in \mathcal{Y}$ such that execution result ($\text{Exec}(y, d)$) correctly answers x . Rather than producing a single hypothesis under the model distribution $P_\theta(y | x, d)$, existing approaches approximate the space of plausible queries by sampling N candidates $\{y^{(1)}, \dots, y^{(N)}\} \sim P_\theta(y|x, D)$ using temperature-scaled nucleus or top- k decoding, typically with either powerful proprietary LLMs (e.g., GPT-4) or task-specific fine-tuned models (e.g., OmniSQL (Li et al., 2025b)), and then selecting among them through heuristic strategies.

While SQL generation has been widely studied, the key challenge lies in selecting the most reliable SQL from $\mathcal{Y}_x$. We therefore formulate this as learning a judgment model $\mathcal{R}(\cdot)$, interpretable as an outcome reward model, which identifies the best candidate $y^* \in \mathcal{Y}_x$ such that $\text{Exec}(y^*, d)$ yields the most accurate answer to the question. To implement this, we explore comparison strategies and adopt pair-wise judgment.

4 Methodology

4.1 Overview of JudgeSQL

To address inconsistent scoring and fragile reasoning in SQL candidate selection, we propose

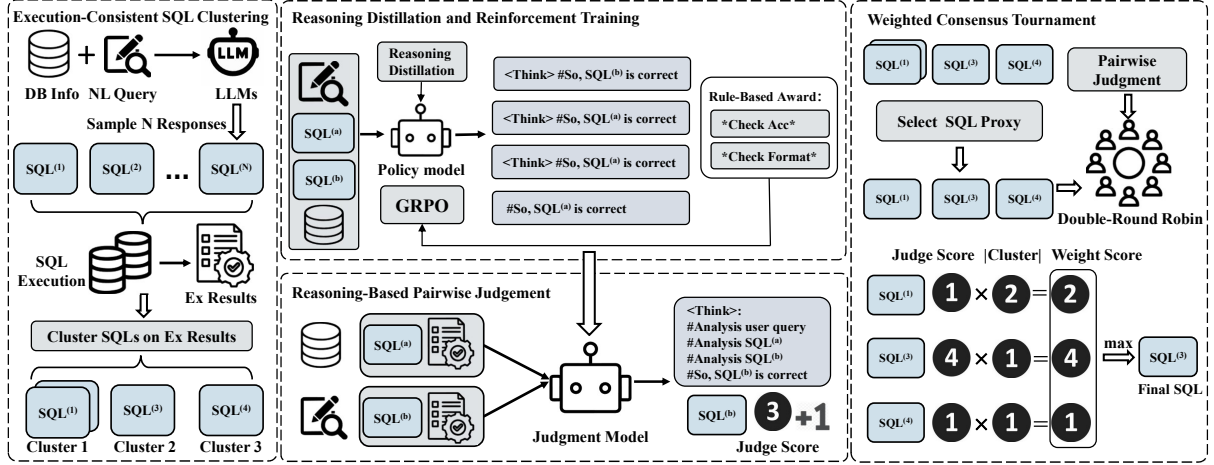


Figure 2: Overview of the JudgeSQL. $\langle \text{SQL}^{(a)}, \text{SQL}^{(b)} \rangle$ is a pair constructed from clusters for model judgment.

JudgeSQL, a novel framework that integrates structured reasoning with a weighted consensus mechanism. JudgeSQL employs a SQL judgment model that distills reasoning traces from stronger LLMs and leverages reinforcement learning guided by verifiable rewards, enabling more accurate and interpretable evaluation of SQL candidates. Building on the judge’s outputs, a weighted consensus tournament combines explicit judge scores with the generator’s implicit confidence to efficiently select the most reliable SQL queries. An overview of our framework is illustrated in Figure 2. Next, we introduce our reasoning-based SQL judgment model and weighted consensus tournament in detail.

4.2 Reasoning-reinforced SQL Judgment

To improve judgment quality in complex SQL comparison scenarios, we propose a reasoning-reinforced SQL judgment framework. Our method employs a two-stage training strategy: reasoning trace distillation followed by reinforcement learning with verifiable rewards. In the first stage, an instruction-tuned language model is trained on synthetic reasoning traces constructed from SQL candidate pairs to distinguish semantically similar queries through reasoning. In the second stage, the model is further optimized via reinforcement learning using execution-verifiable rewards to improve pairwise SQL judgment accuracy.

Reasoning Distillation for SQL Judgment. Instruction-tuned LLMs such as Qwen can be adapted for SQL judgment via prompting, but they often produce inconsistent predictions without structured reasoning supervision. To address this, we introduce a distillation framework that uses synthetic reasoning traces for warm-up training.

We first construct a preference dataset $\mathcal{D}_{\text{pref}}$

by leveraging execution-consistent clustering. For each query x with ground-truth SQL y_{gt} from a training dataset (e.g., BIRD), we sample SQL candidates using a specific generator LLM (i.e., Qwen2.5-Coder-7B-Instruct). These candidates and ground-truth SQL are executed against the database and grouped into execution-consistent clusters $\{\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_K\}$, where queries within each cluster yield identical execution results. We designate the cluster matching the ground-truth execution result as $\mathcal{C}_{pos}$, while the remaining incorrect clusters form the negative set $\mathcal{C}_{neg} = \{\mathcal{C}_1, \dots, \mathcal{C}_K\} \setminus \{\mathcal{C}_{pos}\}$. Pairs are then constructed by pairing $y_{pos} \in \mathcal{C}_{pos}$ with a sampled SQL query y_{neg} from each negative cluster in $\mathcal{C}_{neg}$. Each positive SQL and negative cluster is used at most once to avoid over-representing near-duplicate queries while preserving semantic diversity, yielding $N = \min(|\mathcal{C}_{pos}|, |\mathcal{C}_{neg}|)$ preference pairs. To eliminate position bias, each pair is presented in both candidate orders to ensure training symmetry.

From $\mathcal{D}_{\text{pref}}$, we sample a subset of M pairs to generate high-quality reasoning traces. For each pair, a high-capacity model $\mathcal{G}^*$ (e.g., GPT-4o) is used to generate a reasoning trace $r^{(i)}$ that explains why $y_{pos}^{(i)}$ is preferred over $y_{neg}^{(i)}$:

$$r^{(i)} = \mathcal{G}^*(x^{(i)}, d_{uni}^{(i)}, y_{pos}^{(i)}, y_{neg}^{(i)}, e_{pos}^{(i)}, e_{neg}^{(i)}), \quad (1)$$

where $d_{uni}^{(i)}$ denotes the union of schema referenced by both SQL candidates while $e_{pos}^{(i)}$ and $e_{neg}^{(i)}$ denote the execution results of $y_{pos}^{(i)}$ and $y_{neg}^{(i)}$.

After filtering out traces inconsistent with the ground-truth preference label, $l^{(i)}$, the final distillation dataset $\mathcal{D}_{\text{distill}} = \{\hat{x}^{(i)}, r^{(i)}\}_{i=1}^M$ is constructed with

inputs $\hat{x}^{(i)} = (x^{(i)}, d_{uni}^{(i)}, y_{pos}^{(i)}, y_{neg}^{(i)}, e_{pos}^{(i)}, e_{neg}^{(i)})$. We optimize the model via supervised fine-tuning (SFT) using the negative log-likelihood loss:

$$\mathcal{J}_{\text{SFT}}(\theta) = - \sum_{i=1}^M \sum_{t=1}^{T_i} \log P_{\theta}(r_t^{(i)} | r_{<t}^{(i)}, \hat{x}^{(i)}), \quad (2)$$

where $r_{<t}^{(i)}$ denotes the preceding tokens in the sequence of length T_i . This warm-up phase ensures the judgment model grounds its final decisions in verifiable semantic reasoning paths.

Reinforcement Training for SQL Judgment.

While knowledge distillation adapts a generative model for SQL judgment, it may overfit to specific reasoning patterns and limit the model’s ability to generalize to more complex judgment scenarios, which is critical for the task. To address this limitation, we further optimize the model using reinforcement learning (RL), which provides a more flexible framework for reasoning-oriented optimization. Specifically, we optimize the judgment model with reasoning-informed reward signals using the GRPO algorithm, formally defined as:

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{\substack{\hat{x} \sim \mathcal{D}_{\text{pref}} \\ \{o_i\} \sim \pi_{\theta_{\text{old}}}}} \left[\frac{1}{|G|} \sum_{i=1}^{|G|} \min \left(r_{\theta}(o_i) A_i, \text{clip}(r_{\theta}(o_i), 1 - \epsilon, 1 + \epsilon) A_i \right) \right] \quad (3)$$

Here, $\hat{x}$ denotes an input sampled from the preference dataset $\mathcal{D}_{\text{pref}}$, and o_i is the generated output containing both the reasoning trace and the final judgment l . The group size G denotes the number of judgment traces generated for the same input. π_{θ} and $\pi_{\theta_{\text{old}}}$ represent the current and old policy models, respectively, and $r_{\theta}(o_i) = \frac{\pi_{\theta}(o_i|\hat{x})}{\pi_{\theta_{\text{old}}}(o_i|\hat{x})}$ denotes the probability ratio. A_i is the group-relative advantage computed from normalized reward r .

The reward function jointly enforces reasoning format and judgment correctness. A reward of one ($r = 1$) is assigned only if the output follows the required format, with reasoning enclosed in `<think>...</think>` and the final answer enclosed in `<answer>...</answer>`, and the predicted label matches the ground-truth label. Otherwise, the reward is zero. Formally,

$$r(o_i) = \mathbf{1}[\text{format_check}(o_i) \wedge l_{\text{pred}}(o_i) = l]. \quad (4)$$

$l_{\text{pred}}(o_i)$ denotes the predicted judgment. The binary reward formulation maintains reward verifiability and avoid introducing heuristic supervision.

4.3 Weighted Consensus Tournament (WCT)

Despite strong pairwise judgment models, efficiently selecting the correct SQL from many similar or execution-equivalent candidates remains challenging. Prior work (Pourreza et al., 2025a) typically relies on exhaustive pairwise comparisons, which suffer from two limitations: (i) semantically equivalent SQLs are still treated as separate candidates in the comparison process, resulting in redundant comparisons and potentially noisy vote aggregation; (ii) all candidates are treated equally, ignoring the frequency signal shared by semantically consistent SQLs, which can provide important evidence for reliable selection.

To address these limitations, we propose a *weighted consensus tournament* that combines the robustness of execution-consistent clustering with the discriminative power of pairwise judgment. We first form execution-consistent clusters and randomly select one representative SQL query $y^{(k)} \in \mathcal{C}_k$ from each set as its proxy in subsequent comparisons¹. Instead of comparing all queries, only proxy SQLs compete in a double round-robin tournament. For each pair of proxy SQLs ($y^{(k)}, y^{(j)}$), we provide both SQLs, their respective execution results ($e^{(k)}, e^{(j)}$) and the union schemas involved in two SQL candidates d_{uni} , to the reasoning judgment model $\mathcal{R}_{\psi}$. The model then judges which query better answers the question x , defined as:

$$\mathcal{R}_{\psi}(x, d_{uni}, y^{(k)}, e^{(k)}, y^{(j)}, e^{(j)}) = \begin{cases} 1 & \text{if } y^{(k)} \succ_x y^{(j)}, \\ 0 & \text{otherwise,} \end{cases} \quad (5)$$

where $y^{(k)} \succ_x y^{(j)}$ indicates that $y^{(k)}$ is preferred over $y^{(j)}$ for the question x . After completing all pairwise comparisons across different execution-consistent sets, the score of each set $\mathcal{C}_k$ is computed by aggregating its wins throughout the tournament:

$$S(\mathcal{C}_k) = \sum_{j \neq k} \mathcal{R}_{\psi}(x, d_{uni}, y^{(k)}, e^{(k)}, y^{(j)}, e^{(j)}). \quad (6)$$

While the unweighted score ($S(\mathcal{C}_k)$) reflects judgment signals from reasoning-based comparisons, it overlooks the cardinality of each execution-consistent cluster ($|\mathcal{C}_k|$), which captures the generator’s implicit confidence through the frequency of semantically aligned SQL candidates. To incorporate this additional consensus signal, we introduce

¹ Since queries within an execution-consistent cluster yield identical execution results and are semantically equivalent, we randomly select the representative query. Experiments with five random seeds show no statistically significant differences under a paired t-test. Selecting the proxy SQL using the highest generation likelihood yields comparable results.

a frequency-aware weighted score:

$$\mathcal{S}_w(\mathcal{C}_k) = |\mathcal{C}_k| \cdot \mathcal{S}(\mathcal{C}_k), \quad k \in [1, K], \quad (7)$$

This weighting scheme combines the explicit judgment scores with the generator’s implicit frequency-based confidence. The cluster with the highest weighted score is selected, i.e., $\mathcal{C}^* = \arg \max_k \mathcal{S}_w(\mathcal{C}_k)$. Ties are resolved by choosing the cluster with the higher cardinality. Finally, we return the proxy SQL from $\mathcal{C}^*$ as the final prediction. Algorithm A presents the procedure of WCT.

Compared with a standard double round-robin tournament, our WCT offers three advantages. First, it reduces computational cost by comparing only cluster representatives instead of performing exhaustive candidate-level pairwise comparisons. Second, by consolidating execution-consistent SQLs before selection, WCT reduces noisy and redundant voting among semantically equivalent candidates, leading to more stable judgment aggregation. Third, it exploits the consensus strength within each execution-consistent set, allowing the frequency of semantically aligned SQLs to serve as implicit confidence weights, leading to more accurate SQL selection.

5 Experiments

5.1 Experimental Setup

Datasets. Following recent studies on Text-to-SQL (Talaie et al., 2025; Pourreza et al., 2025a; Li et al., 2025a,b; Liu et al., 2025a), we conducted our experiments on BIRD (Li et al., 2023), a widely-used large-scale cross-domain text-to-SQL benchmark that incorporates external knowledge to handle complex scenarios. Details of the training dataset construction are provided in Appendix B.1.

Metrics. Consistent with previous work (Talaie et al., 2025; Pourreza et al., 2025a; Li et al., 2025a,b; Wang et al., 2025; Liu et al., 2025a), we use execution accuracy (EX) (Li et al., 2023) as our evaluation metric. EX considers a generated SQL query correct if its execution result matches that of the ground-truth query, and is preferable because it avoids incorrectly penalizing valid queries when multiple correct SQL expressions exist. To further evaluate pairwise comparison performance, we also introduce judgment accuracy (JA) in our analysis experiments. JA measures the proportion of test-time pairwise comparisons in which the model identifies the correct SQL candidate.

Baselines. Our pipeline consists of two stages: SQL candidate generation and SQL selection. In the SQL generation stage, we employ representative SQL generation models, including Qwen-2.5-Coder-Instruct (Hui et al., 2024), XiYanSQL-QwenCoder (Liu et al., 2026), and OmniSQL (Li et al., 2025b) at both 7B and 32B scales, enabling comprehensive evaluation across diverse model architectures and scales.

In the SQL selection stage, we compare our approach against three widely used baseline selection strategies: 1) Self-Consistency (SC), as adopted in (Gao et al., 2024; Xie et al., 2025); 2) Double Round-Robin Tournament (DRT), as adopted in (Pourreza et al., 2025a); and 3) Listwise Judgment (List), as adopted in (Liu et al., 2026; Lee et al., 2025). Furthermore, we evaluate different judgment models, including: 1) the Prompted SQL Judgment Model (PJudge), and 2) the Direct-SFT Judgment Model (Direct-SFT), which is adopted in (Pourreza et al., 2025a). The details of these baselines are provided in Appendix B.2.

Implementation Details. To ensure both the generality of our approach and the fairness of comparison, we employ Qwen2.5-Coder-7B-Instruct (Hui et al., 2024) as the backbone SQL judgment model for all settings. For all SQL generation models, We use temperature sampling ($T = 0.8$) to generate multiple SQL candidates, with $N \in \{4, 8, 16, 32, 64\}$. For the greedy baseline, we perform single-pass deterministic decoding by setting the temperature to 0. In the SQL selection stage, we set the decoding temperature of all judgment models to 0 to obtain deterministic outputs. We provide more implementation details in Appendix B.3 and the prompts for PJudge and RJudge in Appendix E.

5.2 Main Results

Table 1 summarizes the experimental results for the WCT, equipped with either prompted SQL judgment model (PJudge) or reinforced SQL judgment model (RJudge), across a variety of SQL generation models including general-purpose (e.g., Qwen2.5-Coder-Instruct) and domain-specific models (e.g., XiYanSQL, OmniSQL) of varying capacities (7B/32B) and sampling sizes.

Our proposed WCT, equipped with either PJudge or RJudge, consistently outperforms SC across nearly all settings. Notably, PJudge with simple execution-guided prompting already establishes a strong baseline. Despite this baseline, incorporat-

Table 1: Evaluation results on BIRD Dev. We compare different SQL generation models and selection strategies across varying sample sizes (N) by each generation model. For reference, we also report the results of greedy sampling, which serve as a baseline for evaluating the benefits of different sampling and selection strategies.

SQL Generation Model	Judgment Model	Selection Strategy	Dev EX(%)					
			Greedy	$N=4$	$N=8$	$N=16$	$N=32$	$N=64$
Qwen2.5-Coder-7B-Instruct	-	SC	50.85	57.63	62.39	62.71	63.89	63.89
	PJudge	WCT	50.85	59.39	64.99	65.78	66.75	67.01
	RJudge	WCT	50.85	62.19 ($\uparrow$ 4.56)	67.01 ($\uparrow$ 4.62)	66.30 ($\uparrow$ 3.59)	68.64 ($\uparrow$ 4.75)	68.71 ($\uparrow$ 4.82)
XiYanSQL-7B	-	SC	53.59	59.32	62.97	63.89	64.47	65.06
	PJudge	WCT	53.59	61.80	64.99	65.38	66.75	68.19
	RJudge	WCT	53.59	63.49 ($\uparrow$ 4.17)	66.43 ($\uparrow$ 3.46)	68.38 ($\uparrow$ 4.49)	68.51 ($\uparrow$ 4.04)	69.75 ($\uparrow$ 4.69)
OmniSQL-7B	-	SC	63.89	65.51	66.75	67.28	67.47	67.28
	PJudge	WCT	63.89	65.71	67.41	68.38	67.80	67.93
	RJudge	WCT	63.89	68.38 ($\uparrow$ 2.87)	69.69 ($\uparrow$ 2.94)	70.47 ($\uparrow$ 3.19)	70.47 ($\uparrow$ 3.00)	70.73 ($\uparrow$ 3.45)
Qwen2.5-Coder-32B-Instruct	-	SC	64.47	67.28	67.73	68.12	68.64	68.71
	PJudge	WCT	64.47	67.86	68.64	69.17	69.95	70.01
	RJudge	WCT	64.47	70.01 ($\uparrow$ 2.73)	70.34 ($\uparrow$ 2.61)	70.80 ($\uparrow$ 2.68)	71.06 ($\uparrow$ 2.42)	71.38 ($\uparrow$ 2.67)
XiYanSQL-32B	-	SC	63.95	66.56	68.51	68.19	68.84	68.51
	PJudge	WCT	63.95	67.28	69.23	69.10	69.81	70.08
	RJudge	WCT	63.95	69.56 ($\uparrow$ 3.00)	70.53 ($\uparrow$ 2.02)	71.19 ($\uparrow$ 3.00)	71.19 ($\uparrow$ 2.35)	71.90 ($\uparrow$ 3.39)
OmniSQL-32B	-	SC	64.54	67.14	67.47	67.21	67.21	67.14
	PJudge	WCT	64.54	67.47	66.75	67.54	67.34	68.58
	RJudge	WCT	64.54	69.43 ($\uparrow$ 2.29)	69.82 ($\uparrow$ 2.35)	70.40 ($\uparrow$ 3.19)	70.73 ($\uparrow$ 3.52)	70.27 ($\uparrow$ 3.13)

Table 2: Execution accuracy (EX, %) of different selection strategies across various generation models and judgment models (PJudge vs. RJudge).

Selection Strategy	Generation Model			
	Qwen7B	XiYanSQL7B	Qwen32B	XiYanSQL32B
Judgment Model: PJudge				
WCT	64.99	64.99	68.64	69.23
CT	60.76	59.65	66.04	64.93
DRT	63.30	62.91	67.14	66.69
List	62.06	61.02	67.92	66.42
Judgment Model: RJudge				
WCT	67.01	66.43	70.34	70.53
CT	66.04	65.12	69.75	69.04
DRT	66.88	65.65	69.75	68.58
List	64.46	62.58	68.84	67.53

ing RJudge further enhances performance, achieving an average EX improvement of 3.33% over SC and 1.35% over WCT+PJudge. WCT+RJudge also achieves substantial gains over greedy decoding, including +6.84% on OmniSQL-7B and +7.95% on XiYanSQL-32B, highlighting the effectiveness of reinforced judgment at test time.

WCT+RJudge delivers consistent improvements across diverse SQL generation frameworks and model scales. Across Qwen2.5-Coder, XiYanSQL, and OmniSQL generators, WCT+RJudge consistently improves EX over self-consistency, with gains reaching 4.82% on Qwen2.5-Coder-7 and 3.45% on OmniSQL-7B under $N = 64$. Notably, WCT+RJudge yields substantial improvements on both 7B and 32B generators. For example, Qwen2.5-Coder-7B-Instruct achieves a 4.82% EX gain under $N = 64$, while meaningful gains

Table 3: Average judgment counts per question across different settings.

Generation Model	Selection Strategy	Sampling Numbers				
		4	8	16	32	64
Qwen7B	DRT	2.6	12.1	53.1	215.2	876.4
	WCT	1.5	3.8	9.4	21.2	48.6
Qwen32B	DRT	2.0	9.3	40.1	167.2	680.9
	WCT	0.9	2.0	3.9	7.3	15.4

are also observed on stronger 32B generators, including a 3.39% improvement on XiYanSQL-32B. Moreover, although RJudge is trained solely on candidates generated by Qwen2.5-Coder-7B-Instruct, it transfers effectively across all evaluated generators, indicating that the learned judgment capability is not tightly coupled to a specific backbone model.

We further analyze the impact of sampling size across different generators and selection strategies. As the sampling size N increases, performance generally improves across all methods, although the gains are more pronounced for 7B generators than for 32B models, likely because stronger generators already produce more concentrated high-quality candidate distributions. Different selection strategies also exhibit distinct scaling behaviors. While SC often saturates or fluctuates at larger sampling sizes, WCT-based methods continue to benefit consistently from additional candidates. In particular, WCT+RJUDGE demonstrates strong sample efficiency: on OmniSQL-7B, using only $N = 4$ surpasses SC with $N = 64$ (68.38% vs. 67.28%).

Table 4: Execution Accuracy (EX) and Judgment Accuracy (JA) (%) under different training recipes.

Training Recipe	Generation Model							
	Qwen-7B		XiYanSQL-7B		Qwen-32B		XiYanSQL-32B	
	EX	JA	EX	JA	EX	JA	EX	JA
Base	64.99	71.33	64.99	73.91	68.64	62.91	69.23	67.53
Direct-SFT	66.30	80.13	65.97	81.38	69.30	71.38	69.56	73.57
GRPO	66.23	78.97	66.88	79.97	69.56	72.34	70.21	73.14
RJudge	67.01	82.55	66.43	83.04	70.34	75.92	70.53	75.51

5.3 Discussions

Comparison of selection strategies. To evaluate WCT, we compare its execution accuracy (EX) against DRT, CT (WCT without cardinality weighting), and Listwise judgment across multiple generators ($N = 8$). As shown in Table 2, WCT consistently delivers the highest EX. For PJudge on Qwen-7B, WCT outperforms CT by 4.23 points, while beating DRT and Listwise judgment by 1.69 and 2.93 points, respectively. The consistent gains of WCT over CT suggest that frequency weighting is most effective when coupled with accurate pairwise judgments, rather than used as a standalone consensus signal. Under RJudge, WCT continues to maintain clear advantages despite the stronger GRPO-trained baseline, demonstrating the robustness of our tournament design.

To analyze the efficiency of WCT against DRT, Table 3 report the average number of judgments per question. While DRT requires twice as many judgments at $N = 4$, the gap widens at $N = 64$, where WCT reduces judgment counts by over $18\times$ on Qwen-7B and $44\times$ on Qwen-32B. This efficiency stems from WCT’s clustering, which avoids exhaustive pairwise comparisons across candidates.

Effects of training recipes. Table 4 compares Direct-SFT (label prediction only) and standalone GRPO against our full recipe (RJudge) under WCT ($N = 8$) to analyze the impact of model training on judgment performance. Besides EX, We also report judgment accuracy to better reflect judge capability beyond tournament outcomes. Performance improves progressively with recipe refinement. Direct-SFT provides clear gains over the base model, particularly improving judgment accuracy by 6.04–8.80% across generators. Standalone GRPO yields mixed results occasionally underperforming SFT. In contrast, RJudge achieves the best performance across all settings, indicating that distillation warmup is critical for bootstrapping reinforcement learning in SQL judgment. We also observe that candidates generated by 32B models

Figure 3: Performance by query difficulty on BIRD dev.

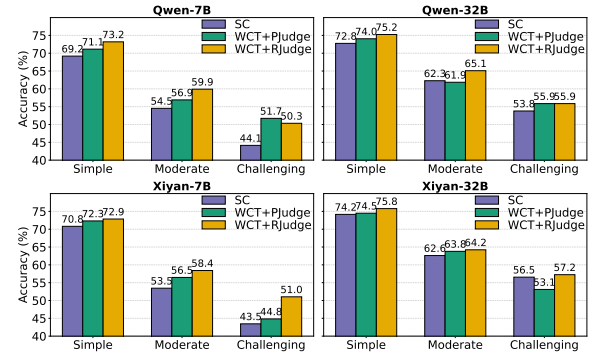


Table 5: Evaluation Results on Out-of-Domain Benchmarks with Different Selection Strategies (EX %).

Benchmark	Selection Strategy		
	Greedy	SC	RJudge
Spider-dk	67.48	72.15	72.71
Spider-syn	63.05	74.85	77.08
ScienceBenchmark	45.15	47.32	50.34

are generally harder to discriminate than those from 7B models, suggesting that stronger generators produce more challenging candidate distributions.

Evaluation on out-of-domain benchmark To further evaluate the out-of-domain generalization capabilities of our proposed framework, we assess the performance of the reinforced SQL judgment model equipped with the Weighted Consensus Tournament (WCT+RJudge) on three unseen datasets: Spider-dk, Spider-syn, and ScienceBenchmark. The SQL generation model is Qwen2.5-Coder-7B-Instruct and $N = 8$. As shown in Table 5, WCT+RJudge consistently outperforms both greedy decoding and the widely adopted Self-Consistency (SC) baseline across all evaluated scenarios. Notably, WCT+RJudge achieves 77.08% execution accuracy (EX) on Spider-syn and 50.34% on ScienceBenchmark, surpassing SC by 2.23 and 3.02 absolute percentage points, respectively. These results confirm that the structured reasoning capabilities reinforced via GRPO, combined with the frequency-aware consensus mechanism of WCT, are not overfitted to the domain-specific characteristics of the BIRD dataset. Instead, our framework exhibits robustness and superior cross-domain generalization, validating its effectiveness as a universally applicable SQL selection utility.

Performance on queries of varying difficulty Figure 3 reports the results on the BIRD dev set across different query difficulty levels, where queries are categorized as Simple, Moderate, and

Table 6: Comparison of different judgment models under zero-shot reasoning with WCT(Dev EX%).

Judgment Model	Generation Model			
	Qwen7B	XiYan7B	Qwen32B	XiYan32B
SC	62.39	62.97	67.73	68.51
Deepseek-6.7B	64.02	65.45	68.64	68.97
Qwen-7B	64.99	64.99	68.64	69.23
Llama-8B	64.34	64.47	68.12	68.51

Challenging by human experts. WCT+PJudge and WCT+RJudge outperform the SC baseline across all difficulty levels. The gains are pronounced on Challenging queries, with improvements of up to 7.5%, indicating that execution-aware selection is particularly beneficial for difficult SQL generation scenarios. This consistent advantage of our execution-aware selection methods confirms their benefit regardless of query complexity, with substantial gains on challenging queries where precision is critical for practical deployment.

Comparison of different judgment models To evaluate the robustness of WCT across different judgment model architectures, we further use Llama3-8B-Instruct (Grattafiori et al., 2024) and DeepSeek-Coder-6.7B-Instruct (Guo et al., 2024) as zero-shot judges, in addition to Qwen-7B. This comparison allows us to examine whether the effectiveness of WCT depends on a specific judgment-model backbone or can transfer across models with different architectures and reasoning capabilities. As shown in Table 6, all three judgment models generally outperform the SC baseline across different SQL generators with $N = 8$, demonstrating the consistent benefit of introducing explicit pairwise judgment into the selection process. Among them, Qwen and DeepSeek achieve the strongest overall performance, while Llama is slightly weaker in several settings. Despite these differences in absolute judgment quality, WCT remains consistently effective across all evaluated backbones. These results indicate that the gains of WCT are not specific to Qwen and that the framework is overall robust to the choice of underlying judgment model.

Evaluation on RL-posttrained text-to-SQL generators Recent Text-to-SQL models increasingly employ reinforcement learning to improve reasoning and generation quality. Such post-training may alter the candidate distribution and potentially reduce the need for candidate selection. To examine whether the selection gap remains in this setting,

Table 7: Evaluation results on RL-posttrained SQL-R1 generators with $N = 8$ (EX, %).

Generator	Greedy	SC	CT +	WCT +	Pass@8
			RJudge	RJudge	
SQL-R1-3B	48.4	54.6	58.3	59.6	68.0
SQL-R1-7B	58.9	66.6	68.4	69.4	75.1
SQL-R1-14B	63.2	67.1	68.3	69.4	76.5

we evaluate JudgeSQL on the SQL-R1 (Ma et al., 2025) model family at 3B, 7B, and 14B scales. For each generator, we sample $N = 8$ candidates and directly apply our existing RJudge without any additional training or generator-specific adaptation.

As shown in Table 7, WCT+RJudge consistently improves over SC by 2.3–5.0 EX points across all three model scales. Importantly, a substantial gap between SC and Pass@8 remains even after RL post-training. For example, SQL-R1-14B achieves 76.5 Pass@8 but only 67.1 EX with SC, leaving a 9.4-point selection gap. WCT+RJudge improves the result to 69.4 without any generator-specific adaptation. These results indicate that RL post-training improves candidate quality but does not eliminate the candidate-selection problem, and JudgeSQL remains effective on RL-posttrained Text-to-SQL generators.

6 Conclusion

In this paper, we introduce JudgeSQL, a framework for SQL candidates selection that integrates structured reasoning and consensus-based decision mechanisms. JudgeSQL enhances judgment accuracy and interpretability through reinforcement learning with verifiable rewards, alleviating the limitations of heuristic strategies such as execution-based majority voting and prior pairwise judgment methods. The weighted consensus tournament additionally combines reasoning-based judgment with generator likelihood, enabling more reliable and efficient SQL selection. Experiments show that JudgeSQL consistently outperforms existing methods, with good generalization and robustness to generator capacity.

Acknowledgements

We thank the reviewers for their constructive comments and suggestions. This work was supported by the National Natural Science Foundation of China (Grant No. 62572034, U22B2021, and U24B20143), State Key Laboratory of Complex & Critical Software Environment (SKLCCSE), and CIE–Tianyi Cloud Research Program.

Limitations

Despite its effectiveness, JudgeSQL still has several limitations. One limitation is that, although WCT significantly reduces the comparison cost relative to exhaustive tournaments, the framework still incurs substantial inference overhead due to multi-sample SQL generation, and reasoning-based pairwise judgment. The overall computational cost increases with the sampling size, which may limit practical deployment in latency-sensitive settings.

Another limitation is that the framework relies heavily on execution-based signals, including execution-consistent clustering and execution-aware judgment. As a result, its effectiveness may degrade in scenarios where database execution is unavailable or computationally expensive. Moreover, in practical deployments, large sampling budgets may introduce additional latency due to repeated database executions.

Moreover, although we mitigate positional bias through bidirectional pair construction, reasoning-based pairwise judgments may still exhibit residual sensitivity to candidate ordering and presentation formats, which could affect selection stability in highly ambiguous cases.

Ethical Considerations

While JudgeSQL improves SQL candidate selection, several ethical considerations should be noted. First, the framework requires access to database content during inference, which may include sensitive or private information; improper use could lead to privacy violations. Second, as JudgeSQL relies on automated reasoning to select SQL queries, errors in judgment could propagate to downstream systems, potentially causing incorrect decisions or unintended data exposure. Finally, the multi-sample generation and reasoning-based inference introduce additional computational and serving costs, which may limit accessibility and scalability in resource-constrained deployment settings.

References

- Arian Askari, Christian Pölitz, and Xinye Tang. 2025. MAGIC: generating self-correction guideline for in-context text-to-sql. In *AAAI-25*.
- Fuat Basik, Benjamin Hättasch, Amir Ilkhechi, Arif Usta, Shekar Ramaswamy, Prasetya Utama, Nathaniel Weir, Carsten Binnig, and Ugur Cetintemel. 2018. Dbpal: A learned nl-interface for databases. In *SIGMOD*, pages 1765–1768.
- Jipeng Cen, Jiaxin Liu, Zhixu Li, and Jingjing Wang. 2025. Sqlfixagent: Towards semantic-accurate text-to-sql parsing via consistency-enhanced multi-agent collaboration. In *Proceedings of the AAAI Conference on Artificial Intelligence*.
- Yongdong Chi, Hanqing Wang, Yun Chen, Yan Yang, Jian Yang, Zonghan Yang, Xiao Yan, and Guanhua Chen. 2025. Pi-sql: Enhancing text-to-sql with fine-grained guidance from pivot programming languages. In *EMNLP (Findings)*, pages 25120–25144.
- Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Jinshu Lin, Dongfang Lou, et al. 2023. C3: Zero-shot text-to-sql with chatgpt. *arXiv preprint arXiv:2307.07306*.
- Ju Fan, Zihui Gu, Songyue Zhang, Yuxin Zhang, Zui Chen, Lei Cao, Guoliang Li, Samuel Madden, Xiaoyong Du, and Nan Tang. 2024. Combining small language models and large language models for zero-shot NL2SQL. *VLDB*.
- Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-sql empowered by large language models: A benchmark evaluation. *Proceedings of the VLDB Endowment*.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.
- Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, et al. 2024. Deepseek-coder: When the large language model meets programming—the rise of code intelligence. *arXiv*.
- Yu Guo, Dong Jin, Shenghao Ye, Shuangwu Chen, Jian Yang, and Xiaobin Tan. 2025. Sqlforge: Synthesizing reliable and diverse data to enhance text-to-sql reasoning in llms. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 8441–8452.
- Mingqian He, Yongliang Shen, Wenqi Zhang, Qiuying Peng, Jun Wang, and Weiming Lu. 2025. STaR-SQL: Self-taught reasoner for text-to-SQL. In *ACL*.
- Binyuan Hui, Ruiying Geng, Qiyu Ren, Binhua Li, Yongbin Li, Jian Sun, Fei Huang, Luo Si, Pengfei Zhu, and Xiaodan Zhu. 2021. Dynamic hybrid relation exploration network for cross-domain context-dependent semantic parsing. In *AAAI*.
- Binyuan Hui, Jian Yang, Zeyu Cui, Jiayi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.
- George Katsogiannis-Meimarakis and Georgia Koutrika. 2023. A survey on deep learning approaches for text-to-SQL. *The VLDB Journal*, 32:905–936.

- Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. 2025. Mcs-sql: Leveraging multiple prompts and multiple-choice selection for text-to-sql generation. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 337–353.
- Boyan Li, Jiayi Zhang, Ju Fan, Yanwei Xu, Chong Chen, Nan Tang, and Yuyu Luo. 2025a. Alpha-sql: Zero-shot text-to-sql using monte carlo tree search. In *Forty-Second International Conference on Machine Learning, ICML 2025*.
- Fei Li and Hosagrahar V Jagadish. 2014. Constructing an interactive natural language interface for relational databases. *VLDB*.
- Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Teying Zhang, Jianjun Chen, Rui Shi, Hong Chen, and Cuiping Li. 2025b. [Omnisql: Synthesizing high-quality text-to-sql data at scale](#). *Proc. VLDB Endow.*, 18(11):4695–4709.
- Jinyang Li, Binyuan Hui, Ge Qu, Jiayi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2023. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36:42330–42357.
- Zhishuai Li, Xiang Wang, Jingjing Zhao, Sun Yang, Guoqing Du, Xiaoru Hu, Bin Zhang, Yuxiao Ye, Ziyue Li, Rui Zhao, et al. 2024. Pet-sql: A prompt-enhanced two-stage text-to-sql framework with cross-consistency. *arXiv*.
- Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let’s verify step by step. In *International Conference on Learning Representations*, volume 2024, pages 39578–39601.
- Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S. Yu. 2023. A comprehensive evaluation of ChatGPT’s zero-shot Text-to-SQL capability. *arXiv*.
- Hanbing Liu, Haoyang Li, Xiaokang Zhang, Ruotong Chen, Haiyong Xu, Tian Tian, Qi Qi, and Jing Zhang. 2025a. Uncovering the impact of chain-of-thought reasoning for direct preference optimization: Lessons from text-to-SQL. In *ACL*.
- Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Nan Tang, and Yuyu Luo. 2025b. 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*.
- Yifu Liu, Yin Zhu, Yingqi Gao, Zhiling Luo, Xiaoxia Li, Xiaorong Shi, Yuntao Hong, Jinyang Gao, Yu Li, Bolin Ding, et al. 2026. Xiyan-sql: A novel multi-generator framework for text-to-sql. *IEEE Transactions on Knowledge and Data Engineering*.
- Peixian Ma, Xialie Zhuang, Chengjin Xu, Xuhui Jiang, Ran Chen, and Jian Guo. 2025. SQL-R1: Training Natural Language to SQL Reasoning Model By Reinforcement Learning. *arXiv*.
- Xutao Mao, Dixuan Zhang, Yifan Li, Jiaming Hou, Rui Li, YunLong Li, Aoze Zheng, Zhiqiang Zhang, Luo Zhewei, Hongying Zan, et al. 2026. Logiccat: A chain-of-thought text-to-sql benchmark for complex reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 29958–29966.
- Linyong Nan, Yilun Zhao, Weijin Zou, Narutatsu Ri, Jaesung Tae, Ellen Zhang, Arman Cohan, and Dragomir Radev. 2023. Enhancing text-to-sql capabilities of large language models: A study on prompt design strategies. In *Findings of the association for computational linguistics: EMNLP 2023*, pages 14935–14956.
- Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates. 2004. Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In *COLING 2004*.
- Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan Arik. 2025a. Chase-sql: Multi-path reasoning and preference optimized candidate selection in text-to-sql. In *ICLR*.
- Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems*, 36:36339–36348.
- Mohammadreza Pourreza and Davood Rafiei. 2024. Dts-sql: Decomposed text-to-sql with small large language models. In *Findings of EMNLP*, pages 8212–8220.
- Mohammadreza Pourreza, Shayan Talaei, Ruoxi Sun, Xingchen Wan, Hailong Li, et al. 2025b. Reasoning-sql: Reinforcement learning with sql tailored partial rewards for reasoning-enhanced text-to-sql. *arXiv*.
- Yang Qin, Chao Chen, Zhihang Fu, Ze Chen, Dezhong Peng, Peng Hu, and Jieping Ye. 2025. Route: Robust multitask tuning and collaboration for text-to-sql. In *The Thirteenth International Conference on Learning Representations*.
- Ge Qu, Jinyang Li, Bowen Li, Bowen Qin, Nan Huo, Chenhao Ma, and Reynold Cheng. 2024. Before generation, align it! a novel and effective strategy for mitigating hallucinations in text-to-sql generation. In *Findings of ACL 2024*.
- Ge Qu, Jinyang Li, Bowen Qin, Xiaolong Li, Nan Huo, Chenhao Ma, and Reynold Cheng. 2025. SHARE: An SLM-based hierarchical action CorREction assistant for text-to-SQL. In *ACL*.

- Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *NeurIPS*, 36:53728–53741.
- Diptikalyan Saha, Avriella Floratou, Karthik Sankaranarayanan, Umar Farooq Minhas, Ashish R Mittal, and Fatma Özcan. 2016. Athena: an ontology-driven system for natural language querying over relational data stores. *VLDB*.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv*.
- Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. Hybridflow: A flexible and efficient rlhf framework. In *EuroSys*.
- Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.
- Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2025. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *ICLR*.
- Yibo Sun, Duyu Tang, Nan Duan, Jianshu Ji, Guihong Cao, Xiaocheng Feng, Bing Qin, Ting Liu, and Ming Zhou. 2018. Semantic parsing with syntax- and table-aware sql generation. *arXiv preprint arXiv:1804.08338*.
- Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2025. Chess: Contextual harnessing for efficient sql synthesis. In *ICML 2025 Workshop on Multi-Agent Systems in the Era of Foundation Models*.
- Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. In *ACL*.
- Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, et al. 2025. Mac-sql: A multi-agent collaborative framework for text-to-sql. In *COLING*, pages 540–557.
- Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2024. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *ICLR*.
- Xiangjin Xie, Guangwei Xu, Lingyan Zhao, and Ruijie Guo. 2025. OpenSearch-SQL: Enhancing Text-to-SQL with Dynamic Few-shot and Consistency Alignment.
- Jiaxi Yang, Binyuan Hui, Min Yang, Jian Yang, Junyang Lin, and Chang Zhou. 2024. Synthesizing text-to-sql data from weak and strong llms. In *ACL*.
- Zhewei Yao, Guoheng Sun, Lukasz Borchmann, Zheyu Shen, Minghang Deng, Bohan Zhai, Hao Zhang, Ang Li, and Yuxiong He. 2025. Arctic-text2sql-r1: Simple rewards, strong reasoning in text-to-sql. *arXiv*.
- Bohan Zhai, Canwen Xu, Yuxiong He, and Zhewei Yao. 2025. [Optimizing reasoning for text-to-SQL with execution feedback](#). In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 19206–19218, Vienna, Austria. Association for Computational Linguistics.
- Bin Zhang, Yuxiao Ye, Guoqing Du, Xiaoru Hu, Zhishuai Li, Sun Yang, Chi Harold Liu, Rui Zhao, Ziyue Li, and Hangyu Mao. 2024. Benchmarking the Text-to-SQL Capability of Large Language Models: A Comprehensive Evaluation. *arXiv*.
- Yuxin Zhang, Meihao Fan, Ju Fan, Mingyang Yi, Yuyu Luo, Jian Tan, and Guoliang Li. 2025. Reward-SQL: Boosting Text-to-SQL via Stepwise Reasoning and Process-Supervised Rewards. *arXiv*.

A Pseudocode

Algorithm 1 presents the pseudocode for the Weighted Consensus Tournament (WCT) framework, detailing the complete procedure for grouping, evaluating, and selecting sampled SQL candidates.

Algorithm 1 Weighted Consensus Tournament (WCT)

```

1: Input: Question  $x \in \mathcal{X}$ , Database  $D \in \mathcal{D}$ ,
   Candidate set  $\mathcal{Y}_x = \{y^{(1)}, \dots, y^{(N)}\}$ , Judg-
   ment model  $\mathcal{R}_\psi$ 
2: Output: Final SQL query  $y^* \in \mathcal{Y}$ 
3: Group candidates into execution-consistent
   clusters:
    $\{\mathcal{C}_1, \dots, \mathcal{C}_K\} \leftarrow \text{Cluster}(\mathcal{Y}_x, D)$ 
4: Initialize scores:  $S[k] \leftarrow 0 \ \forall k \in [1, K]$ 
5: Select proxy SQLs for each cluster:  $\{y^{(k)}\}_{k=1}^K$ 
6: for  $k = 1$  to  $K$  do
7:   for  $j = 1$  to  $K$  do
8:     if  $k = j$  then continue
9:   end if
10:   $e^{(k)} \leftarrow \text{Exec}(y^{(k)}, D)$ ,  $e^{(j)} \leftarrow$ 
    $\text{Exec}(y^{(j)}, D)$ 
11:   $d_{uni} \leftarrow \text{Schema}(y^{(k)}) \cup \text{Schema}(y^{(j)})$ 
12:   $w \leftarrow \mathcal{R}_\psi(x, d_{uni}, y^{(k)}, e^{(k)}, y^{(j)}, e^{(j)})$ 
    $\triangleright w = 1$  if  $y^{(k)} \succ_x y^{(j)}$ 
13:  if  $w = 1$  then
14:     $S[k] \leftarrow S[k] + 1$ 
15:  else
16:     $S[j] \leftarrow S[j] + 1$ 
17:  end if
18: end for
19: end for
20: Compute weighted scores:  $S_w[k] \leftarrow |\mathcal{C}_k| \cdot$ 
    $S[k] \ \forall k \in [1, K]$ 
21:  $\mathcal{C}^* \leftarrow \arg \max_k S_w[k]$ 
22:  $y^* \leftarrow \text{Sample}(\mathcal{C}^*)$ 
23: return  $y^*$ 

```

B Experimental Setup

B.1 Datasets

BIRD contains 9,428, 1,534, and 1,789 text–SQL pairs in the training, development, and test sets, respectively. Since the test set is not publicly available, we follow previous work and evaluate our method on the development set. In addition, given the large number of baseline models considered in our study, unified test-set submission and comparison are infeasible. We utilize the BIRD train

set as the primary data source for both SFT and GRPO training. Following the preference data construction procedure described in Section 4.2, we generate 16 candidate SQL queries for each filtered example and transform these into preference pairs, where each pair is constructed from SQL queries drawn from different execution-consistent sets, representing a preferred and a non-preferred interpretation. For listwise judgement training, following previous work (Liu et al., 2026; Lee et al., 2025), we use the same candidate sets, select one representative SQL from each execution-consistent set to form a list, and provide a preference signal indicating the correct representative. We randomly select 200 pairs and use GPT-4o to generate their corresponding reasoning processes for the reasoning distillation stage. The remaining preference pairs are then used for GRPO training.

B.2 Baselines

We compare our approach against widely used baseline selection strategies, including:

1) **Self-Consistency (SC)**, which selects the final SQL by majority voting over execution-consistent outputs, serving as a standard consistency-based selection strategy in Text-to-SQL; 2) **Double Round-Robin Tournament (DRT)**, which performs exhaustive pairwise comparisons over executable SQL candidates. Each candidate accumulates scores across all pairwise judgments, and the highest-scoring candidate is selected as the final output; 3) **Listwise Judgment (List)**, which performs joint comparison over multiple candidate SQL queries and directly predicts the most likely correct query in a single listwise decision step using an LLM-based judge.

We also compare different judgment models under same selection strategies, including 1) **Prompted SQL Judgment Model (PJudge)** designs a step-by-step prompt to guide a general-purpose language model to judge candidate SQLs without task-specific training. The prompt first instructs the model to interpret the user question, then to analyze each SQL and its execution result, and finally to output a normalized decision label (A / B). 2) **Direct-SFT for SQL Judgment Models (Direct-SFT)** is trained to perform direct binary classification over candidate SQL queries without relying on any explicit reasoning process. *Note that in this paper, we refer to our final reinforced SQL judgment model as **RJudge**.*

Table 8: Comparison between RJudge and GPT-4o under the WCT framework on a 10% stratified subset of the BIRD dev set.

Judgment Model	EX (%)	SA (%)
GPT-4o	65.05	77.06
RJudge	67.28	82.55

B.3 Implementation Details

We use Qwen2.5-Coder-7B-Instruct (Hui et al., 2024) as the backbone SQL judgment model in all experiments to ensure both generality and fair comparison. All evaluations are conducted under both zero-shot and reinforced reasoning settings, enabling a comprehensive assessment of the effectiveness and robustness of different selection strategies.

Training is performed in two stages. In the reasoning distillation stage, we conduct full-parameter fine-tuning for 3 epochs with a learning rate of 5×10^{-6} . In the reinforced reasoning stage, we adopt GRPO training implemented using the VERL framework (Sheng et al., 2025). The policy model is trained for 8 epochs with a learning rate of 1×10^{-6} , a mini-batch size is set to 128. For each prompt, 8 candidate responses are sampled with a temperature of 1.0. The clip ratio is set to 0.2, while KL regularization is disabled. The maximum response length is limited to 2048 tokens. To improve training efficiency, we employ Fully Sharded Data Parallel (FSDP) for memory optimization and use vLLM for efficient rollout generation.

During inference, our pipeline consists of two stages: SQL candidate generation and SQL tournament selection. For SQL generation, we evaluate each generation model independently, including Qwen2.5-Coder-Instruct (7B and 32B) (Hui et al., 2024), XiYanSQL-QwenCoder (7B-2502 and 32B-2412) (Liu et al., 2026), and OmniSQL (7B and 32B) (Li et al., 2025b). For all generation models, temperature sampling with $T = 0.8$ is used to generate SQL candidates. In the tournament selection stage, the judgment model is decoded deterministically, with the decoding temperature set to 0 and a single sample per prompt, ensuring stable and reproducible selection results. The prompts used for PJudge and RJudge are provided in Appendix E.

C Experimental Results

C.1 Comparison with Proprietary LLMs

To assess the effectiveness of RJudge against proprietary LLMs, we compare it with GPT-4o under

the WCT framework. Due to API cost constraints, experiments are conducted on a 10% stratified subset of the BIRD development set, preserving the original distributions of database domains and query difficulty levels. Both models perform selection over SQL candidates generated by Qwen2.5-Coder-7B-Instruct with $N = 8$ samples.

As shown in Table 8, RJudge consistently outperforms GPT-4o, achieving higher execution accuracy and judgment accuracy. Despite its substantially smaller scale, RJudge benefits from task-specific reasoning optimization via GRPO, leading to stronger execution-grounded discrimination of fine-grained semantic differences between SQL candidates.

C.2 Efficiency and Cost Analysis

We further profile the inference cost of JudgeSQL beyond the number of judge calls reported in the main paper. We use Qwen2.5-Coder-7B-Instruct as the SQL generator and RJudge as the judgment model on BIRD dev. Profiling is conducted using $4 \times A100$ -40G GPUs and 32 CPU cores. Under the same sampling budget N , SC and WCT+RJudge operate on exactly the same generated candidates and therefore require the same number of database executions.

Latency. On average, generating one SQL candidate takes approximately 0.11 s, executing one SQL takes 0.31 s, and one RJudge comparison takes approximately 0.02 s. Consequently, the additional judgment overhead introduced by WCT remains small relative to candidate generation and database execution. Under the same sampling budget, WCT+RJudge improves EX over SC by 4.62, 4.75, and 4.82 points at $N = 8, 32, 64$, while adding only 0.08 s, 0.42 s, and 0.97 s of latency per question, respectively.

The comparison across different sampling budgets further illustrates the accuracy–efficiency trade-off. WCT+RJudge with $N = 8$ achieves 67.01 EX, exceeding SC with $N = 64$ by 3.12 points while reducing the average latency from 26.88 s to 3.44 s, corresponding to a $7.8\times$ speedup. The results shown on table 9,

Token Cost. Under the same sampling budget, WCT+RJudge introduces a token overhead of 24.0–38.4% relative to SC while improving EX by 4.62–4.82 points. Most of the total token consumption still comes from SQL generation, accounting for 72.3–80.6% of the total tokens used by

Table 9: Average per-question latency on BIRD dev. All latency values are reported in seconds.

Method	N	EX (%)	Total	Generation	Judge	Exec. Count	Execution
Greedy	1	50.85	0.42	0.11	–	1	0.31
SC	8	62.39	3.36	0.88	–	8	2.48
WCT+RJudge	8	67.01	3.44	0.88	0.08	8	2.48
SC	32	63.89	13.44	3.52	–	32	9.92
WCT+RJudge	32	68.64	13.86	3.52	0.42	32	9.92
SC	64	63.89	26.88	7.04	–	64	19.84
WCT+RJudge	64	68.71	27.85	7.04	0.97	64	19.84

Table 10: Average per-question token consumption on BIRD dev.

Method	N	EX (%)	Total Tokens	Gen. Input	Gen. Output	Judge Input	Judge Output
Greedy	1	50.85	2,504	2,202	302	–	–
SC	8	62.39	20,032	17,616	2,416	–	–
WCT+RJudge	8	67.01	24,839	17,616	2,416	3,040	1,767
SC	32	63.89	80,128	70,464	9,664	–	–
WCT+RJudge	32	68.64	106,946	70,464	9,664	16,960	9,858
SC	64	63.89	160,256	140,928	19,328	–	–
WCT+RJudge	64	68.71	221,735	140,928	19,328	38,880	22,599

WCT+RJudge. An RJudge comparison uses approximately 800 input tokens on average, substantially fewer than the 2,202 input tokens required for generating each SQL candidate, because RJudge only receives the union schema referenced by the two compared SQLs rather than the full database schema.

Reducing the sampling budget can more than compensate for the additional judgment cost. WCT+RJudge with $N = 8$ achieves higher EX than SC with $N = 64$ while using 24.8K versus 160.3K tokens, corresponding to a $6.5\times$ reduction in token consumption. Overall, these results show that WCT+RJudge provides a favorable accuracy–efficiency trade-off: at a fixed sampling budget, it adds modest judgment overhead, while its stronger sample efficiency can substantially reduce the total inference cost required to reach a given accuracy level. The results shown on table 10.

D Case Study

While the quantitative results above demonstrate the superiority of GRPO over the base model, we further present a case study to illustrate how GRPO enhances the reasoning process by producing more accurate and reliable reasoning thought. Specifically, Figure 4 presents the question and data pair for question ID 506, which is annotated as a challenging case in the BIRD development set. Figure 5 presents the inference result for the case. Notably, this example is correctly solved by the RSJ model, whereas both the PSJ model and

the SFT model fail on it. This case study tasked the judge model with selecting the correct SQL query to find “the percentage of card set with Japanese translations that are non-foil only.” The model’s reasoning demonstrated a sophisticated grasp of the problem. It first identified that the core operation must be performed at the set level, not the card level. This allowed it to diagnose the flaw in Candidate A, an unnecessary join with the cards table, which it reasoned would cause a Cartesian product and inflate the result to an invalid 153.9%. Conversely, the model validated Candidate B for its correct use of a subquery to filter and aggregate at the series level, which aligned perfectly with the task’s semantics. This shows the model’s ability to map natural language to database operations and perform logical code review grounded in systematic reasoning.

Case Study: Task description

Question:

For all the set of cards that has Japanese translation, what is the percentage of them are only available in non-foil?

Candidate A SQL:

```
SELECT CAST(SUM(CASE WHEN T2.isNonFoilOnly = 1
                    THEN 1 ELSE 0 END) AS REAL) * 100
/ SUM(T1.language = 'Japanese')
FROM set_translations AS T1
INNER JOIN sets AS T2 ON T1.setCode = T2.code
INNER JOIN cards AS T3 ON T2.code = T3.setCode;
```

Execution Result of A: [[153.8992408557626]]

Candidate B SQL:

```
SELECT SUM(CASE WHEN isNonFoilOnly = 1
                  THEN 1 ELSE 0 END) * 100.0 / COUNT(*)
AS percentage_non_foil
FROM sets
WHERE code IN
      (SELECT setCode FROM set_translations
       WHERE language = 'Japanese');
```

Execution Result of B: [[11.570247933884298]]

The model will be prompt to think step by step and choose the candidates that correctly answer the question

Figure 4: The task description of the example case study.

GRPO model response

Reasoning: To determine the correct SQL query, we need to carefully analyze the given schema and the natural language question. The question asks for the percentage of sets that have a Japanese translation and are only available in non-foil format.

1. We need to join the `sets` table with the `set_translations` table on `setCode` to filter sets that have a Japanese translation.
2. We need to calculate the percentage of these sets that are only available in non-foil format.

Let's analyze the two candidate SQL queries:

Execution Result of B:

[[11.570247933884298]]

Both queries involve joining the `set_translations` table with the `sets` table to filter sets with a Japanese translation and then calculating the percentage of non-foil sets among them.

Candidate B filters sets with a Japanese translation and calculates the percentage of non-foil sets correctly. Candidate A seems to have an issue with the division and filtering logic.

Given the execution results, Candidate B returns a reasonable percentage, while Candidate A returns an unusually high value.

Therefore, the correct query is Candidate B.

Figure 5: The inference process of GRPO model for this case study.

E Prompt Template for RJudge and PJudge Model

You first thinks about the reasoning process in the mind and then provides the user with the answer.

Task Overview:

You are a data science expert. Below, you are provided with a database schema, a natural language question, two candidates SQL and its corresponding execution result. Your task is to understand the schema and choose the correct SQL which answers the natural language question from the two candidates.

Database Engine:

SQLite

Database Schema:

```
{DATABASE_SCHEMA}
```

Question:

```
{EVIDENCE}  
{QUESTION}
```

Here are two candidate SQLs and their execute results:

```
Candidate A:  
{CANDIDATE_A_QUERY}  
  
Execution result of A:  
{CANDIDATE_A_RESULT}  
  
Candidate B:  
{CANDIDATE_B_QUERY}  
  
Execution result of B:  
{CANDIDATE_B_RESULT}
```

Instructions:

Before choosing the final answer, please think through the steps of how to confirm its correctness. You should think through these steps:

1. Understanding the user question requirements: What information is being requested? What are the key columns and filters?
2. Analyzing the Database schema: Which tables and joins are necessary to answer the question? What are the key relationships?
3. Evaluating candidate A and its results: Does it select the correct columns? Does it apply the correct conditions and joins? Does the result match expectations?
4. Evaluating candidate B and its results: Does it select the correct columns? Does it apply the correct conditions and joins? Does the result match expectations?
5. Comparing two candidates: Identify differences between the two queries in terms of structure, logic, and execution result.
6. Determining the Correct answer: Select the query that best satisfies the question requirements, with correct columns, filters, and joins.

Remember:

- The correct SQL should return all of the information asked in the question without any missing or extra information. If the question asks for a specific column, the correct candidate SQL only include that column in the SELECT clause, nothing more, nothing less.
- The empty execution result "[]" does not necessarily mean that the SQL query is incorrect; it might simply indicate that the database does not contain such data.

Output Format:

Show your work in `<think>` `</think>` tags. And return your answer 'A' or 'B' in `<answer>` `</answer>` tags. For example, `<think>reasoning process here` `</think>``<answer>A</answer>` if the candidate A is correct, `<think>reasoning process here` `</think>``<answer>B</answer>` if the candidate B is correct.

Figure 6: The prompt template for PJudge model.

You first thinks about the reasoning process in the mind and then provides the user with the answer.

Task Overview:

You are a data science expert. Below, you are provided with a database schema, a natural language question, two candidates SQL and its corresponding execution result. Your task is to understand the schema and choose the correct SQL which answers the natural language question from the two candidates.

Database Engine:

SQLite

Database Schema:

```
{DATABASE_SCHEMA}
```

This schema describes the database's structure, including tables, columns, primary keys, foreign keys, and any relevant relationships or constraints.

Question:

```
{EVIDENCE} {QUESTION}
```

Here are two candidate SQLs and their execute results:

```
Candidate A:
{CANDIDATE_A_QUERY}
Execution result of A:
{CANDIDATE_A_RESULT}

Candidate B:
{CANDIDATE_B_QUERY}
Execution result of B:
{CANDIDATE_B_RESULT}
```

Instructions:

- The correct SQL should return all of the information asked in the question without any missing

or extra information. If the question asks for a specific column, the correct candidate SQL only include that column in the SELECT clause, nothing more.

- The empty execution result "[]" does not necessarily mean that the SQL query is incorrect; it might simply indicate that the database does not contain such data.
- Before choosing the final answer, please think through the steps of how to confirm its correctness.

Output Format: Show your work in `<think> </think>` tags. And return your answer 'A' or 'B' in `<answer> </answer>` tags. For example, `<think>reasoning process here</think><answer>A</answer>` if the candidate A is correct, `<think>reasoning process here</think><answer>B</answer>` if the candidate B is correct.

Figure 7: The prompt template for RJudge model