SQL-STRUCTURE-AWARE ON-POLICY DISTILLATION FOR TEXT-TO-SQL

Anonymous authors
Paper under double-blind review

ABSTRACT

On-policy distillation (OPD) provides token-level teacher feedback for training Text-to-SQL models. However, uniform weighting allows lengthy reasoning text to dominate the distillation objective and does not account for how tokens jointly express SQL operations, columns, and values. We propose a SQL-structure-aware weighting method for OPD that controls the total weight assigned to final SQL and reasoning steps and adjusts local weights through clause comparisons. The method uses abstract syntax trees to compare student-generated clauses with the reference SQL and other execution-correct samples, accounting for their query context. It then propagates SQL weights to the corresponding reasoning steps through aligned SQL fragments. These weights scale the existing teacher feedback without changing the task reward or adding inference-time procedures. Trained only on BIRD with a Qwen3.5-4B student, our method achieves 64.64% execution accuracy on BIRD-dev, improving over vanilla OPD by 3.58 percentage points. It also improves execution accuracy across five Spider benchmarks by 1.75–2.61 points.

1 INTRODUCTION

Text-to-SQL translates natural language questions into executable database queries. Reinforcement learning with verifiable rewards (RLVR) uses execution results for training, but final outcomes provide limited guidance about individual generation steps. On-policy distillation (OPD) supplies token-level teacher feedback on student-generated responses. Recent applications include Progressive RLVR, which studies compression and generalization through OPD (Zhu et al., 2026), and SERL-SQL, which uses execution-informed teacher scores to selectively reweight GRPO advantages for SQL and tool-action tokens (Liu et al., 2026).

Token-level feedback still leaves open how much weight each part of a response should receive during training. Reasoning text and the final SQL serve different roles. With equal token weights, longer text receives a larger share of the total weight. In the example in Figure 1(a), the response contains 464 reasoning tokens and 14 final SQL tokens. Under equal weighting, these account for 97.1% and 2.9% of the total distillation weight, respectively. The final SQL, which is directly executed, can therefore receive only a small share of the weight. Reasoning text is valuable, but its weight relative to the final SQL should not be determined solely by text length.

Adjusting the weight ratio between reasoning text and the final SQL is not sufficient; weighting must also account for the query decisions within each part. The meaning of a SQL clause depends jointly on its operations, columns, and values, as well as the query context. A local token matching the reference does not imply that the complete clause expresses the correct query decision. Prior work on SQL error correction likewise notes that token-level edits can be ambiguous without context and instead uses clause-level edits to retain more context (Chen et al., 2023). In Figure 1(b), the question asks for the total enrollment across all universities in 2011, but the student generates `SELECT num_students`, omitting `SUM`. Although `SELECT` matches the reference, the clause fails to express the required aggregation. Distillation weighting should therefore incorporate comparisons of the operations, columns, and values in complete clauses alongside token-level teacher feedback. The same issue can arise in the associated reasoning steps: locally matching wording does not guarantee a correct query decision at the step level.

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

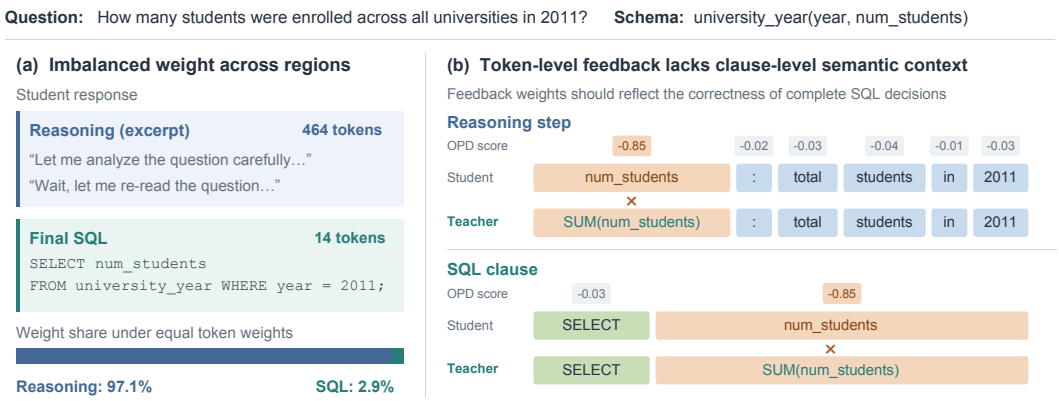


Figure 1: Two motivations for weighting OPD using SQL structure. (a) In this example, equal token weights allocate 97.1% of the total distillation weight to reasoning and only 2.9% to the final SQL. (b) The question asks for the total enrollment across all universities in 2011, but the generated query omits `SUM`. Even when local content such as `SELECT` matches the reference, the complete clause and its associated reasoning step may fail to express the required query decision.

We propose an OPD weighting framework with three components. First, it assigns predefined shares of the total distillation weight to final SQL and alignable reasoning steps. Second, it uses ASTs to compare operations, columns, and values in student-generated clauses against the training reference and other execution-correct samples. Comparisons require matching clause types and dependency contexts and measure reference agreement rather than semantic correctness. Third, a Step–SQL Fragment protocol pairs each natural language explanation with a SQL fragment. When that fragment matches a unique location in the final SQL, its SQL weights guide the distillation weights of the corresponding explanation and fragment.

The framework changes the strength of existing teacher feedback while preserving task rewards and feedback direction. The teacher still scores each token using the full student-generated prefix; no additional local correctness reward is introduced.

Our main contributions are as follows:

- We propose an OPD weighting framework based on SQL structure that controls the total distillation weights assigned to the final SQL and alignable reasoning steps, preventing their relative shares from being directly determined by text length.
- We use ASTs to compare operations, columns, and values in SQL clauses, adjust SQL distillation weights based on these comparisons, and guide weight allocation to the explanations and SQL fragments in corresponding reasoning steps through the Step–SQL Fragment protocol.
- We conduct extensive experiments on BIRD and Spider to validate the generalization ability of the proposed method.

2 Related Work

Text-to-SQL. LLM-based Text-to-SQL methods improve SQL generation through model training and inference-time workflows. CodeS (Li et al., 2024b) uses SQL-focused incremental pre-training, while OmniSQL (Li et al., 2025) trains models on large-scale synthetic Text-to-SQL data. Prompting-based methods improve generation without task-specific parameter updates: C3 (Dong et al., 2023) combines clear prompts, calibration hints, and output consistency, and DIN-SQL (Pourreza & Rafiei, 2023) decomposes generation into subtasks and applies self-correction. SEA-SQL (Li et al., 2024a) combines semantic-enhanced schema information, a trained bias eliminator, and execution-based adjustment. Multi-agent systems, including MAC-SQL (Wang et al., 2023) and SQLFixAgent (Cen et al., 2024), coordinate generation, review, and repair. These approaches im-

prove training data or the generation workflow. Our work studies how SQL structure can guide the allocation of teacher feedback during distillation.

On-policy distillation. OPD trains a student using teacher feedback on sequences sampled from the student itself. This lets the teacher guide the student on the prefixes it actually encounters during generation. Generalized Knowledge Distillation (GKD) (Agarwal et al., 2024) addresses the mismatch between training sequences and student-generated sequences and supports combining distillation with reinforcement learning. MiniLLM (Gu et al., 2024) uses reverse KL divergence and policy-gradient optimization to transfer knowledge from a larger model. These studies establish useful objectives and training procedures for language-model distillation. Our work focuses on weight allocation for structured outputs: we control the total weight assigned to reasoning steps and final SQL, then adjust local weights using SQL clause comparisons.

OPD for Text-to-SQL. Recent work has applied on-policy teacher feedback to Text-to-SQL. Progressive RLVR (Zhu et al., 2026) distills an RLVR-trained teacher into a student with TinyLoRA updates and studies compression-based generalization bounds on tasks including Text-to-SQL. SERL-SQL (Liu et al., 2026) gives a teacher execution feedback to rescore student-generated actions, then uses teacher–student probability differences to reweight GRPO advantages for SQL and tool-action tokens. Our method uses a fixed teacher and applies structural weights only to the OPD loss, leaving the task-reward loss unchanged. It compares SQL clauses with the reference query and other execution-correct samples, and transfers their weights to the corresponding reasoning steps through aligned SQL fragments. This makes the relation between reasoning steps and final SQL part of the distillation weighting process, without adding inference-time procedures.

3 METHOD

We train on student-generated responses and assign a nonnegative weight to each token in the OPD branch. The teacher scoring procedure and task rewards remain unchanged. We first compare SQL clauses to compute weights for positions in the final SQL, then transfer these weights to the corresponding reasoning steps, and finally rescale the two parts to obtain predefined shares of the total weight.

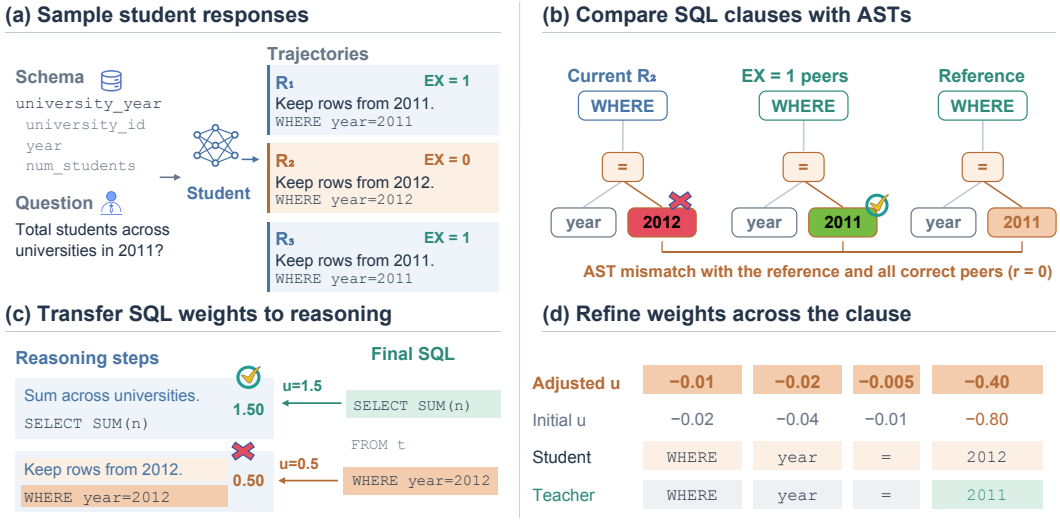


Figure 2: OPD weighting using SQL structure. (a) Sample responses to the same question. (b) Compare clauses against execution-correct peers and the training reference. (c) Transfer clause weights to the corresponding SQL fragments and explanations. (d) Scale existing token-level OPD advantages using the clause weight. The labels Initial u and Adjusted u in (d) denote advantages before and after scaling, not nonnegative weights.

Figure 2 asks for the total number of students across universities in 2011. Responses R_1 and R_3 execute correctly, whereas R_2 incorrectly filters for 2012. For readability, `t` and `n` abbreviate `university_year` and `num_students`. We use this example to explain clause comparison, weight transfer, and feedback scaling, before describing how total weight is allocated between SQL and reasoning steps.

3.1 Training objective

Given an input x containing a natural language question, a database schema, and any available auxiliary information, the student π_θ samples K responses:

$$y^{(i)} \sim \pi_\theta(\cdot \mid x), \qquad i = 1, \ldots, K. \tag{1}$$

Each response contains reasoning text and a final SQL query. Under the Step–SQL Fragment protocol, each reasoning step consists of a natural language explanation and its SQL fragment. Distillation primarily covers the final SQL and reasoning steps that can be reliably aligned to it. Other reasoning text that does not follow the protocol or cannot be aligned does not directly receive this distillation feedback.

We omit the response index i below. A fixed teacher π_ϕ scores the same sampled token y_t conditioned on the student's actual prefix:

$$\ell_{\theta,t} = \log \pi_\theta(y_t \mid x, y_{<t}), \qquad \ell_{\phi,t} = \log \pi_\phi(y_t \mid x, y_{<t}). \tag{2}$$

Following the base OPD implementation, we form a token-level advantage from the clipped log-probability difference and multiply it by our weight w_t:

$$A_t^{\mathrm{OPD}} = -\,\mathrm{sg}[\mathrm{clip}(\ell_{\theta,t} - \ell_{\phi,t}, -c_{\mathrm{KL}}, c_{\mathrm{KL}})]\,, \qquad \widetilde{A}_t^{\mathrm{OPD}} = \mathrm{sg}(w_t) A_t^{\mathrm{OPD}}. \tag{3}$$

Here, $c_{\mathrm{KL}} > 0$ is the clipping threshold and sg denotes stop-gradient. The training objective is

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{task-GRPO}}(\theta) + \lambda_{\mathrm{OPD}} \mathcal{L}_{\text{PG-OPD}}(\theta; \widetilde{A}^{\mathrm{OPD}}). \tag{4}$$

The coefficient λ_{OPD} controls the overall strength of the OPD branch. We retain the task loss and OPD policy-loss form, replacing only the original OPD advantage with its weighted version. Selecting distillation positions does not remove teacher context: even when a reasoning span is excluded from the distillation loss, the teacher still sees the full student prefix when scoring subsequent tokens.

3.2 Comparing SQL clauses

An execution-incorrect SQL query may still contain clauses that agree with a correct query. We use local agreements and disagreements to adjust weights, without interpreting them as complete judgments of semantic correctness.

Which queries to compare. Let $s^{(i)}$ be the final SQL in response i and $s^\star$ the training reference. In addition to the reference, we use other execution-correct SQL queries sampled for the same question:

$$\mathcal{P}_i = \{j \in \{1, \ldots, K\} \setminus \{i\} : \mathrm{EX}(s^{(j)}, s^\star) = 1\}. \tag{5}$$

Here, EX indicates matching execution results. A response never contributes to its own comparison. Other sampled queries are selected by execution correctness alone, not by a combined reward that includes format or length penalties. They can provide alternatives absent from the reference SQL, but must still meet the clause-comparison conditions below.

What to compare. We parse each query into an AST and extract three types of information for each clause c. The first is a normalized representation σ_c that preserves operations, column references, and literal values. We resolve table aliases and column bindings where possible, normalize identifiers, and serialize the AST back into SQL text. This removes resolvable alias differences while preserving the value difference between `year=2011` and `year=2012`. The second representation, η_c, records the clause's structural form: column names in σ_c are replaced with a shared placeholder, and strings and numbers with placeholders of their respective types, while operations and table bindings are retained. The third, δ_c, records the query context on which the clause depends, including `FROM`/`JOIN` relations and other relevant clauses. For example, comparing `SELECT` requires accounting for `GROUP BY`, while comparing `ORDER BY` also considers `WHERE`, `GROUP BY`, and `HAVING`.

How to interpret the comparison. Let $\mathcal{C}(c, s)$ contain the reliably parsed clauses in query s with the same type and dependency context as c. We define

$$r(c,s) = \begin{cases} 1, & \exists c' \in \mathcal{C}(c,s) : \sigma_c = \sigma_{c'}, \\ 0, & \mathcal{C}(c,s) = \{c'\},\ \eta_c = \eta_{c'},\ \sigma_c \neq \sigma_{c'}, \\ \bot, & \text{otherwise.} \end{cases} \tag{6}$$

A value of 1 means that an agreeing clause was found; 0 means that the unique comparable clause has the same structural form but different content; and $\bot$ means unknown. We also return unknown if the candidate clause cannot be parsed or its column bindings cannot be resolved reliably. In Figure 2(b), `WHERE year=2012` in R_2 uses the same column and comparison operator as `WHERE year=2011` in the reference and both correct peers, but a different year. With matching contexts and a unique comparable clause, each comparison yields 0. In contrast, `SELECT n` and `SELECT SUM(n)` have different operation structures and yield unknown.

3.3 Computing SQL weights from comparison results

Let $g_c = r(c, s^\star)$. We initialize from the reference comparison when its result is known, and otherwise use the neutral value $1/2$:

$$q_c^{(0)} = \begin{cases} g_c, & g_c \neq \bot, \\ 1/2, & g_c = \bot. \end{cases} \tag{7}$$

For clauses that permit comparisons with other sampled queries, collect the known results as $b_1, \ldots, b_{n_c} \in \{0, 1\}$. We combine them with the initial value:

$$q_c = \frac{\kappa q_c^{(0)} + \sum_{j=1}^{n_c} b_j}{\kappa + n_c}, \qquad h_c = \begin{cases} 1, & g_c \neq \bot, \\ n_c/(n_c + \kappa), & g_c = \bot. \end{cases} \tag{8}$$

Here, $\kappa > 0$ controls the weight of the initial value, q_c summarizes the comparisons, and h_c controls how strongly they affect the weight. If the reference comparison is unknown, fewer known comparisons lead to weaker adjustments. We set $n_c = 0$ for reference-only clauses. If the candidate clause itself cannot be handled reliably, we directly set $q_c = 1/2$ and $h_c = 0$. These quantities are not calibrated probabilities of correctness.

The initial clause weight is

$$u_c = \mathrm{clip}(1 + \beta h_c (2q_c - 1), u_{\min}, u_{\max}), \tag{9}$$

where $\beta \geq 0$ controls adjustment strength and $0 < u_{\min} \leq 1 \leq u_{\max}$. Agreement favors an upward adjustment from the neutral weight 1, disagreement favors a downward adjustment, and missing comparisons leave the weight at 1. These descriptions refer to weights before the subsequent normalization.

For the filter in Figure 2(b), all known comparisons yield 0, giving $q_c = 0$ and $h_c = 1$. Taking $\beta = 0.5$ and clipping bounds that include 0.5 and 1.5 gives weight 0.5; an aggregation clause with all known comparisons equal to 1 receives 1.5. These values illustrate the mechanism rather than specify experimental hyperparameters.

We assign u_c to the clause's character span in the original SQL. Overlapping positions take the weight of the more deeply nested clause, and uncovered characters retain weight 1. A token's initial weight u_t is the average weight of the characters it covers. Thus, weights are based on complete-clause comparisons; we do not compute separate correctness scores for keywords, columns, or values.

3.4 Transferring SQL weights to reasoning steps

We first identify which part of the final SQL a reasoning step describes, then transfer the corresponding weights. In Figure 2(c), "Sum across universities" is paired with `SELECT SUM(n)`, and "Keep rows from 2012" with `WHERE year=2012`. These fragments locate the aggregation and filter in the final query.

For a successfully aligned step k, let $\mathcal{A}_k$ contain the final-SQL tokens covered by its fragment. We compute

$$\bar{u}_k = \frac{1}{|\mathcal{A}_k|} \sum_{t \in \mathcal{A}_k} u_t, \qquad u_k^{\mathrm{NL}} = 1 + \gamma(\bar{u}_k - 1), \quad 0 \le \gamma \le 1. \tag{10}$$

All tokens in the natural language explanation receive u_k^{NL}. A smaller γ keeps this weight closer to 1, reducing the influence of SQL comparisons on the explanation. The step's SQL fragment instead inherits character weights from the corresponding positions in the final SQL, which are then mapped to the fragment's own tokens. This preserves weight differences within the fragment.

Figure 2(c) illustrates direct transfer with $\gamma = 1$: the aggregation and year-filter explanations receive weights 1.5 and 0.5, respectively. In general, their weights follow the equation above.

Transfer covers only the explanation body and SQL fragment, excluding step headings, code fences, and other reasoning text. Alignment or agreement with a reference does not verify the preceding explanation; it provides only limited guidance for its weight.

Figure 2(d) applies weight 0.5 to every position in the filter. For example, the advantage for `2012` changes from -0.80 to -0.40; the other three values are also halved. A shared clause weight preserves differences in the original teacher feedback while reducing its negative magnitude without reversing its sign. The teacher row shows `2011` to contrast the intended year; during training, the teacher still scores the student-generated `2012`. This panel illustrates local scaling only; the final training weights also undergo the normalization below.

3.5 Controlling total weights for SQL and reasoning steps

The preceding computations determine relative weights across positions. We apply two normalization steps to control the total weights assigned to the final SQL and reasoning steps. First, for the set $\mathcal{E}$ of valid tokens marked for local weighting, we compute

$$\mu = \frac{1}{|\mathcal{E}|} \sum_{t \in \mathcal{E}} u_t, \qquad v_t = \begin{cases} u_t/\mu, & t \in \mathcal{E}, \\ 1, & t \text{ valid and } t \notin \mathcal{E}, \\ 0, & t \text{ invalid.} \end{cases} \tag{11}$$

If $\mathcal{E}$ is empty, all valid tokens receive $v_t = 1$. Valid positions are determined by the response mask used for training.

Let $\mathcal{Q}$ contain final-SQL tokens and $\mathcal{R}$ contain the explanation and SQL-fragment tokens of successfully aligned reasoning steps. The two sets are disjoint. Let B be the number of valid tokens in the full response and $\rho \in [0, 1]$ the share assigned to the final SQL. When both parts are nonempty and reliably located, the final weights are

$$w_t = \begin{cases} \rho B \dfrac{v_t}{\sum_{j \in \mathcal{Q}} v_j}, & t \in \mathcal{Q}, \\ (1-\rho) B \dfrac{v_t}{\sum_{j \in \mathcal{R}} v_j}, & t \in \mathcal{R}, \\ 0, & \text{otherwise.} \end{cases} \tag{12}$$

The total weights are therefore ρB for the final SQL and $(1-\rho)B$ for reasoning steps. The parameter ρ determines the share of each part, while clause comparisons determine how that share is distributed within it. This controls weight sums, not actual gradient magnitudes. Since B still depends on the full response length, the method does not remove all length effects.

If no reasoning step can be aligned, the full weight is assigned to the final SQL: $w_t = B v_t / \sum_{j \in \mathcal{Q}} v_j$ for $t \in \mathcal{Q}$, and 0 elsewhere. If AST comparison is unavailable but SQL positions remain identifiable, local weights remain neutral and total-weight allocation still applies. If SQL boundaries or character-to-token mappings cannot be determined reliably, we revert to the original valid-response mask while retaining any local weights already computed.

Finally, we use w_t in Equation equation 3. Nonnegative weights do not reverse the sign of the original advantage. Agreement with a reference therefore does not necessarily increase a token's probability, and downweighting a disagreeing clause may also weaken negative corrective feedback

from the teacher. We adjust the strength of existing teacher feedback rather than introduce a local correctness reward. Comparisons and weight computation are used only during training. At inference, the student follows the same output protocol without teacher scoring, reference SQL, or other sampled queries, and without adding SQL repair or candidate selection.

Table 1: Execution accuracy (EX, %) on six Text-to-SQL benchmarks.

		BIRD	Spider		Spider variants		
Method	Model/size	Dev	Dev	Test	Syn	DK	Real
Methods based on closed-source models							
MAC-SQL	GPT-4	59.40	86.70	82.80	–	–	–
CHASE-SQL	Gemini-1.5	73.00	–	87.60	–	–	–
Claude Sonnet-3.7	Closed	60.10	78.90	83.20	–	–	–
Methods based on open-source models							
CodeS	15B	58.50	79.40	–	69.40	–	75.60
OmniSQL	32B	64.50	–	**87.60**	–	76.10	–
SENSE	13B	55.50	84.10	86.60	70.20	–	76.60
ROUTE	14B	60.80	80.90	87.10	–	–	–
SQL-o1	8B	63.40	79.60	85.40	77.60	–	82.70
Controlled comparison: Qwen3.5-4B student							
vanilla OPD	4B	61.06	82.79	81.46	81.33	76.64	81.10
Ours	4B	**64.64**	**84.82**	83.56	**83.08**	**79.25**	**83.27**
Gain over vanilla OPD		+3.58	+2.03	+2.10	+1.75	+2.61	+2.17

Bold denotes the highest EX among open-source models and our method in each column; closed-source results are excluded from this comparison. Shading identifies our method. Gains are in percentage points. A dash denotes an unavailable result. Real abbreviates Spider-Realistic.

4 Experiments

4.1 Datasets

We train exclusively on the BIRD training split, without using any other training datasets. We evaluate on BIRD-dev, Spider-dev, Spider-test, Spider-Syn, Spider-DK, and Spider-Realistic (Spider-Real) to assess generalization across benchmarks and robustness to variations in question wording and domain-knowledge requirements. Detailed dataset descriptions and references are provided in Appendix A.

4.2 Evaluation Metric

We use execution accuracy (EX) as the sole evaluation metric, computed with the official BIRD evaluation suite for BIRD-dev and the official Spider evaluation suite for Spider-dev, Spider-test, and the three Spider variants. The definition and computation of EX are detailed in Appendix B.

4.3 Baselines

We compare with methods based on closed-source models (MAC-SQL, CHASE-SQL, and Claude Sonnet-3.7) and open-source models (CodeS, OmniSQL, SENSE, ROUTE, and SQL-o1). vanilla OPD provides a controlled comparison using the same student, teacher, and training setup, with equal weights for all valid response tokens. Descriptions of the baselines and their model configurations are provided in Appendix D.

4.4 Implementation Details

We implement GRPO with OPD in verl, using Qwen3.5-4B as the main student and Qwen3.8-27B as the frozen teacher. We perform full-parameter training for 800 updates. Sampling and teacher scoring use vLLM, while student updates use FSDP2. Model specifications and reproducibility settings are provided in Appendix C.

4.5 Main Results

Table 1 presents the main results. Our method outperforms vanilla OPD on all six datasets. BIRD-dev EX increases from 61.06% to 64.64%, a gain of 3.58 percentage points. The gains on Spider-dev and Spider-test are 2.03 and 2.10 points, respectively. On Spider-Syn, Spider-DK, and Spider-Real, the gains are 1.75, 2.61, and 2.17 points. Since training uses only BIRD, the improvements on the Spider benchmarks show that the benefit also extends to other datasets and question variants.

With a 4B student, our method reaches 64.64% on BIRD-dev, compared with 64.50% for OmniSQL-32B and 63.40% for SQL-o1-8B. It also has the highest listed scores on the three Spider variants. However, its Spider-test score remains below those of several larger models and CHASE-SQL. Overall, our method achieves competitive results with a smaller student and consistently outperforms vanilla OPD across datasets.

4.6 Ablation Study

We progressively enable four components on BIRD-dev (Table 2): region weighting, which controls the total weights assigned to final SQL and reasoning steps; within-region weighting, which varies token weights within each region; AST-based clause comparison; and weight propagation from SQL to aligned reasoning steps. A check mark denotes an enabled component and a dash a disabled component. A0 is vanilla OPD, and A4 is the complete method.

Table 2: Progressively enabling components on BIRD-dev.

	Enabled components				BIRD-dev	
Config.	Region weighting	Within-region weighting	AST clause comparison	Weight propagation	EX (%)	Δ
A0	–	–	–	–	61.06	–
A1	✓	–	–	–	62.56	+1.50
A2	✓	✓	–	–	62.36	−0.20
A3	✓	✓	✓	–	63.34	+0.98
A4	✓	✓	✓	✓	**64.64**	+1.30

✓/– denotes enabled/disabled. A0 is vanilla OPD; A4 is the complete method. Δ is the change from the preceding row in percentage points.

Region weighting raises EX by 1.50 points, from 61.06% to 62.56%, supporting explicit control over the weight allocated to final SQL and reasoning steps. Adding within-region weighting lowers EX by 0.20 points, so this addition alone does not improve the preceding configuration. Adding AST-based clause comparison then raises EX to 63.34% (+0.98 points), and propagating SQL weights to reasoning steps increases it to 64.64% (+1.30 points). The complete method improves over vanilla OPD by 3.58 points and over region weighting alone by 2.08 points. These comparisons support the benefit of clause comparison and step-level propagation in the tested sequence; they do not isolate each component's contribution across all possible combinations.

4.7 Parameter Sensitivity

We examine the clause-weight adjustment coefficient $\beta \in \{0.25, 0.50, 0.75\}$ and the final-SQL weight share $\rho \in \{0.50, 0.65, 0.75, 0.90\}$. The former controls how strongly clause comparisons adjust local weights, while the latter controls the fraction of total distillation weight assigned to final

SQL. To reduce training cost, all runs in this analysis use 500 updates, with BIRD-dev EX recorded every 100 updates. The main experiments retain their 800-update budget.

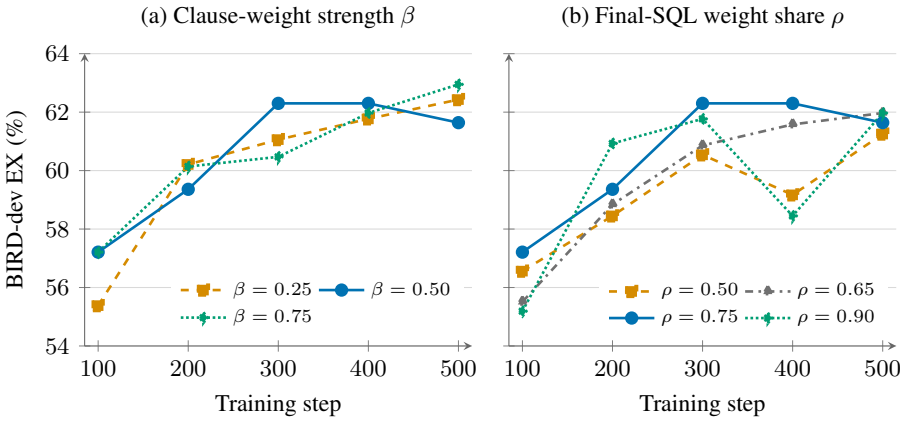


Figure 3: Sensitivity to the clause-weight adjustment coefficient β (left) and final-SQL weight share ρ (right). Both panels report BIRD-dev EX (%) within a common 500-update budget. Blue solid lines denote the parameter values used in the main experiments.

At update 500, EX ranges from 61.64% to 62.95% across β values, with $\beta = 0.75$ performing best (Figure 3). However, $\beta = 0.50$ performs better at updates 300 and 400, showing that the ranking changes during training. Across ρ values, final EX ranges from 61.25% to 61.97%, a spread of 0.72 percentage points. Intermediate fluctuations are larger: $\rho = 0.90$ drops from 61.77% at update 300 to 58.45% at update 400 before recovering to 61.97%. Thus, within the tested range, the final-SQL weight share has a limited effect on EX at update 500, although it affects performance during training.

4.8 Effectiveness Across Student Model Sizes

We additionally evaluate Qwen3.5-9B using the same Qwen3.8-27B teacher. Our method improves BIRD-dev EX over vanilla OPD from 61.06% to 64.64% for the 4B student and from 61.64% to 65.04% for the 9B student. The similar gains of 3.58 and 3.40 percentage points support applicability at both tested scales. Both results also exceed the teacher's 63.08%. Appendix E provides the full comparison with initial models, vanilla OPD, and the teacher.

5 Conclusion and Future Work

We proposed a SQL-structure-aware weighting method for OPD that controls weight allocation between final SQL and reasoning steps and improves distillation through clause comparison and weight propagation. Experiments on BIRD and Spider show consistent gains in execution accuracy over vanilla OPD. However, AST-based matching has limited coverage of diverse SQL expressions and their semantic equivalences, so clause matches do not fully determine semantic correctness. Future work will explore more reliable ways to assess clause correctness across a broader range of SQL semantics, providing a more accurate basis for distillation weighting.

AI USE STATEMENT

Generative AI tools, including ChatGPT and Codex, assisted with manuscript drafting and editing, Chinese-to-English translation, literature search and summarization, interpretation of experimental results, figure preparation, and LaTeX code and formatting. The experimental measurements presented in this manuscript were supplied by the authors. The authors are responsible for the final text, claims, references, and artifacts.

ETHICS STATEMENT

This work studies Text-to-SQL using existing BIRD and Spider benchmarks. Incorrect generated queries can produce misleading answers or expose sensitive database contents. Execution accuracy on these benchmarks does not establish correctness or safety for deployment. Practical applications should enforce database access controls, read-only execution, and appropriate validation before query results inform decisions. AST-based clause comparisons provide limited training guidance and should not be treated as a guarantee of semantic correctness.

REPRODUCIBILITY STATEMENT

Section 3 specifies the training objective, clause comparisons, weight propagation, and normalization. Section 4 describes the evaluation setup and reports the main results and ablations. Appendices A and B describe the datasets and evaluation metric; Appendix C provides model and framework choices and training hyperparameters; and Appendix D details the comparison methods. Parameter sensitivity, including the full learning curves, is reported in Section 4.7. The student-size comparison is summarized in Section 4.8, with the complete comparison in Appendix E.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2306.13649.

Jipeng Cen, Jiaxin Liu, Zhixu Li, and Jingjing Wang. SQLFixAgent: Towards semantic-accurate Text-to-SQL parsing via consistency-enhanced multi-agent collaboration. *arXiv preprint arXiv:2406.13408*, 2024. URL https://arxiv.org/abs/2406.13408.

Ziru Chen, Shijie Chen, Michael White, Raymond Mooney, Ali Payani, Jayanth Srinivasa, Yu Su, and Huan Sun. Text-to-SQL error correction with language models of code. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1359–1372. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-short.117. URL https://aclanthology.org/2023.acl-short.117/.

Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. Structure-grounded pretraining for Text-to-SQL. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2021. URL https://aclanthology.org/2021.naacl-main.105/.

Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Lu Chen, Jinshu Lin, and Dongfang Lou. C3: Zero-shot Text-to-SQL with ChatGPT. *arXiv preprint arXiv:2307.07306*, 2023. URL https://arxiv.org/abs/2307.07306.

Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R. Woodward, Jinxia Xie, and Pengsheng Huang. Towards robustness of Text-to-SQL models against synonym substitution. In *Proceedings of ACL-IJCNLP*, 2021a. URL https://aclanthology.org/2021.acl-long.195/.

Yujian Gan, Xinyun Chen, and Matthew Purver. Exploring underexplored limitations of cross-domain Text-to-SQL generalization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021b. URL https://aclanthology.org/2021.emnlp-main.702/.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. MiniLLM: Knowledge distillation of large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2306.08543.

Chaofan Li, Yingxia Shao, Yawen Li, and Zheng Liu. SEA-SQL: Semantic-enhanced Text-to-SQL with adaptive refinement. *arXiv preprint arXiv:2408.04919*, 2024a. URL https://arxiv.org/abs/2408.04919.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. CodeS: Towards building open-source language models for Text-to-SQL. *arXiv preprint arXiv:2402.16347*, 2024b. URL https://arxiv.org/abs/2402.16347.

Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, Hong Chen, and Cuiping Li. OmniSQL: Synthesizing high-quality Text-to-SQL data at scale. *arXiv preprint arXiv:2503.02240*, 2025. URL https://arxiv.org/abs/2503.02240.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C. 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*, 2023. URL https://arxiv.org/abs/2305.03111.

Tao Liu, Tao Feng, Xiangheng Li, Jinwang Song, Yifan Li, Xiaoqing Cheng, Dixuan Zhang, Siquan Li, Lin Lan, Hongying Zan, Kunli Zhang, and Chao Wu. SERL-SQL: Selective hindsight distillation for Text-to-SQL reinforcement agentic learning. *arXiv preprint arXiv:2608.00485*, 2026. URL https://arxiv.org/abs/2608.00485.

Shuai Lyu, Haoran Luo, Zhonghong Ou, Yifan Zhu, Xiaoran Shang, Yang Qin, and Meina Song. SQL-o1: A self-reward heuristic dynamic search method for Text-to-SQL. *arXiv preprint arXiv:2502.11741*, 2025. URL https://arxiv.org/abs/2502.11741.

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*, 2023. URL https://arxiv.org/abs/2304.11015.

Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O. Arik. CHASE-SQL: Multi-path reasoning and preference optimized candidate selection in Text-to-SQL. *arXiv preprint arXiv:2410.01943*, 2024. URL https://arxiv.org/abs/2410.01943.

Yang Qin, Chao Chen, Zhihang Fu, Ze Chen, Dezhong Peng, Peng Hu, and Jieping Ye. ROUTE: Robust multitask tuning and collaboration for Text-to-SQL. *arXiv preprint arXiv:2412.10138*, 2024. URL https://arxiv.org/abs/2412.10138.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. MAC-SQL: A multi-agent collaborative framework for Text-to-SQL. *arXiv preprint arXiv:2312.11242*, 2023. URL https://arxiv.org/abs/2312.11242.

Jiaxi Yang, Binyuan Hui, Min Yang, Jian Yang, Junyang Lin, and Chang Zhou. Synthesizing Text-to-SQL data from weak and strong LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7864–7875, 2024. URL https://aclanthology.org/2024.acl-long.425/.

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*, 2018. URL https://aclanthology.org/D18-1425/.

Yuxuan Zhu, Rohan Alur, and Daniel Kang. Non-vacuous generalization bounds for reinforcement learning with verifiable rewards. *arXiv preprint arXiv:2607.14506*, 2026. URL https://arxiv.org/abs/2607.14506.

A Dataset Details

BIRD. BIRD (Li et al., 2023) is a cross-domain Text-to-SQL benchmark that emphasizes understanding database contents, including challenges posed by noisy values and external knowledge. We use its training split as the sole source of training data and evaluate on BIRD-dev.

Spider. Spider (Yu et al., 2018) evaluates complex SQL generation across databases, with database schemas separated across its standard training and evaluation splits. We evaluate on Spider-dev and Spider-test to assess transfer from BIRD to another Text-to-SQL benchmark. The Spider training split is not used.

Spider-Syn. Spider-Syn (Gan et al., 2021a) replaces schema-related words in Spider questions with manually selected synonyms. It tests whether a model can interpret alternative wording rather than rely on direct word matches between questions and database schemas. We use it for evaluation only.

Spider-DK. Spider-DK (Gan et al., 2021b) selects questions from Spider and modifies some of them to require domain knowledge, covering five knowledge categories. We use it for evaluation only, to examine how models handle questions whose interpretation requires such knowledge.

Spider-Realistic (Spider-Real). Spider-Realistic (Deng et al., 2021) is derived from Spider-dev by removing explicit mentions of column names from questions. It evaluates whether a model can identify the intended columns when users do not directly name them. We use Spider-Real as an abbreviation and include this dataset only in evaluation.

B Execution Accuracy

Execution accuracy (EX) measures the proportion of predictions whose execution results match those of the reference SQL under the corresponding official evaluator. For an evaluation set with N examples, let $e_i \in \{0, 1\}$ be the execution-correctness outcome returned by that evaluator for example i. We report

$$\mathrm{EX} = \frac{100}{N} \sum_{i=1}^{N} e_i \quad (\%). \tag{13}$$

A prediction receives $e_i = 1$ when it executes successfully and its result matches the reference under the evaluator's comparison rules; otherwise it receives 0. Higher EX indicates better execution accuracy. The metric compares execution results rather than requiring identical SQL text, so queries with different surface forms can receive the same score.

We use the official BIRD evaluation suite for BIRD-dev (Li et al., 2023) and the official Spider evaluation suite for Spider-dev, Spider-test, Spider-Syn, Spider-DK, and Spider-Real (Yu et al., 2018). Each suite determines execution correctness according to its own implementation, including result comparison and execution-failure handling. EX is computed separately for each evaluation set. We report only EX, without combining it with SQL text matching or execution-efficiency scores.

C Implementation Details

Models and framework. We implement GRPO with OPD in verl. The main student is Qwen3.5-4B, initialized from its original weights and trained with full-parameter updates without LoRA. The teacher is Qwen3.8-27B, whose parameters remain frozen during training. Both models have thinking enabled. Student rollouts and teacher scoring use vLLM; student updates use FSDP2 with bfloat16 precision and gradient checkpointing.

Training resources. The 4B run uses eight GPUs: four for student rollout and training and four for teacher inference. Student rollout and teacher tensor-parallel sizes are both 1; teacher GPU memory utilization is 0.8.

Optimization and sampling. We use AdamW with learning rate 10^{-6}, moment coefficients $(0.9, 0.999)$, weight decay 0.01, and gradient clipping at 1.0. The learning rate warms up over the first 3% of updates and then remains constant. Each update samples 24 questions and four responses per question, yielding 96 responses. The PPO mini-batch size is 24; the 4B per-GPU micro-batch size is 4. Training sampling uses temperature 0.7 and top-$p = 0.95$. Prompt and response limits are 8,192 and 4,096 tokens, respectively. The random seed is 42.

Training consists of 800 parameter updates. Validation uses greedy decoding with temperature 0, one response per question, and a 4,096-token response limit.

Loss and reward settings. GRPO advantages are normalized by the within-group standard deviation. The task and OPD policy losses are added with OPD coefficient 1. The policy-gradient k_1 OPD advantage is the negative, stop-gradient student–teacher log-probability difference in Equation equation 3, with clipping threshold $c_{\mathrm{KL}} = 10$, followed by token weighting. The PPO clip ratio is 0.2. Reference KL regularization and KL rewards are disabled. SQL reward computation uses read-only SQLite with a 30-second timeout. The binary execution reward is combined with a structure penalty $-0.2(1 - q_{\mathrm{fmt}})$, where $q_{\mathrm{fmt}} \in [0, 1]$ measures conformity to consecutive step numbering, one fragment per step, and fragment correspondence to the final SQL. DAPO-style overlength penalties use a 1,024-token buffer, coefficient 0.2, and response limit 4,096; the penalty starts above 3,072 tokens.

OPD weighting. Weighting applies only to the OPD branch. Final SQL receives 75% of the total weight and aligned reasoning steps receive 25% ($\rho = 0.75$). Keyword, value, and identifier weighting, clause comparisons, and step transfer are enabled. Keyword importance uses document-frequency IDF computed from training reference SQL, with strength $\alpha = 2$. Clause comparisons use the reference SQL and other execution-correct responses to the same question, with $\kappa = 2$ and $\beta = 0.5$. Step transfer uses $\gamma = 0.5$, and raw local weights are clipped to $[0.5, 2.5]$.

D Baseline Descriptions

We include systems based on closed-source models, open-source Text-to-SQL models, and vanilla OPD. The first two groups provide performance references across model sizes and method designs. vanilla OPD isolates the choice of distillation weights under the same training setup.

D.1 Methods based on closed-source models

MAC-SQL. MAC-SQL (Wang et al., 2023) coordinates multiple agents to select relevant database information, decompose the question and generate SQL, and refine queries using execution feedback. Our main table includes its GPT-4-based version.

CHASE-SQL. CHASE-SQL (Pourreza et al., 2024) generates SQL candidates through question decomposition, reasoning based on query execution plans, and examples synthesized for the current question. A trained selection model compares candidates in pairs to choose the final query. Our main table includes its Gemini-1.5-based version.

Claude Sonnet-3.7. Claude Sonnet-3.7 is a closed-source general-purpose language model, included as a performance reference for general-purpose models. We list it separately from methods such as MAC-SQL and CHASE-SQL that design a dedicated SQL generation workflow.

D.2 Methods based on open-source models

CodeS. CodeS (Li et al., 2024b) performs incremental pretraining on SQL-focused corpora and combines schema linking, prompt construction, and data augmentation to improve SQL generation. We compare with its 15B version.

OmniSQL. OmniSQL (Li et al., 2025) trains SQL generation models using large-scale, diverse synthetic databases and Text-to-SQL examples. Its synthetic data include questions, SQL queries, and reasoning traces. We compare with its 32B version.

SENSE. SENSE (Yang et al., 2024) combines data synthesized by strong models with errors produced by weaker models, using instruction tuning and preference learning to improve SQL generation. Strong-model data broaden task coverage, while weaker-model errors provide signals for preference training. We compare with its 13B version.

ROUTE. ROUTE (Qin et al., 2024) combines SQL generation, schema linking, error correction, and query continuation in multitask fine-tuning. At inference, it coordinates these tasks to improve the final query. We compare with its 14B version.

SQL-o1. SQL-o1 (Lyu et al., 2025) combines self-reward signals with Monte Carlo tree search to explore SQL generation paths and uses schema-aware training data. We compare with its 8B version.

D.3 VANILLA OPD

vanilla OPD uses the same Qwen3.5-4B student, Qwen3.8-27B teacher, training data, prompts, task rewards, optimizer, and 800-update budget as our method. The teacher scores student-generated response tokens, and training jointly optimizes the task and OPD losses. The baseline assigns equal weights to all valid response tokens and disables region allocation, within-region weighting, AST-based clause comparison, and weight propagation to reasoning steps. This comparison therefore evaluates our weighting method under the same training setup.

E STUDENT MODEL SIZE COMPARISONS

To examine applicability across student model sizes, we additionally evaluate Qwen3.5-9B alongside Qwen3.5-4B. Figure 4 compares BIRD-dev EX before training, after vanilla OPD, and with our method, using the fixed Qwen3.8-27B teacher as a reference. The initial model is evaluated before task-specific training in our experiments.

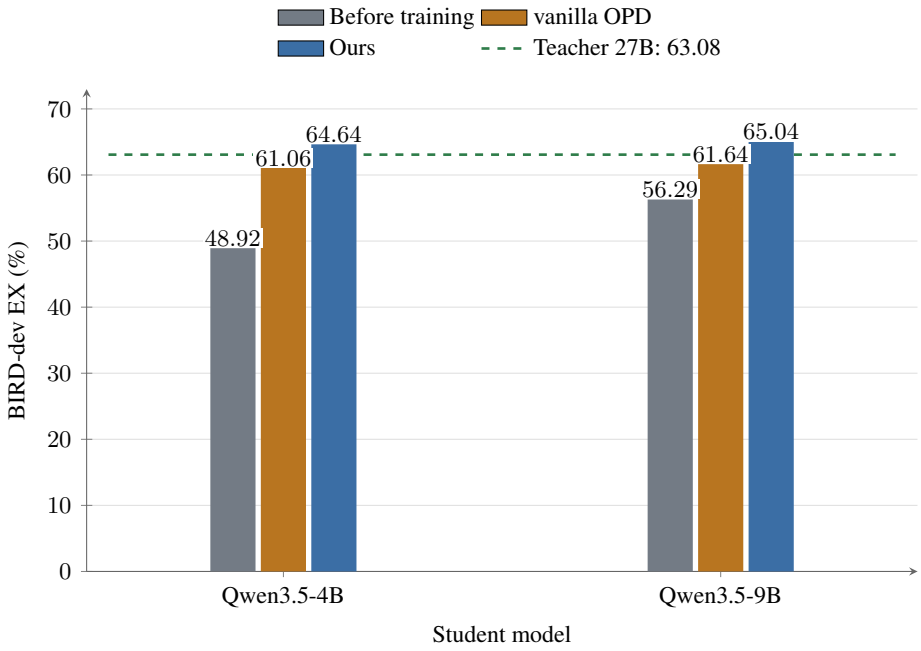


Figure 4: Execution accuracy (EX, %) on BIRD-dev across student model sizes. Bars show the initial model, vanilla OPD, and our method. The dashed line denotes the same 27B teacher (63.08%). The 4B results match Table 1.

For the 4B student, vanilla OPD and our method achieve 61.06% and 64.64%, respectively, a gain of 3.58 percentage points. For the 9B student, the corresponding scores are 61.64% and 65.04%, a

gain of 3.40 points. Our method outperforms both the initial model and vanilla OPD at both sizes, with similar gains over vanilla OPD. These results support the applicability of the proposed weighting method to both the 4B and 9B students evaluated here. Furthermore, our method exceeds the teacher's EX of 63.08% at both student sizes, whereas vanilla OPD remains below it. This suggests that, under the current experimental setup, our method uses teacher feedback more effectively to improve task performance.