HUMAN-LEVEL TEXT-TO-SQL VIA RL WITH VERIFIED DATA AND TASK-SPECIFIC REWARDS

Anonymous authors
Paper under double-blind review

ABSTRACT

Despite increasingly complex multi-stage inference pipelines, existing text-to-SQL systems are still limited by the SQL reasoning capability of their underlying large language model (LLM). These systems still underperform human experts by over 10% on benchmarks. In this paper, we show that reinforcement learning with verified training data and task-specific rewards enables a single fine-tuned LLM to achieve human-level performance. We introduce BIRD-Platinum, a dataset of 2.5k instances sampled from BIRD Train and curated through multiple rounds of expert verification. We corrected annotation errors in 61% of the instances. We then diagnose two limitations of standard execution-based outcome rewards. First, they can reward semantically incorrect queries that coincidentally return the correct result. Second, they provide no direct incentive to use provided external knowledge. To address these limitations, we propose REVISQL, which combines execution-based grading with bounded SQL equivalence verification and process rewards that encourage external-knowledge use. We show that fine-tuning Kimi-K2.6 on BIRD-Platinum with REVISQL matches the 92.96% human performance on Arcwise-Plat, the expert-verified variants of BIRD. Our model outperforms open-source pipelines by 9–22% and surpasses GPT-5.6 Sol Ultra and Claude Fable 5 by 3–8% at 12–15% of their inference cost.

1 INTRODUCTION

Translating natural language questions to SQL queries (text-to-SQL) is a foundational capability for democratizing data access (Androutsopoulos et al., 1995). With their strong reasoning and coding capabilities, LLMs have become central to modern text-to-SQL systems (Liu et al., 2025b). However, accurate text-to-SQL requires models to connect natural language questions to correct schema elements, apply domain knowledge, and convert the intended semantics to SQL queries. Errors in any of these steps can produce incorrect final answers (Shi et al., 2025).

To improve the accuracy of individual LLMs, prior work has built increasingly complex multi-stage pipelines around LLMs. These pipelines combine task decomposition, schema linking, in-context learning, and model-based candidate selection to address errors in SQL generation (Wang et al., 2025; Shkapenyuk et al., 2025; Pourreza et al.; Liu et al., 2025d; Sheng & Xu, 2025a; Pourreza et al., 2025; Dönder et al., 2025; Xie et al., 2025; Qu et al., 2025; Talaei et al., 2024; Sheng & Xu, 2025b; Li et al.). However, query generation, refinement, and selection remain limited by the capability of the LLM to perform correct reasoning on databases and SQL queries. Therefore, misunderstandings of schema relationships or domain knowledge can persist across stages (Cemri et al., 2025; Wang et al., 2025). These limitations motivate us to directly improve LLMs through training.

We show that fine-tuning an LLM using reinforcement learning (RL) with verified data and task-specific rewards can achieve human-level text-to-SQL performance, without a complex inference pipeline. We propose two ingredients: a verification pipeline that produces clean training data (§ 3) and reward shaping methods tailored to text-to-SQL problems (§ 4). Our experiments show that verified data is a dominant factor that brings our fine-tuned model within 3–4% of human parity, while task-specific reward shaping closes the residual gap by providing accurate learning signals.

BIRD-Platinum. While prior work has reported pervasive annotation errors in existing text-to-SQL datasets (Jin et al., 2026; Wretblad et al., 2024; Liu et al., 2025c), correcting these errors for RL training remains underexplored. To bridge this gap, we constructed BIRD-Platinum by verifying

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.

2.5k instances sampled from BIRD Train (Li et al., 2023) with 2–4 rounds of independent error identification and correction by SQL experts. We corrected errors in 52% of SQL queries, 26% of natural language questions, and 18% of external knowledge entries. To support future text-to-SQL data curation, we analyze these error patterns and provide our verification guidelines in Section 3.

RL on BIRD-Platinum improves accuracy and generalizes beyond BIRD. We fine-tuned Qwen3-235B-A22B on BIRD-Platinum using RL with standard execution-based outcome rewards (Yao et al., 2025; Databricks, 2025; Ma et al., 2025). Compared with training the same model on BIRD Train, training on BIRD-Platinum improves accuracy by 11–16% on the Arcwise-Plat benchmarks, expert-verified versions of BIRD Mini-Dev (Jin et al., 2026; Arcwise, 2025). These gains generalize beyond BIRD. On Spider2-SQLite and Spider2-Snow (Lei et al., 2024), BIRD-Platinum improves accuracy by 12% and 14% over BIRD Train, respectively.

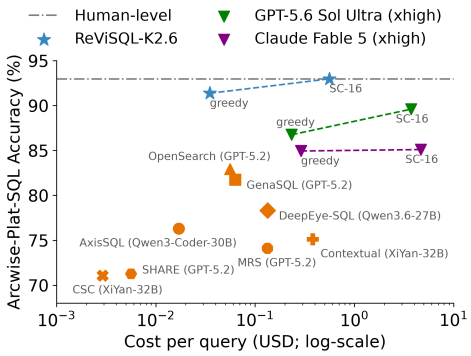


Figure 1: Accuracy and inference cost on Arcwise-Plat-SQL. REVISQL-K2.6 matches human performance with self-consistency over 16 candidates (SC-16), outperforms open-source systems by 9–22%, and surpasses GPT-5.6 Sol and Fable 5 by 3–8% at 12–15% of their cost.

REVISQL. Even with verified training data, RL with execution-based outcome rewards can provide misleading learning signals. First, we find that execution-based grading rewards semantically incorrect queries that coincidentally produce the gold result on a specific database. In a pilot training run, we identified 32.8% of positive rewards as false positives. Second, we find that fine-tuned models systematically ignore the provided external knowledge: task-provided domain information that specifies how to interpret the question or database. To address these limitations, we introduce REVISQL, which downweights execution-based rewards when bounded verification refutes equivalence and adds process reward penalties for omitting the required steps of checking external knowledge (§4).

We fine-tuned Kimi-K2.6 with REVISQL on BIRD-Platinum and evaluated the resulting model, REVISQL-K2.6, on Arcwise-Plat-Full and Arcwise-Plat-SQL (Jin et al., 2026). With greedy decoding, REVISQL-K2.6 achieves 93.8% and 91.4% accuracy, respectively. With self-consistency over 16 candidates, REVISQL-K2.6 reaches 94.2% and 93.0%, respectively. On both benchmarks, REVISQL-K2.6 matches the *proxy* human-level reference of 92.96% reported on BIRD (Li et al., 2023). Under greedy decoding, REVISQL-K2.6 outperforms OpenSearch (Xie et al., 2025), the strongest prior open-source pipeline, by 6–8%. On Arcwise-Plat-SQL, it also surpasses GPT-5.6 Sol Ultra and Claude Fable 5 by 3–8% at only 12–15% of their inference cost (Figure 1).

We summarize our contributions as follows:

1. **Verified training data.** We introduce BIRD-Platinum, a set of 2.5k expert-verified instances, which achieves consistent accuracy improvements on Arcwise-Plat and Spider2 over BIRD Train.
2. **Task-specific rewards.** We identify two limitations of standard RL rewards for text-to-SQL: false positive rewards and failure to incentivize external-knowledge use. We develop REVISQL to address these limitations via bounded verification and rule-based process rewards.
3. **Human-level performance.** We show that fine-tuning Kimi-K2.6 on BIRD-Platinum using REVISQL achieves human-level performance on Arcwise-Plat, outperforming prior state-of-the-art open-source pipelines, GPT-5.6 Sol Ultra, and Claude Fable 5.

2 RELATED WORK

In this section, we review existing text-to-SQL methods (Section 2.1) and datasets (Section 2.2).

2.1 TEXT-TO-SQL METHODS

Multi-stage LLM pipelines. Recent advances in text-to-SQL, particularly on benchmarks such as BIRD (Li et al., 2023), augment LLMs with complex inference-time pipelines (Wang et al., 2025;

Shkapenyuk et al., 2025; Pourreza et al.; Liu et al., 2025d; Sheng & Xu, 2025a; Dönder et al., 2025; Xie et al., 2025; Qu et al., 2025; Talaei et al., 2024; Sheng & Xu, 2025b; Li et al.; 2024; 2026). These pipelines address three common limitations. First, schema linking, question rewriting, and few-shot example retrieval help an LLM map natural language questions to relevant tables and columns (Shkapenyuk et al., 2025; Xie et al., 2025; Qu et al., 2025; Maamari et al.). Second, iterative refinement uses execution feedback to identify and repair errors in generated SQL queries (JDCHO, 2025; Wang et al., 2025; Pourreza et al.). Finally, prior work uses secondary reward models to select among multiple generations of SQL queries (Dönder et al., 2025; Liu et al., 2025d; Sheng & Xu, 2025a), or uses a planner model to route across multi-stage pipelines (JDCHO, 2025).

Although these modules improve accuracy, combining them increases inference cost and creates opportunities for errors to propagate across stages (Wang et al., 2025; Cemri et al., 2025). For example, a wrong schema-linking result corrupts all subsequent query generation and refinement. More fundamentally, these pipelines remain limited by the underlying model's intrinsic SQL capability.

Training on synthetic data. In parallel to building LLM pipelines, prior work has developed large-scale synthetic text-to-SQL data for training LLMs (Li et al., 2025; Pourreza et al., 2024; Wolff et al.). While synthetic data enables scalable training, it fails to address the inherent gap between human data and synthetic logic. These datasets typically generate natural language questions with clear, consistent logic, which does not reflect real-world deployment.

RL for text-to-SQL. RL has been a promising alternative to multi-stage pipelines, shifting investment from inference-time orchestration to training-time optimization of a model (Yao et al., 2025; Ma et al., 2025; Infly, 2025; Papicchio et al., 2025; Pourreza et al., 2025). However, prior efforts fail to outperform complex pipelines (Team, 2024). We argue this stems from two limitations. First, all these efforts train, at least in part, on BIRD Train whose data quality has not been verified. Second, prior efforts rely on standard execution-based outcome rewards, which we show in Section 4 are unreliable. While recent work uses formal verification for text-to-SQL evaluation (Klopfenstein et al., 2026), it has not yet been extended to RL reward shaping or model training.

2.2 Noise in Text-to-SQL Datasets

Recent audits have documented pervasive annotation errors in text-to-SQL datasets (Jin et al., 2026; Arcwise, 2025; Wretblad et al., 2024; Pourreza & Rafiei, 2023; Liu et al., 2025c). These errors compromise both evaluation and training. They not only make it difficult to distinguish model failures from incorrect benchmark annotations but also provide misleading learning signals.

Errors in evaluation sets. Prior audits of BIRD have identified annotation errors in 52% of instances from BIRD Dev (Jin et al., 2026) and noise in 15-49% of instances in its financial domain (Wretblad et al., 2024). Liu et al. (2025c) proposed labeling these errors to test error-detection capabilities of LLMs. For reliable evaluation, Arcwise (2025) and Jin et al. (2026) released Arcwise-Plat-Full and Arcwise-Plat-SQL, two verified variants of BIRD Mini-Dev that we use for evaluation.

The two variants differ in which sources of errors they corrected (Appendix A). Arcwise-Plat-Full corrects errors in gold queries, natural language questions, and external knowledge. Arcwise-Plat-SQL corrects only the gold queries and deliberately retains the original questions and external knowledge, including their factual errors and ambiguities. These two variants thus evaluate different capabilities. Arcwise-Plat-Full measures accuracy under clean inputs, while Arcwise-Plat-SQL measures robustness to noisy inputs that are closer to real-world deployments (Liu et al., 2025b).

Errors in training sets. While prior work has verified the BIRD Mini-Dev evaluation set, text-to-SQL training data remains underexamined. Pourreza & Rafiei (2023) partially categorized errors in BIRD-Train into sorting errors, schema matching ambiguity, and incorrect content assumptions, detecting errors in 18.2% of a subset. To date, there are no large-scale, publicly available corrections for BIRD-Train. This lack of clean data represents a fundamental barrier to model training.

3 Curating a Verified Dataset for Text-to-SQL RLVR

We constructed BIRD-Platinum, a verified dataset of 2.5k instances sampled from BIRD Train and corrected through a hybrid LLM–human-expert pipeline. In this section, we describe the design principles behind our correction process (§3.1), the four-stage pipeline that implements them (§3.2),

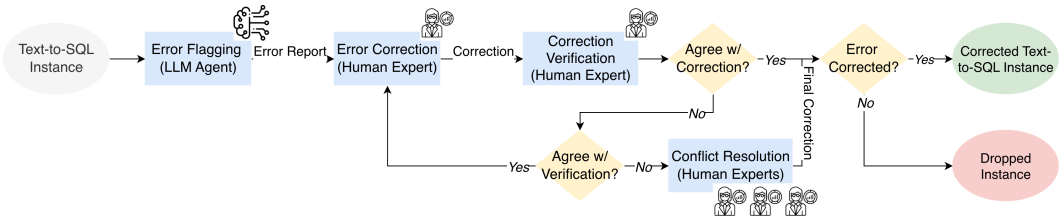


Figure 2: A hybrid LLM–human-expert pipeline for correcting text-to-SQL annotation errors.

a taxonomy of the errors we encountered with worked examples (§3.3), and summary statistics for both the construction process and the resulting dataset (§3.4).

3.1 Design Principles and Verification Checklist

Soundness over completeness. For a verified dataset to improve RL training, two properties of data cleaning are essential: (1) every correction must target a *genuine* annotation error; (2) corrections themselves must not introduce new errors. We optimize the pipeline for these two soundness properties, even at the cost of completeness. We do not claim to find every error in every instance. Instead, we ensure that every change we make is principled and verified by independent experts.

Checklist. To achieve soundness, we develop a checklist that experts apply to every instance. The checklist enumerates the specific failure modes we observed during pilot annotation and serves as both a guide for correction and verification. We defer a full checklist to Appendix B.

3.2 Construction Pipeline

Each instance passes through four stages designed to combine the efficiency of LLM-based error detection with the precision of human expert judgment (Figure 2). The pipeline incorporates correction with independent verification to not propagate individual mistakes to the final dataset.

Stage 1: LLM-assisted error flagging. We applied an LLM reviewer (Jin et al., 2026) based on OpenAI o3 to detect annotation errors and provide correction suggestions. The agent can execute SQL queries on an example database with a budget of 30 queries per instance. The agent generates an error report describing errors in the question, external knowledge, and SQL query.

Stage 2: Independent expert correction. A human expert independently investigates the instance *before* reading the LLM agent's report. The expert then incorporates the LLM report and produces a draft correction covering the question, external knowledge, and SQL query.

Stage 3: Independent verification. A second expert independently re-investigates an instance and drafts a new correction. The expert verifies that each correction is sound and searches for errors the first expert may have missed. Disagreement between the correction and verification experts triggers revision by a third expert.

Stage 4: Conflict resolution. When verification and the third experts disagree on the substance of a correction, three experts convene to reach consensus. In our process, every conflict was resolved at this stage. The one outcome that we treat specially is when the experts agree that no correction is possible. This is typically because the question cannot be answered from the given data schema (Appendix C.1). In these cases, we drop the instance.

3.3 Taxonomy of Annotation Errors

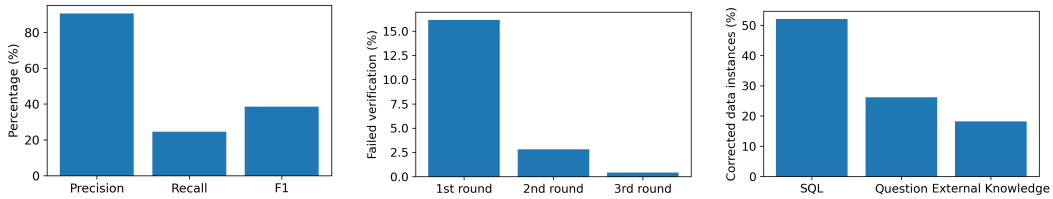
The corrections produced by our pipeline form a more comprehensive taxonomy of annotation errors than those investigated in prior work (Jin et al., 2026; Liu et al., 2025c; Wretblad et al., 2024). We describe the taxonomy to characterize BIRD-Platinum and hopefully to inspire future data curation. We organize the identified errors into four categories and defer detailed examples to Appendix C.

Question and external knowledge errors. We identified and corrected the following three types of errors in the natural language questions and external knowledge. First, we discard questions that the

Table 1: Structural complexity comparison between BIRD Train and BIRD-Platinum. We find that gold SQL queries exhibit increases in complexity across every structural metric.

Per-query stats	# tables	# joins	# functions	# aggregations	# set ops.	# subqueries	# CTEs	# window func.
BIRD Train	2.08	1.02	1.67	0.62	0.0024	0.088	0.0	0.0012
BIRD-Platinum	2.36	1.06	2.10	0.80	0.024	0.26	0.11	0.0012
Difference	+13.5%	+3.9%	+25.7%	+29.0%	+900%	+195.5%	—*	+2.0%

* The original BIRD Train contains no CTEs, so the relative change is undefined.



(a) LLM reviewer demonstrates high precision but low recall.

(b) Iterative expert correction and verification required up to 4 rounds.

(c) Pervasive annotation errors in queries, questions, and knowledge.

Figure 3: Quantitative analysis of the BIRD-Platinum expert curation process. Relying solely on automated LLMs for data correction is insufficient due to poor recall (3a). Therefore, we use an expert-driven pipeline that requires up to four iterative rounds to pass manual verification (3b). We identified and corrected errors for over 50% of instances (3c).

provided database schema cannot answer. Second, for questions whose filters yield empty results, we revise the filters and corresponding gold queries to produce non-empty targets. Otherwise, trivial queries that return nothing can receive full execution-based reward (Skalse et al., 2022). Finally, we correct external knowledge that contradicts the question's semantics (Appendix C.1).

Gold SQL query errors. We correct queries with incorrect or missing tables, joins, predicates, output columns, grouping keys, or ranking. We also repair semantic errors that require deep data analysis, such as excluding missing data using indicator columns, preserving ties when the question requires all extrema, and removing duplicates when counting distinct entities (Appendix C.2). These errors can propagate through RL into a model. In an ablation study, a model trained on queries that omitted deduplication also failed to remove duplicates in 83% of validation problems.

Schema errors. We correct schema for 1.7% of tables to align it with the databases. We removed references to nonexistent entities and revised descriptions that contradict values (Appendix C.3).

Grading-method errors. Standard set-based grading (Li et al., 2023) cannot fully evaluate questions that allow multiple valid subsets or require a specific row order. We annotate these instances with alternative grading methods (Appendix C.4). For any-k questions, we use *subset-based grading*: the gold query returns all eligible answers, and the generated result must contain exactly k elements from that set. For questions that specify row order, we use *list-based grading*, requiring the generated and gold result tables to match in dimensions and values at each cell position.

3.4 Construction and Dataset Statistics

Construction process. In Figure 3, we highlight the effectiveness of the LLM reviewer, the necessity of expert involvement, and the importance of multi-round correction. As shown in Figure 3a, the LLM reviewer achieves a high precision of 90.6% and a low recall of 24.5% in error detection. While the LLM reviewer is relatively reliable, expert involvement in data correction remains necessary.

As shown in Figure 3b, we find that 16.2% of the first-round proposed corrections fail the subsequent verification, indicating the necessity of the multi-round process. We encountered five cases (0.2%) where conflicts occurred between correction and verification, which were resolved at the conflict-resolution stage. Eventually, all the proposed corrections passed verification after up to four rounds.

Dataset characteristics. In Figure 3c and Table 1, we summarize the statistics of our corrections and a comparison between BIRD Train and BIRD-Platinum. As shown in Figure 3c, our process

corrected gold SQL queries in 52.1% of instances, natural language questions in 26.2%, and external knowledge in 18.2%. Additionally, we discarded 1.5% of instances whose questions are unanswerable given the database schema. In total, 61.1% of the instances contain at least one error.

In Table 1, we show the average number of tables, joins, functions, aggregations, set operations, subqueries, common table expressions (CTEs), and window functions per query. We find that our corrected queries have higher average counts in all categories, indicating that the corrected gold queries exhibit greater structural complexity, reflecting a rigorous adherence to domain constraints.

4 REVISQL: SHAPING RLVR FOR EFFECTIVE TEXT-TO-SQL LEARNING

We propose REVISQL, a reward shaping method tailored to text-to-SQL. We identify two failures of the standard RL rewards (§4.1) and describe our novel reward shaping technique (§4.2).

4.1 DIAGNOSING THE FAILURES OF EXECUTION-BASED OUTCOME REWARDS

BIRD-style text-to-SQL problems. We target the text-to-SQL setting introduced by BIRD (Li et al., 2023). A *BIRD-style text-to-SQL problem* provides (1) a natural language question, (2) a relational database, given by its schema and contents, and (3) a set of *external knowledge* entries. The goal is to produce a SQL query that, when executed against the database, returns the answer to the question. The external knowledge, termed *evidence* in BIRD, consists of short natural-language annotations that are "required to map the natural language instructions into counterpart database values" (Li et al., 2023). Because a question is often under-specified without this evidence, a text-to-SQL model is expected to read the external knowledge and reflect it in the generated query.

Prior RL methods for text-to-SQL (Yao et al., 2025; Databricks, 2025; Ma et al., 2025) use binary execution-based outcome rewards, setting the reward to 1 if the result of the generated SQL on the example database is the same as the gold SQL query's, and 0 otherwise. While this reward is cheap to compute, we show that it corrupts text-to-SQL learning in two systematic ways.

Failure 1: false positive rewards from result coincidence. Execution equivalence on an example database instance does not guarantee semantic SQL equivalence (Chen et al., 2025b; He et al., 2024). Queries with incorrect joins, missing predicates, or compensating constants can coincidentally produce the gold result. Rewarding these queries reinforces logical errors that the example database fails to expose. In Appendix D.1, we illustrate this failure with two queries that use different aggregation rules but return the same result.

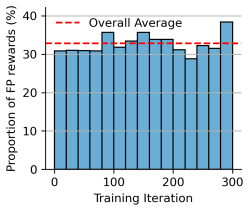


Figure 4: False positives persist across the entire training process of Kimi-K2.6.

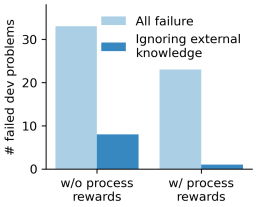


Figure 5: Process rewards reduce the failures of ignoring external knowledge entries.

We quantified this failure during a pilot run with Kimi-K2.6 using VeriEQL (He et al., 2024), which verifies SQL equivalence under bounded inputs. We classify a reward as a false positive when execution-based grading accepts the query but VeriEQL refutes. Across the run, VeriEQL identified 28.8–38.4% of positive rewards as false positives, with an average of 32.8% (Figure 4).

Failure 2: outcome rewards do not incentivize external-knowledge use. External knowledge often specifies domain constraints, while outcome rewards evaluate only the generated query and provide no direct supervision for applying external knowledge. Hence, a model can adopt a plausible interpretation that conflicts with the supplied knowledge while passing grading (Appendix D.2).

We manually analyzed 33 validation failures from Kimi-K2.6 trained with outcome rewards. We classify a failure as ignoring external knowledge when the reasoning follows an interpretation that the knowledge rules out or misses a requirement that the knowledge specifies. As shown in Figure 5, we found that 24.2% of failures meet this criterion. Outcome rewards cannot distinguish reasoning that follows the knowledge from reasoning that ignores it but coincidentally returns the same result.

4.2 REWARD SHAPING WITH REVISQL

We develop a composite reward that complements the execution-based reward and targets the two failures diagnosed above. For a generated rollout τ with gold answer a, the REVISQL reward is

$$R_{\text{REVISQL}}(\tau, a) \;=\; \rho(\tau, a, \beta) \;-\; \sum_{k \in \mathcal{K}} \lambda_k \cdot \mathbb{1}[\text{violation}_k(\tau)] \tag{1}$$

where ρ is a verification-aware outcome reward that mitigates false positives with penalty β, and $\mathcal{K}$ is a set of rules for external-knowledge use whose penalties λ_k are deducted when violated.

Verification-aware outcome reward. To address Failure 1, we integrate VeriEQL (He et al., 2024) as a second-stage check after the execution-based grading. We set the reward $\rho(\tau, a, \beta) = 0$ if the generated query fails execution-based grading. Otherwise, the reward is $1 - \beta$ if VeriEQL refutes equivalence to the gold query, and 1 if VeriEQL confirms equivalence, times out, or does not support the query. This formulation penalizes detected semantic errors while preserving partial credit for matching the gold result and retaining the original reward when verification is inconclusive.

Process supervision for external knowledge use. To address Failure 2, we require the model to explicitly translate and verify each external knowledge entry as part of its reasoning process. We use a system prompt that requires reasoning blocks: (1) a "requirement" block that translates each external knowledge entry into a concrete query constraint, and (2) a "verification" block right before the final answer that audits whether the generated SQL query satisfies each knowledge entry.

We deduct penalties λ_k when the model generates a solution without complying with these requirements. We conducted a pilot evaluation on the dev set to validate our reward shaping intervention. With process rewards added, only 4.3% of failed instances are attributable to ignoring external knowledge. We defer more ablation results on benchmarks to Section 5.2.

Implementation. We fine-tune models using LoRA and the CISPO loss (Chen et al., 2025a). During each rollout, the model can execute intermediate SQL queries to inspect the database for exploration. In Appendix E, we detail the training setup, rollout limits, and hyperparameters.

5 EXPERIMENTS

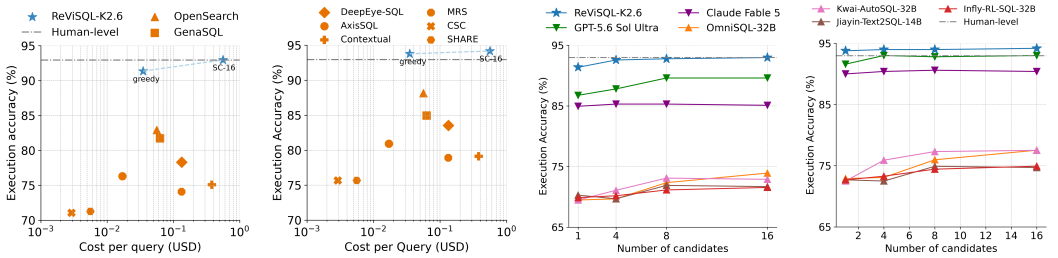
We evaluate BIRD-Platinum and REVISQL on four benchmarks, examining accuracy relative to prior systems and models. We also demonstrate contributions of each component (Section 5.2), and generalization beyond BIRD (Section 5.3).

5.1 EXPERIMENTAL SETUP

Benchmarks. We evaluate on four benchmarks. First, as discussed in Section 2.2, we use the expert-verified versions of BIRD Mini-Dev: Arcwise-Plat-Full and Arcwise-Plat-SQL (Jin et al., 2026). Furthermore, we evaluate on two variants of Spider2 (Lei et al., 2024), the successor of the widely used Spider1 benchmark (Yu et al., 2018). Following prior work (Li et al., 2025), we use Spider2-SQLite, containing 135 SQLite-based Spider2 problems. We also use the official Spider2-Snow, containing all 547 Spider2 problems with gold queries written in the Snowflake dialect.

Baselines. We compare with the top-eight open-source pipelines on the BIRD leaderboard: Contextual-SQL (Agrawal & Nguyen, 2025), CSC-SQL (Sheng & Xu, 2025a), GenaSQL (Dönder et al., 2025), OpenSearch (Xie et al., 2025), SHARE (Qu et al., 2025), AxisSQL (Shareef, 2026), MRS (Chan et al., 2026), and DeepEye-SQL (Li et al., 2026). We replace the GPT-4o in OpenSearch and GenaSQL with GPT-5.2, a newer model with better performance and comparable cost. We also compare with the top-four open-weight text-to-SQL models on the leaderboard: OmniSQL-32B (Li et al., 2025), Infly-RL-SQL-32B (Infly, 2025), Kwai-AutoSQL-32B (Kwai, 2026), and Jiayin-Pangu-Text2SQL-14B (Research, 2026). In addition, we evaluate two proprietary models: GPT-5.6 Sol Ultra and Claude Fable 5. We defer detailed configurations to Appendix F.

Metrics. Following prior work (Li et al., 2023; Lei et al., 2024), we measure the execution accuracy of generated SQL queries. We also measure the inference cost for each SQL query, using the official token pricing for proprietary models and the cheapest pricing of open-source models provided by Together (Team, 2025d), Groq (Team, 2025b), Tinker (Tinker, 2025), or Fireworks (Team, 2025a).



(a) Arcwise-Plat-SQL. (b) Arcwise-Plat-Full. (c) Arcwise-Plat-SQL. (d) Arcwise-Plat-Full.

Figure 6: REVISQL-K2.6 reaches human-level accuracy on Arcwise-Plat and outperforms prior methods, including top open-source text-to-SQL pipelines and single-model baselines.

5.2 REVISQL REACHES HUMAN-LEVEL ACCURACY ON BIRD

We fine-tuned Kimi-K2.6 using REVISQL on BIRD-Platinum and evaluated the resulting model, REVISQL-K2.6, on Arcwise-Plat-SQL and Arcwise-Plat-Full under two decoding methods: (1) greedy decoding and (2) self-consistency over 16 candidates (SC-16). For self-consistency, we follow the standard procedure (Wang et al., 2022) to sample candidates with temperature 1, group them by execution results, and return a query from the largest group, breaking ties randomly.

REVISQL-K2.6 achieves human-level accuracy. As shown in Figure 6b, with greedy decoding, REVISQL-K2.6 achieves 93.8% on Arcwise-Plat-Full. As shown in Figure 6a, with SC-16, REVISQL-K2.6 achieves 93.0% on Arcwise-Plat-SQL. On both benchmarks, REVISQL-K2.6 matches the *proxy* human-level accuracy of 92.96%. In contrast, all evaluated open-source systems fall significantly below the human proxy. To our knowledge, REVISQL-K2.6 is the first text-to-SQL method to achieve human parity on both expert-verified BIRD datasets.

REVISQL-K2.6 outperforms complex pipelines. As shown in Figures 6a and 6b, the prior best open-source pipeline, OpenSearch, achieves 82.9% on Arcwise-Plat-SQL and 88.2% on Arcwise-Plat-Full with $0.056 per query. REVISQL-K2.6 with greedy decoding outperforms OpenSearch by 8.4% and 5.6%, *while costing 37% less per query*. Compared to other pipelines, REVISQL-K2.6 with greedy decoding achieves 9.6–20.3% and 8.8–18.1% higher accuracy, respectively. With SC-16, REVISQL-K2.6 achieves 10.0–21.9% and 6.0–18.5% higher accuracy, respectively.

REVISQL-K2.6 outperforms open-weight and proprietary models. In Figures 6c and 6d, we compare REVISQL-K2.6 against the best open-weight text-to-SQL models, GPT-5.6 Sol Ultra and Fable 5, as the number of candidates varies from 1 to 16. Compared to open-source models, REVISQL-K2.6 achieves 16.7–22.9% higher accuracy across different benchmarks and candidate budgets. Compared to proprietary models, REVISQL-K2.6 achieves higher accuracy by 3.2–7.9% on Arcwise-Plat-SQL and 1.0–3.8% on Arcwise-Plat-Full, at 12–15% of their inference cost.

BIRD-Platinum is a dominant factor while REVISQL bridges the gap to human parity. To isolate the contribution of each ingredient, we ablate REVISQL-K2.6 under greedy decoding (Figure 7), comparing across three settings: (1) fine-tuning Kimi-K2.6 on BIRD-Platinum with standard execution-based rewards, (2) REVISQL on BIRD Train, and (3) Kimi-K2.6 without fine-tuning.

We find that verified data is the dominant driver. Training on BIRD-Platinum improves the accuracy of Kimi-K2.6 by 7.2% and 5.0% on Arcwise-Plat-SQL and -Full, whereas training on BIRD Train lowers it by 7.0% and 5.0%. Furthermore, REVISQL reward bridges the residual gap, improving the accuracy by 2.8% and 3.8%. While these absolute gains appear modest, they are decisive at this accuracy level. Reward shaping removes 24.6% and 38.0% of the errors that training with verified data alone leaves behind.

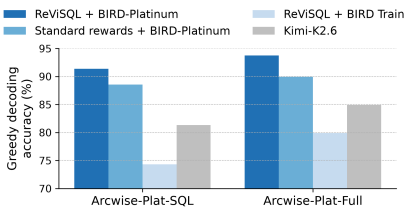


Figure 7: Ablation of REVISQL-K2.6 on Arcwise-Plat. With REVISQL, replacing BIRD Train with BIRD-Platinum improves accuracy by 13.9–17.1%. On BIRD-Platinum, adding reward shaping improves accuracy by 2.8–3.8% over standard rewards, reducing error rates by 24.6–38.0%.

Table 2: Fine-tuning Qwen3-235B-A22B on BIRD-Platinum outperforms the strongest prior open-source pipelines on Arcwise-Plat and Spider2-SQLite, at lower per-query cost.

Method	Arcwise-Plat-SQL (%)	Arcwise-Plat-Full (%)	Spider 2-SQLite (%)	Per-query cost (USD)
SHARE	71.29	75.70	21.48	5.6×10^{-3}
CSC-SQL	71.08	75.70	10.37	2.9×10^{-3}
MRS agent	74.10	78.92	17.78	1.3×10^{-1}
Contextual-SQL	75.10	79.12	11.11	3.8×10^{-1}
AxisSQL	76.31	80.92	15.56	1.7×10^{-2}
DeepEye-SQL	78.31	83.53	21.48	1.3×10^{-1}
GenaSQL	81.73	84.94	34.81	6.3×10^{-2}
OpenSearch	82.93	88.15	34.07	5.6×10^{-2}
Fine-tuned Qwen3-235B	**87.34**	**88.75**	**37.04**	7.7×10^{-3}

5.3 BIRD-Platinum Improves Accuracy and Generalizes Across Benchmarks

We evaluate BIRD-Platinum and whether its benefits generalize beyond BIRD. To isolate the contribution of training data, we compare Qwen3-235B-A22B fine-tuned on BIRD-Platinum with the same model fine-tuned on BIRD-Train, using the standard execution-based rewards for both.

Although BIRD-Platinum contains only SQLite queries, we evaluate the trained models on Spider2-Snow without retraining via two adaptations. First, we use SQLGlot (Team, 2025c) to translate generated queries into Snowflake SQL. Second, we apply ReFoRCE's table and column filtering (Deng et al., 2025) to retain relevant schema elements when the schema and question exceed the model's context window. We match other inference settings with those used for Arcwise-Plat.

BIRD-Platinum improves accuracy across benchmarks. In Figure 8, we compare Qwen3-235B-A22B fine-tuned on BIRD-Platinum with the same model fine-tuned on BIRD Train. Training on BIRD-Platinum improves accuracy by 16.1% on Arcwise-Plat-SQL and 11.0% on Arcwise-Plat-Full. Furthermore, training on BIRD-Platinum achieves 11.8% and 14.0% higher accuracy on Spider2-SQLite and Spider2-Snow compared to training on BIRD Train. With self-consistency over 128 candidates, Qwen3-235B-A22B fine-tuned on BIRD-Platinum reaches 55.58% accuracy on Spider2-Snow, outperforming prior complex pipelines built on larger models, including DeepSeek-R1 and Qwen3-Coder (Appendix G). These results show that the benefits of BIRD-Platinum extend to other benchmarks and SQL dialects.

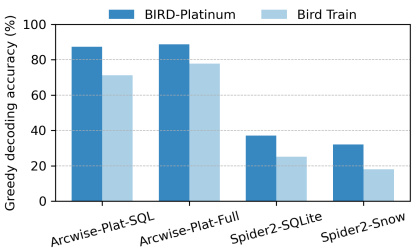


Figure 8: Training on BIRD-Platinum yields consistent accuracy improvements (11.0–16.1%) across Arcwise-Plat and Spider2 benchmarks, compared with training on BIRD Train.

Fine-tuned Qwen3-235B-A22B outperforms the open-source pipelines. In Table 2, we compare Qwen3-235B-A22B fine-tuned on BIRD-Platinum against the top open-source systems. On Spider2-SQLite, with greedy decoding, our fine-tuned model achieves 37.04% accuracy at $0.0077 per query, outperforming the previous best pipeline by 2.2% at 8× lower cost. In addition, fine-tuned Qwen3-235B-A22B also leads on Arcwise-Plat-SQL and Arcwise-Plat-Full, outperforming prior pipelines by 0.6–16.3%.

6 Conclusion

We show that human-level text-to-SQL accuracy is achievable by fine-tuning a single LLM with RL on verified data and task-specific rewards, without a complex inference pipeline. We introduce BIRD-Platinum, a dataset of 2.5k instances with corrected annotations in 61% of the instances. Training on BIRD-Platinum improves the accuracy on the expert-verified Arcwise-Plat benchmarks and Spider2 benchmarks by 11-16% over BIRD Train. Furthermore, we introduce ReViSQL, which combines bounded verification to reduce false positive rewards with rule-based verification to encourage external-knowledge use. Fine-tuning Kimi-K2.6 on BIRD-Platinum with ReViSQL produces a model that achieves human-level accuracy (92.96%) on Arcwise-Plat, outperforming open-source pipelines, GPT-5.6 Sol Ultra, and Fable 5 at lower inference cost.

AI USE STATEMENT

In this work, we used generative AI tools for checking grammar, checking format correctness of citations, and part of the data curation as mentioned in Section 3. We have not used generative AI tools for generating ideas, running experiments, writing complete sentences, paragraphs, or sections, or anything related to dataset construction beyond what's mentioned in Section 3. Helping develop theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, proposing or refining hypotheses, designing or providing feedback on research methodology or experiments, implementing methods, assisting with translation, supporting qualitative and thematic data analysis, and interpreting results are not applicable to this work. We have reviewed all AI-assisted work by manually verifying LLM's grammar corrections, citation format corrections, and proposed annotation corrections (Section 3). We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We have provided code and data in the supplementary material of the submission, and detailed configurations to reproduce our results, including prompts and reward formulations, in Appendices.

REFERENCES

Sheshansh Agrawal and Thien Nguyen. Open-sourcing the best local text-to-sql system, 2025. URL https://contextual.ai/blog/open-sourcing-the-best-local-text-to-sql-system/.

Ion Androutsopoulos, Graeme D Ritchie, and Peter Thanisch. Natural language interfaces to databases–an introduction. *Natural language engineering*, 1(1):29–81, 1995.

Arcwise. Bird minidev - corrections, 2025. URL https://docs.google.com/spreadsheets/d/1IGm9Otruey60ujUnl8AOkepY3qgWHdFJHnX7hQGUeCw. Accessed: 2025-09-15.

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657*, 2025.

Sokheang Chan, Vungsovanreach Kong, Anand Nayyar, and Tae-Kyung Kim. Mrs-agent: A map-reduce framework for quantifiable schema grounding in text-to-sql. *ICT Express*, 2026.

Aili Chen, Aonian Li, Bangwei Gong, Binyang Jiang, Bo Fei, Bo Yang, Boji Shan, Changqing Yu, Chao Wang, Cheng Zhu, et al. Minimax-m1: Scaling test-time compute efficiently with lightning attention. *arXiv preprint arXiv:2506.13585*, 2025a.

Chunyu Chen, Zhengjie Miao, Yong Zhang, and Jiannan Wang. Parseval: Plan-aware test database generation for sql equivalence evaluation. *Proceedings of the VLDB Endowment*, 18(11):4750–4762, 2025b.

Databricks. The power of rlvr: Training a leading sql reasoning model on databricks, 2025. URL https://www.databricks.com/blog/power-rlvr-training-leading-sql-reasoning-model-databricks. Accessed: 2025-10-10.

Minghang Deng, Ashwin Ramachandran, Canwen Xu, Lanxiang Hu, Zhewei Yao, Anupam Datta, and Hao Zhang. Reforce: a text-to-sql agent with self-refinement, consensus enforcement, and column exploration. *arXiv preprint arXiv:2502.00675*, 2025.

Yusuf Denizay Dönder, Derek Hommel, Andrea W Wen-Yi, David Mimno, and Unso Eun Seo Jo. Cheaper, better, faster, stronger: Robust text-to-sql without chain-of-thought or fine-tuning. *arXiv preprint arXiv:2505.14174*, 2025.

Yang He, Pinhan Zhao, Xinyu Wang, and Yuepeng Wang. Verieql: Bounded equivalence verification for complex sql queries with integrity constraints. *Proceedings of the ACM on Programming Languages*, 8(OOPSLA1):1071–1099, 2024.

Infly. infly/inf-rl-qwen-coder-32b-2746, 2025. URL https://huggingface.co/infly/inf-rl-qwen-coder-32b-2746.

JDCHO. Joyagent-jdgenie, 2025. URL https://github.com/jd-opensource/joyagent-jdgenie.

Tengjun Jin, Yoojin Choi, Yuxuan Zhu, and Daniel Kang. Pervasive annotation errors break text-to-sql benchmarks and leaderboards. *VLDB*, 2026.

Rocky Klopfenstein, Yang He, Andrew Tremante, Yuepeng Wang, Nina Narodytska, and Haoze Wu. Spotit: Evaluating text-to-sql evaluation with formal verification. In *International Conference on Learning Representations*, volume 2026, pp. 69012–69048, 2026.

Kwai. Kwai-autosql-32b, 2026. URL https://huggingface.co/Kwai-AutoSQL/Kwai-AutoSQL-32B. Accessed on 09-24-2026.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, et al. Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows. *arXiv preprint arXiv:2411.07763*, 2024.

Boyan Li, Jiayi Zhang, Ju Fan, Yanwei Xu, Chong Chen, Nan Tang, and Yuyu Luo. Alpha-sql: Zero-shot text-to-sql using monte carlo tree search. In *Forty-second International Conference on Machine Learning*.

Boyan Li, Chong Chen, Zhujun Xue, Yinan Mei, and Yuyu Luo. DeepEye-SQL: A software-engineering-inspired text-to-sql framework. *Proceedings of the ACM on Management of Data*, 4 (3), 2026. doi: 10.1145/3802035. URL https://doi.org/10.1145/3802035.

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. *Proceedings of the ACM on Management of Data*, 2(3):1–28, 2024.

Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, et al. Omnisql: Synthesizing high-quality text-to-sql data at scale. *arXiv preprint arXiv:2503.02240*, 2025.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 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, 2023.

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

Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Nan Tang, and Yuyu Luo. A survey of text-to-sql in the era of llms: Where are we, and where are we going? *IEEE Transactions on Knowledge and Data Engineering*, 2025b.

Xinyu Liu, Shuyu Shen, Boyan Li, Nan Tang, and Yuyu Luo. Nl2sql-bugs: A benchmark for detecting semantic errors in nl2sql translation. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 5662–5673, 2025c.

Yifu Liu, Yin Zhu, Yingqi Gao, Zhiling Luo, Xiaoxia Li, Xiaorong Shi, Yuntao Hong, Jinyang Gao, Yu Li, Bolin Ding, et al. Xiyan-sql: A novel multi-generator framework for text-to-sql. *arXiv preprint arXiv:2507.04701*, 2025d.

Peixian Ma, Xialie Zhuang, Chengjin Xu, Xuhui Jiang, Ran Chen, and Jian Guo. Sql-r1: Training natural language to sql reasoning model by reinforcement learning. *arXiv preprint arXiv:2504.08600*, 2025.

Karime Maamari, Fadhil Abubaker, Daniel Jaroslawicz, and Amine Mhedhbi. The death of schema linking? text-to-sql in the age of well-reasoned language models. In *NeurIPS 2024 Third Table Representation Learning Workshop*.

Simone Papicchio, Simone Rossi, Luca Cagliero, and Paolo Papotti. Think2sql: Reinforce llm reasoning capabilities for text2sql. *arXiv preprint arXiv:2504.15077*, 2025.

Mohammadreza Pourreza and Davood Rafiei. Evaluating cross-domain text-to-sql models and benchmarks. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 1601–1611, 2023.

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. In *The Thirteenth International Conference on Learning Representations*.

Mohammadreza Pourreza, Ruoxi Sun, Hailong Li, Lesly Miculicich, Tomas Pfister, and Sercan O Arik. Sql-gen: Bridging the dialect gap for text-to-sql via synthetic data and model merging. *arXiv preprint arXiv:2408.12733*, 2024.

Mohammadreza Pourreza, Shayan Talaei, Ruoxi Sun, Xingchen Wan, Hailong Li, Azalia Mirhoseini, Amin Saberi, Sercan Arik, et al. Reasoning-sql: Reinforcement learning with sql tailored partial rewards for reasoning-enhanced text-to-sql. *arXiv preprint arXiv:2503.23157*, 2025.

Ge Qu, Jinyang Li, Bowen Qin, Xiaolong Li, Nan Huo, Chenhao Ma, and Reynold Cheng. Share: An slm-based hierarchical action correction assistant for text-to-sql. *arXiv preprint arXiv:2506.00391*, 2025.

Jiayin AI Research. Jiayin-pangu-text2sql-14b, 2026. URL https://huggingface.co/Jiayin-AI-Research/Jiayin-Pangu-Text2Sql-14B-135. Accessed on 09-24-2026.

Shaik Nagur Shareef. AxisSQL: mapping the axes of inference-time scaling for text-to-sql. https://github.com/ShaikNagurShareef/AxisSQL, 2026.

Lei Sheng and Shuai-Shuai Xu. Csc-sql: Corrective self-consistency in text-to-sql via reinforcement learning. *arXiv preprint arXiv:2505.13271*, 2025a.

Lei Sheng and Shuai-Shuai Xu. Slm-sql: An exploration of small language models for text-to-sql. *arXiv preprint arXiv:2507.22478*, 2025b.

Liang Shi, Zhengju Tang, Nan Zhang, Xiaotong Zhang, and Zhi Yang. A survey on employing large language models for text-to-sql tasks. *ACM Computing Surveys*, 58(2):1–37, 2025.

Vladislav Shkapenyuk, Divesh Srivastava, Theodore Johnson, and Parisa Ghane. Automatic metadata extraction for text-to-sql. *arXiv preprint arXiv:2505.19988*, 2025.

Joar Skalse, Nikolaus HR Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, pp. 9460–9471, 2022.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. Chess: Contextual harnessing for efficient sql synthesis. *arXiv preprint arXiv:2405.16755*, 2024.

BIRD Team. Bird-bench, 2024. URL https://bird-bench.github.io/. Accessed: 2025-02-10.

Fireworks Team. Fireworks pricing page, 2025a. URL https://fireworks.ai/pricing. Accessed: 2026-02-10.

Groq Team. Groq pricing page, 2025b. URL https://groq.com/pricing. Accessed: 2025-10-10.

SQLGlot Team. Sqlglot: Python sql parser and transpiler, 2025c. URL https://github.com/tobymao/sqlglot. Accessed: 2025-10-10.

Together AI Team. Together ai pricing page, 2025d. URL https://together.ai/pricing. Accessed: 2025-10-10.

Tinker. Tinker: a training api for researchers and developers, 2025. URL https://tinker-docs.thinkingmachines.ai/. Accessed: 2025-10-10.

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

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Cornelius Wolff, Daniel Gomm, and Madelon Hulsebos. Sqale: A large text-to-sql corpus grounded in real schemas. In *EurIPS 2025 Workshop: AI for Tabular Data*.

Niklas Wretblad, Fredrik Riseby, Rahul Biswas, Amin Ahmadi, and Oskar Holmström. Understanding the effects of noise in text-to-sql: An examination of the bird-bench benchmark. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 356–369, 2024.

Xiangjin Xie, Guangwei Xu, Lingyan Zhao, and Ruijie Guo. Opensearch-sql: Enhancing text-to-sql with dynamic few-shot and consistency alignment. *Proceedings of the ACM on Management of Data*, 3(3):1–24, 2025.

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

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3911–3921, 2018.

Table 3: Correction statistics of two expert-verified datasets based on BIRD mini-Dev.

Correction	Arcwise-Plat-Full	Arcwise-Plat-SQL
SQL query	132 (26.5%)	80 (16.1%)
Question	88 (17.7%)	0
External knowledge	97 (19.5%)	0

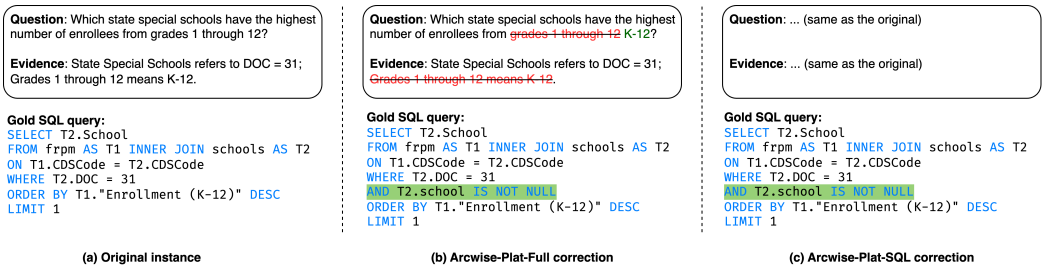


Figure 9: A representative instance of two expert-verified variants of BIRD Mini-Dev. (a) The original instance contains an incorrect phrasing in the question ("grades 1 through 12" as opposed to "K-12") and a gold SQL query that omits a `NOT NULL` filter, causing it to include data about districts and schools. (b) Arcwise-Plat-Full corrects all three error sources: rephrasing the question to "K-12," removing the redundant external-knowledge entry, and augmenting gold SQL. (c) Arcwise-Plat-SQL corrects only the gold SQL, preserving the original noisy question and external knowledge.

A Two Expert-Verified Arcwise-Plat Benchmarks

We evaluate on two expert-verified variants of the BIRD Mini-Dev set released by prior work (Arcwise, 2025; Jin et al., 2026): Arcwise-Plat-Full and Arcwise-Plat-SQL. The two are built from the same instances but differ in which sources of annotation error are corrected (Table 3). Arcwise-Plat-Full corrects all three sources: the natural language question, the external knowledge, and the gold SQL query. Therefore, it measures text-to-SQL accuracy under clean inputs. Arcwise-Plat-SQL corrects only the gold SQL query and *deliberately preserves* the original, often noisy, questions and external knowledge. Hence, it measures robustness to the ambiguous inputs that are typical of real-world text-to-SQL deployments (Liu et al., 2025b). We walk through a representative instance to make the distinction concrete.

A representative instance. In Figure 9, we show one BIRD Mini-Dev instance under all three versions. The original instance (Figure 9a) contains an error in each of the three sources. The question asks about schools serving "grades 1 through 12," an incorrect phrasing of the intended "K-12" criterion; the external knowledge includes a redundant entry; and the gold SQL query omits a `NOT NULL` filter, so its result wrongly includes rows about districts in addition to schools.

Arcwise-Plat-Full repairs all three (Figure 9b): it rephrases the question to "K-12," removes the redundant external-knowledge entry, and augments the gold SQL with the missing filter. A model evaluated here sees an unambiguous question and is graded against a correct gold query, so the benchmark isolates the model's SQL-reasoning ability.

Arcwise-Plat-SQL repairs only the gold SQL (Figure 9c): the missing `NOT NULL` filter is added, but the noisy "grades 1 through 12" question and the redundant external knowledge are left untouched. A model must now recover the intended "K-12" meaning from an imperfect question without the benefit of a sanitized prompt. This is the harder, deployment-realistic condition: the input is exactly as noisy as the original benchmark, yet the gold answer is now correct, so a model is neither rewarded for matching an erroneous gold query nor rescued by a cleaned question.

B TEXT-TO-SQL DATA VERIFICATION CHECKLIST

For each natural language question, we check:

1. Is the question answerable from the given database?
2. Does the question admit a single interpretation?
3. Is the expected output, including whether duplicates should be removed, unambiguously specified?
4. Does the question yield a non-empty answer?
5. Do all filter conditions in the question constrain the result?

For each external knowledge entry, we check:

1. Is the external knowledge relevant to answering the question?
2. Is the external knowledge consistent with the question?

For each gold SQL query, we check:

1. Does the SQL query output all and only the required columns?
2. Does the SQL query handle ties correctly?
3. Does the SQL query handle duplicates correctly?
4. Does the SQL query handle missing values correctly?
5. Does the SQL query use the correct tables via appropriate joins and set operations?
6. Are all filter conditions from the question correctly translated?
7. Are aggregations computed over the correct partitions?
8. Does string-value extraction generalize across the full database, not just the sampled rows?

C EXAMPLES OF DATA ANNOTATION VERIFICATION AND CORRECTION

We illustrate the four error categories in Section 3.3: question and external knowledge errors, gold SQL query errors, schema errors, and grading-method errors. Each example explains the error and our correction or decision to discard the instance. We retain the original question wording; ellipses in SQL listings abbreviate unchanged clauses.

C.1 QUESTION AND EXTERNAL KNOWLEDGE ERRORS

Unanswerable questions. The following question asks whether products have been discontinued, but the database provides no attribute that records this status. We discard the instance because the available data cannot answer the question.

> **Question:** Among the products, how many of them were discontinued in production?
>
> **Action:** Discard the instance.

Empty results. When the target result is empty, an unrelated query such as `SELECT 1 WHERE FALSE` can receive full execution-based reward. We revise the question's filter conditions and the corresponding gold query to produce a non-empty target result.

Conflicting external knowledge. We align external knowledge with the question when their constraints disagree. For example, one question requires "at least 200 movies," but its external knowledge specifies `> 200`. We correct the condition to `>= 200` to include the boundary value.

C.2 GOLD SQL QUERY ERRORS

Mishandled missing data. The original query computes BMI extrema without excluding invalid measurements. We require positive weight and height values so that zero-valued placeholders for missing measurements do not distort the result.

Question: What are the basketball players' BMI ranges?

Original SQL:

```
SELECT
    MIN(weight * 1.0 / (height * height) * 703),
    MAX(weight * 1.0 / (height * height) * 703)
FROM players
```

Corrected SQL:

```
SELECT ... FROM ... -- same as the original
WHERE weight > 0 AND height > 0;
```

Unhandled ties and missing joins. The original query counts rows in `movies` rather than ratings and returns only one title. We join `movies` with `ratings`, count ratings for each movie, and return every movie whose count equals the maximum. This correction also preserves ties.

Question: Name the movies with the most ratings.

Original SQL:

```
SELECT movie_title FROM movies
GROUP BY movie_title
ORDER BY COUNT(movie_title) DESC LIMIT 1
```

Corrected SQL:

```
WITH cte AS (
    SELECT m.movie_title title, count(*) cnt
    FROM movies AS m JOIN ratings AS r
    ON m.movie_id = r.movie_id
    GROUP BY m.movie_id, m.movie_title )
SELECT title FROM cte
WHERE cnt = (SELECT MAX(cnt) FROM cte)
```

Incorrect predicates and duplicate counting. The original query uses a threshold of 10 likes and counts tweets, although the question asks for users who posted a tweet with more than 1,000 likes. We correct the threshold and count distinct user IDs so that each qualifying user contributes once, regardless of how many qualifying tweets they posted.

Question: Among all the users that have posted a tweet with over 1000 likes, how many of them are male?

Original SQL:

```
SELECT COUNT(T1.TweetID)
FROM twitter AS T1 INNER JOIN user AS T2
ON T1.UserID = T2.UserID
WHERE T1.Likes > 10 AND T2.Gender = 'Male'
```

Corrected SQL:

```
SELECT COUNT(DISTINCT T1.UserID)
FROM ... JOIN ... -- same as the original
WHERE T1.Likes > 1000 AND T2.Gender = 'Male'
```

Incorrect ranking criteria and invalid column references. The original query ranks products by `UnitsInStock`, which does not exist in the provided `Products` table. Stock levels also do not

measure the quantity ordered. We instead aggregate `Quantity` by product in `OrderDetails` and rank products by their total ordered quantity.

Question: List the product ID of the top five products, by descending order, the number of quantities ordered.

Original SQL:

```
SELECT ProductID FROM Products
ORDER BY UnitsInStock DESC LIMIT 5
```

Corrected SQL:

```
SELECT `ProductID` FROM `OrderDetails`
GROUP BY `ProductID`
ORDER BY SUM(`Quantity`) DESC LIMIT 5
```

C.3 Schema Errors

Incorrect value descriptions. The original documentation assigns the same value meanings to the `professor` and `student` columns, reversing the encoding of `professor`. We correct that column's description to match the data and retain the description of `student`.

Sample rows from `person`:

```
p_id|professor|student|...
   3|        0|      1|...
   4|        0|      1|...
```

Original value descriptions:

```
column, value_description
professor, [...] 0: professor. 1: student.
student, [...] 0: professor. 1: student.
```

Corrected value descriptions:

```
column, value_description
professor, [...] 1: professor. 0: student.
student, [...] 0: professor. 1: student.
```

C.4 Grading-Method Errors

Subset-based grading for any-k questions. A question such as "Please list the titles of any two papers that Jundu has written" admits multiple valid answers. Comparing the generated result with one fixed pair can reject a correct answer. We modify the gold query to return all eligible titles and accept a generated result if it contains exactly two distinct titles from that set. More generally, we require a subset of the gold result with the requested cardinality k.

List-based grading for ordered results. Set-based comparison ignores row order and can therefore accept results that violate an explicit ordering requirement. For the top-five-products question in Appendix C.2, we compare the generated and gold result tables cell by cell, requiring the same dimensions and values in the same positions. This check enforces the requested descending order.

D Examples of RLVR Reward Failure Modes

We illustrate the two failure modes diagnosed in Section 4.1: result coincidence that produces false positive rewards and failure to follow external knowledge. We retain the original questions and SQL excerpts; ellipses omit shared joins and filters.

D.1 False Positive Rewards from Result Coincidence

The question asks for the customer's average spending per order. The gold query first sums line-item prices within each order and then averages these order totals. The generated query instead averages individual line-item prices. These computations produce the same result on the example database because each of the customer's orders contains a single line item. Execution-based grading therefore rewards the generated query despite its incorrect aggregation.

Question: How much money on average does Lucas Wyldbore spend on book orders?

Gold SQL:

```
SELECT AVG(order_total)
FROM (
    SELECT T1.order_id, SUM(T1.price) AS order_total
    FROM order_line AS T1 JOIN cust_order As T2 ...
    GROUP BY T1.order_id
);
```

Incorrect generated SQL accepted by execution-based grading:

```
SELECT AVG(price)
FROM order_line AS T1 JOIN cust_order As T2 ...
-- same as the gold except for no GROUP BY
```

This example motivates the verification-aware outcome reward in Section 4.2: matching the gold result on one database does not establish semantic equivalence.

D.2 Failure to Follow External Knowledge

The external knowledge defines sodium-free recipes by the condition `sodium = 0`. The gold query follows this definition, whereas the generated query uses `sodium < 5`. The generated predicate admits recipes that the supplied definition excludes, illustrating how a model can substitute an alternative interpretation for an explicit domain constraint.

Question: Among the recipes from The California Tree Fruit Agreement, calculate the percentage of sodium-free recipes.

External knowledge: The California Tree Fruit Agreement is a source; calculation = `MULTIPLY(DIVIDE(COUNT(sodium = 0 THEN recipe_id), COUNT(recipe_id)), 100)`

Gold SQL:

```
SELECT CAST(SUM(CASE WHEN T2.sodium = 0 THEN 1 ELSE 0 END) AS REAL) *
    100 / COUNT(*) FROM ...
```

Generated SQL that violates the external knowledge:

```
SELECT CAST(SUM(CASE WHEN n.sodium < 5 THEN 1 ELSE 0 END) AS REAL) *
    100 / COUNT(*)FROM ...
```

An outcome reward can penalize this error when it changes the execution result, but it cannot distinguish the two predicates when they produce the same result. This limitation motivates the process supervision in Section 4.2, which requires the model to translate and verify each external knowledge entry explicitly.

E Training Algorithm and Implementation

We describe the training setup used to optimize the rewards in Section 4.2. We follow prior RLVR training practice (Liu et al., 2025a; Yu et al., 2025) and use the CISPO loss function (Chen et al., 2025a).

Prompt. We combine a shared system prompt with the problem's database schema, question, and external knowledge. We render the schema in data definition language and annotate columns with the descriptions provided by BIRD, following prior work (Li et al., 2025; Liu et al., 2025a). When a column description is unavailable, we include three randomly sampled values from that column. Appendix H provides the system prompts and input format.

Rollouts. During training, we sample multiple rollouts per problem. During both training and inference, the model can execute intermediate SQL queries to explore the database, test candidate queries, and refine its answer. We enforce a 30-second timeout for each intermediate query. The environment appends a turn counter to its execution responses to help the model plan within the interaction budget. We allow up to five turns per rollout; in pilot experiments, rollouts averaged three to four turns. Following prior work (Liu et al., 2025a), we limit each turn to 3,072 output tokens. Appendix H.3 specifies the tool interface and termination protocol.

Optimization and hyperparameters. We optimize the model with CISPO (Chen et al., 2025a) using LoRA with rank 32. We use a group size of 16, a batch size of 64, and a learning rate of 5×10^{-5}. For the composite reward in Equation 1, we set the verification penalty to $\beta = 0.2$ and each process penalty to $\lambda_k = 0.1$. Appendix H.4 specifies the reward computation and process-supervision rules.

Validation and checkpoint selection. We split BIRD-Platinum into training and validation sets at an 85:15 ratio. We determine the training duration from validation-accuracy convergence and evaluate the checkpoint with the highest validation accuracy on the benchmarks.

Infrastructure. We use Tinker (Tinker, 2025) as the training infrastructure for REVISQL. Our experimental setup includes a machine with an Intel 8568Y CPU and 512 GB of memory. We run open-source models on a single H100 GPU.

F BASELINE CONFIGURATIONS

We evaluate eight open-source pipelines selected from the BIRD leaderboard, four open-weight text-to-SQL models, GPT-5.6 Sol Ultra, and Claude Fable 5. We use each pipeline's highest-scoring leaderboard configuration when publicly available; otherwise, we follow the defaults in its technical report. For single-model baselines, we use the authors' prompt templates when available and the REVISQL prompt otherwise (Appendix H.2).

Choice of GPT-5.2. We use GPT-5.2 for the API-based generation and reasoning components specified below. Under otherwise identical pipeline configurations, replacing GPT-4o with GPT-5.2 improves execution accuracy on Arcwise-Plat-SQL from 74.1% to 82.9% for OpenSearch and from 68.5% to 81.7% for GenaSQL. We also use GPT-5.2 for MRS to maintain consistency with OpenSearch and GenaSQL. This choice strengthens the evaluated baselines while retaining token pricing comparable to GPT-4o. GPT-5.2 has a lower input-token price and a slightly higher output-token price than GPT-4o, while higher-priced flagship alternatives, such as GPT-5.4 and GPT-5.5, charge more than GPT-5.2 for both input and output tokens.[1]

F.1 OPEN-SOURCE PIPELINES

Contextual-SQL. Contextual-SQL (Agrawal & Nguyen, 2025) uses XiYanSQL-32B (Liu et al., 2025d) to generate candidate queries and a fine-tuned 32B reward model to select the final query. The reward model was trained on the full BIRD Train set of approximately 9.5k instances.

CSC-SQL. CSC-SQL (Sheng & Xu, 2025a) generates candidates with the fine-tuned XiYanSQL-QwenCoder-32B-2412 model and selects the final query with Qwen2.5-Coder-7B-Instruct.

GenaSQL. GenaSQL (Dönder et al., 2025) combines GPT-4o, Gemini 1.5, and Cohere for schema linking, retrieval, and generation. We retain its workflow, use OpenAI's text-embedding-3-small for retrieval, and use GPT-5.2 for the subsequent orchestration and generation steps.

[1] Official model descriptions and token prices: `https://developers.openai.com/api/docs/models/gpt-5.2`, `https://developers.openai.com/api/docs/models/gpt-4o`, `https://developers.openai.com/api/docs/models/gpt-5.4`, and `https://developers.openai.com/api/docs/models/gpt-5.5`.

OpenSearch. OpenSearch (Xie et al., 2025) combines retrieval with query generation and refinement. We retain bge-large-en-v1.5 for few-shot example retrieval and replace GPT-4o with GPT-5.2 for schema extraction, query generation, and refinement.

SHARE. SHARE (Qu et al., 2025) refines an input SQL query using three distilled models specialized in reasoning, schema linking, and iterative query repair. We retain these refinement models and apply the pipeline to initial queries generated by GPT-5.2.

DeepEye-SQL. DeepEye-SQL (Li et al., 2026) organizes text-to-SQL as a software engineering workflow comprising value retrieval, schema linking, candidate generation, revision, and selection. It checks syntax, execution, and query results to repair candidates, then uses database feedback to select the final query. We use its default settings.

AxisSQL. AxisSQL (Shareef, 2026) extends the DeepEye-SQL pipeline by scaling candidate generation, iterative revision, and execution-based aggregation at inference time. Its aggregation agent executes and compares candidate queries and can synthesize a new query by combining their parts. We use its default settings.

MRS agent. MRS (Chan et al., 2026) decomposes each question into semantic components, scores table relevance in parallel, and aggregates these scores to identify a focused schema. It generates candidates using different schema and metadata contexts, refines them with execution feedback, and uses an LLM judge to select the final query. We use GPT-5.2 for its generation and reasoning components.

F.2 Single-Model Baselines

OmniSQL-32B. OmniSQL-32B (Li et al., 2025) fine-tunes Qwen2.5-Coder-32B-Instruct on 2.5 million synthetic examples together with Spider Train and BIRD Train.

Infly-RL-SQL-32B. Infly-RL-SQL-32B (Infly, 2025) does not publicly document its fine-tuning process or detailed usage instructions. We use the same inference setup as OmniSQL-32B.[2]

Kwai-AutoSQL-32B. Kwai-AutoSQL-32B (Kwai, 2026) fine-tunes Qwen3-32B for text-to-SQL. Its public release provides model weights but no training or inference scripts. We evaluate the released checkpoint using the ReViSQL prompt.

Jiayin-Pangu-Text2SQL-14B. Jiayin-Pangu-Text2SQL-14B (Research, 2026) provides an open-weight text-to-SQL model without a model card documenting its training procedure or recommended inference setup. We evaluate the released `Jiayin-Pangu-Text2Sql-14B-135` checkpoint using the ReViSQL prompt.

GPT-5.6 Sol Ultra and Claude Fable 5. We also evaluate GPT-5.6 Sol Ultra and Claude Fable 5 as standalone reasoning models using the same inference settings as our models.

G Generalization to the Snowflake SQL Dialect

We evaluate the fine-tuned Qwen3-235B-A22B on Spider2-Snow to assess transfer from the SQLite queries in BIRD-Platinum to the Snowflake dialect. We apply the inference-time adaptations described in Section 5.3 without retraining the model.

Comparison with open-weight pipelines. With self-consistency over 128 candidates, the fine-tuned Qwen3-235B-A22B achieves 55.58% accuracy on Spider2-Snow (Table 4). It outperforms several agentic pipelines built on larger models: ReFoRCE with DeepSeek-V3 (671B), AutoLink with DeepSeek-R1 (671B), and Spider-Agent with Qwen3-Coder (480B). Since Spider2-Snow is held out from training, our results support transfer to a new SQL dialect under the stated inference-time adaptations.

[2] We requested usage instructions from the authors but had not received a response at the time of evaluation.

Table 4: Fine-tuned Qwen3-235B-A22B on the Spider2-Snow Leaderboard among open-source methods with open-weight models, as of May 1, 2026. Trained on BIRD-Platinum, a 235B model outperforms open-source methods built on bigger models.

Method	# param.	Accuracy (%)
Spider-Agent (QwQ-32B)	32B	8.96
Spider-Agent (DeepSeek-R1)	671B-A37B	10.55
Spider-Agent (Qwen3-Coder)	480B-A35B	31.08
ReFoRCE (DeepSeek-V3)	671B-A37B	38.03
AutoLink (DeepSeek-R1)	671B-A37B	54.84
Fine-tuned Qwen3-235B-A22B	**235B-A22B**	**55.58**

H Prompts and Rollout Specification

This appendix specifies the exact prompts and rollout mechanics used for RL with execution-based outcome rewards and ReViSQL, during both RLVR training and inference. RL with execution-based outcome rewards uses the base system prompt of §H.1, while ReViSQL extends it with the two external-knowledge use directives of §H.2. The ReViSQL prompt is also what we give the single-model baselines that have not published their own system prompts (Section 5.1).

H.1 System Prompt for RL with execution-based outcome rewards

We use a universal system prompt for all text-to-SQL problems, across all benchmarks and both SQL dialects. It defines the task, the input format, the output contract, and the multi-turn protocol.

```
Task Overview:
You are a data science expert. Below, you are provided with a database schema and a natural
      language question. Your task is to understand the schema and generate a valid SQL
     query to answer the question within limited turns. You should breakdown the problem,
     draft your reasoning process, and generate the solution.

The task you will receive should have the following format:

Database Engine:
{engine}

Database Schema:
<<db_details>>
This schema describes the database's structure, including tables, columns, primary keys,
     foreign keys, and any relevant relationships or constraints.

External Knowledge:
<<external_knowledge>>

Question:
<<question>>

Instructions:
- Make sure you only output the information that is asked in the question. If the question
     asks for a specific column, make sure to only include that column in the SELECT clause
     , nothing more.
- The generated query should return all of the information asked in the question without
     any missing or extra information.
- If the external knowledge is not empty, you must use all the information provided in the
     external knowledge to help you generate the SQL query.
- Before generating the final SQL query, please think through the steps of how to write the
      query. It should include detailed considerations such as analyzing questions,
     summarizing relevant findings, brainstorming new ideas, verifying the accuracy of the
     current steps, refining any errors, thinking of how to call SQL tools, and revisiting
     previous steps.

Format:
- Conduct thinking every time you get new observation or information.
- You can use the SQL tool to explore data or verify result. You will receive the execution
      results or error information of the SQL from a user. Based on this information, you
     can think again and refine.
- The returned dataframe will be truncated in 50 rows if observation is too long.
```

```
- Only if you find no further exploration is needed or reach max turns, you directly
    provide the final SQL query solution inside <solution>...</solution>.
- Do not request a SQL tool execution and provide a solution in the same response.
```

H.2 EVIDENCE-STRUCTURED PROMPT (REVISQL)

REVISQL uses the same system prompt with two additional directives in the `Format` section, which operationalize the process supervision discussed in §4.2. Every other part of the prompt is identical to §H.1.

```
- In the first or second turn, you must perform a translation from the external knowledge
    to requirements of the final SQL query. The external knowledge may contain multiple
    pieces of information separated by semicolons. You must list the requirement for each
    external knowledge piece the following format: "Requirement of external knowledge {i}
    ({raw external knowledge text}): ...". This will help you better understand the
    question and the external knowledge, and guide your thinking in the subsequent turns.
    If the external knowledge is empty, skip this translation step.

- In the response where you produce the final solution inside <solution>...</solution>, you
     must first verify the external knowledge one by one and conduct a final check to
    ensure all the external knowledge is correctly used in the generated SQL query. You
    must use the following format to verify each piece of external knowledge: "
    Verification of external knowledge {i} ({raw external knowledge text}): ...". If you
    find any issues during the verification, you must update your draft before finalizing
    the query inside <solution>. If the external knowledge is empty, skip this
    verification step.
```

Relation to process supervision. The *requirement* directive instructs the model to translate each external-knowledge entry into an explicit query constraint before it writes SQL. The *verification* directive instructs it to audit the generated query against each entry, and to revise the draft if the audit fails, in the same response that commits the answer. Because both blocks are emitted in a fixed, indexed format, we check the compliance by a string-level test on the reasoning trace, with no learned reward model in the loop. §H.4 specifies the resulting rule set $\mathcal{K}$ and its penalties.

H.3 ROLLOUT SPECIFICATION

A rollout is a multi-turn interaction between the model and a database environment. The model is given a tool, `execute_sql_query`, which executes a SQL query against the problem's database and returns the result.

Intermediate turns. On each intermediate turn the model emits its reasoning followed by a tool call. The environment executes the query with a 30-second timeout and replies with one of the following states.

1. *Success.* The result table, rendered as text. To bound the context, the table is truncated to its first 50 rows and individual cells are truncated to 200 characters. We explicitly tell the model when truncation occurred.
2. *Execution error.* The database error message, so that the model can diagnose and repair its query.
3. *Malformed action.* If the tool call cannot be parsed, the environment returns an instruction to re-emit a tool call.

To aid planning, the environment appends a turn counter to every successful response: ``` ``You have {k} turns left to complete the task.'' ``` The model may not request a tool execution and submit a final solution in the same response.

Terminal turn. The model terminates a rollout by emitting its final query inside `<solution>...</solution>`, which it is instructed to do only when further exploration is unnecessary or the turn budget is exhausted. A response that violates the output contract receives a reward of -1.

H.4 REWARD SPECIFICATIONS

RL with execution-based outcome rewards is trained with the base prompt; the verification-aware stage and the process penalties are specific to REVISQL. The reward for a rollout is computed as follows.

1. **Format check.** A solution that does not satisfy the output contract receives -1.
2. **Execution-based grading.** We execute the generated query and the gold query against the database and compare their results using the grading method annotated for that instance—set-based, subset-based, or list-based (§3.3). A mismatch yields reward 0. This is the only reward term used by execution-based outcome rewards.
3. **Verification-aware grading (REVISQL only).** If result-based grading accepts the query, we invoke VeriEQL He et al. (2024) with a 30-second timeout to test semantic equivalence to the gold query. If VeriEQL refutes equivalence, the reward is downweighted to $1 - \beta = 0.8$; if VeriEQL confirms equivalence, times out, or does not support the query, the reward is 1. This stage is applied to SQLite problems, the dialect VeriEQL supports. We skip VeriEQL verification for any-k questions because a valid subset need not be equivalent to the gold query.
4. **Process penalties (REVISQL only).** We deduct $\lambda_k = 0.1$ for each violated rule $k \in \mathcal{K}$, as specified below.

Evidence-supervision rules. The rule set $\mathcal{K}$ of Equation 1 contains two rules, each carrying $\lambda_k = 0.1$. Both are evaluated by string-level checks on the reasoning trace.

1. **Requirement rule.** The requirement block must appear by the second turn. We evaluate this rule once per rollout, at turn two, and deduct 0.1 if the block is absent. Because it is checked before the model commits an answer, the penalty applies whether or not the rollout ultimately produces a solution.
2. **Verification rule.** The verification block must appear in the same response as the final `<solution>`. We evaluate this rule when the model commits a solution and deduct 0.1 if the block is absent.

We relax both rules in a special case when the model answers on the first turn, which happens on easy problems that require no exploration. In this case, demanding a separate translate-then-verify pass is unnecessary, so we deduct 0.1 only if *both* blocks are missing, and nothing if either is present. A rollout therefore incurs at most 0.2 in process penalties, and at most 0.1 when it terminates on its first turn. Problems that carry no external knowledge require neither block.