TRUST-SQL: Tool-Integrated Multi-Turn Reinforcement Learning for Text-to-SQL over Unknown Schemas

Anonymous ACL submission

Abstract

Text-to-SQL parsing has achieved remarkable progress under the **Full Schema Assumption**, where the complete database structure is pre-loaded into the model's context. Yet this assumption overlooks a more fundamental capability question of whether an LLM agent can resolve a query without explicit access to the underlying tables and relations. Moreover, full-schema injection may exceed the effective context window, introduce irrelevant metadata into the reasoning process, and become less reliable as databases evolve. We therefore study the **Unknown Schema** setting, in which an agent must actively explore the database and commit to only the relevant subset of metadata. To address this, we propose **TRUST-SQL** (**T**ruthful **R**easoning with **U**nknown **S**chema via **T**ools). We formulate the task as a **Partially Observable Markov Decision Process** in which our autonomous agent follows a structured **four-phase protocol** to ground reasoning in verified metadata. This protocol provides a structural boundary for our novel **Phase-Aware GRPO** strategy, which applies token-level masked advantages to isolate exploration rewards from execution outcomes and resolve credit assignment, yielding a **9.9% relative improvement** over standard GRPO. Extensive experiments across five benchmarks demonstrate that TRUST-SQL is effective at **multiple scales (4B/ 8B/14B)**. Despite operating entirely without pre-loaded metadata, our agent **matches or surpasses prior baselines on most benchmarks** that rely on schema prefilling.[1]

1 Introduction

Text-to-SQL parsing, which translates natural language questions into executable SQL queries, has seen remarkable progress driven by Large Language Models (LLMs) (Shkapenyuk et al., 2025; Wang et al., 2025b). However, much of this progress relies on a critical yet underexplored premise known as the **Full Schema Assumption**, where the complete database structure is pre-loaded into the model's context. Under this setting, the task is largely framed as a *static translation* problem, and existing methods report strong performance on benchmarks with pre-injected database structures (Li et al., 2024; Yu et al., 2018).

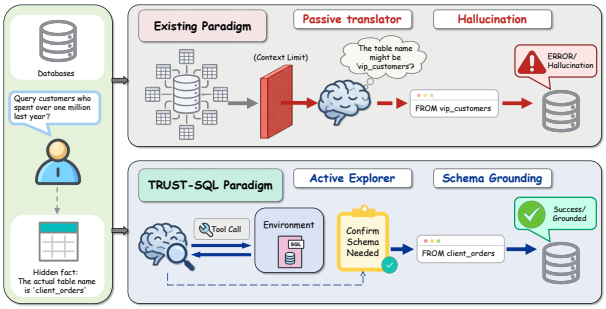


Figure 1: Existing methods rely on pre-loaded schemas, while the Unknown Schema setting requires exploration.

This assumption is both empirically fragile and conceptually restrictive, as full-schema injection introduces substantial irrelevant metadata into the prompt as database structures grow, increasing contextual burden and potentially degrading reasoning quality, with important information often buried in the middle of the prompt (Laban et al., 2025). Meanwhile, real-world databases continuously evolve through additions, deletions, and structural modifications (Zhang et al., 2026), making static representations increasingly incomplete over time. More importantly, this paradigm avoids a deeper capability question of whether an LLM agent can resolve a database query without direct access to the full database structure. As illustrated in Figure 1, we define this missing setting as the **Unknown Schema** scenario, where an agent must move beyond passive consumption and actively explore the database to retrieve only the metadata necessary for the query.

[1] Training data and model checkpoints will be released after the anonymity period.

Once the schema is no longer pre-loaded, standard single-turn methods break down, as they lack any mechanism for active retrieval under partial observability. The task must instead be formulated as a **multi-turn tool-integrated decision-making process**. Recent agentic frameworks have explored this iterative setting, yet two key bottlenecks remain unresolved. ***Architecturally***, LLMs struggle to maintain coherent reasoning over long interaction horizons. Without explicit grounding mechanisms, they often lose track of intermediate observations (Laban et al., 2025) and fall back on hallucinating schema elements from parametric priors. ***Algorithmically***, credit assignment across long interaction trajectories remains a fundamental challenge for large language models (Zhou et al., 2025; Yang et al., 2026). Existing approaches typically rely on a single terminal reward (Yang et al., 2025; Xu et al., 2025) or naive aggregation of intermediate signals (Hua et al., 2026), which conflates schema exploration quality with SQL generation quality and makes it difficult to attribute final execution outcomes to specific actions.

In this paper, we propose **TRUST-SQL** (**T**ruthful **R**easoning with **U**nknown **S**chema via **T**ools) to systematically address these challenges. We formulate the task as a Partially Observable Markov Decision Process and introduce a **four-phase interaction protocol** comprising *Explore*, *Propose*, *Generate*, and *Confirm*. The *Propose* phase acts as a mandatory cognitive checkpoint that forces the agent to commit to verified metadata before generation, suppressing downstream hallucination. This checkpoint also provides a structural boundary for **Phase-Aware GRPO**, a training strategy built upon Group Relative Policy Optimization (GRPO) (DeepSeek-AI, 2025) that applies token-level masked advantages to separately credit schema exploration and SQL execution, co-optimizing schema grounding and SQL generation.

Our contributions are summarized as follows:

- We formalize the **Unknown Schema** setting as a partial-observability testbed for agentic Text-to-SQL, and propose **TRUST-SQL**, an autonomous framework that interacts directly with unobservable databases to retrieve and verify metadata, supporting grounded SQL generation via iterative exploration instead of static context.
- We propose **Phase-Aware GRPO**, a novel training strategy utilizing token-level masked advantages and execution-coupled schema rewards. This granular optimization yields a **9.9% relative improvement** in execution accuracy over standard GRPO on BIRD-Dev.
- Extensive experiments across five diverse benchmarks demonstrate that TRUST-SQL is **consistently effective across the 4B, 8B, and 14B scales**. Despite operating without pre-loaded metadata, our models match or surpass baselines that rely on schema injection on most benchmarks.

2 Related Work

Text-to-SQL under Full Schema Assumption. Most existing methods operate under the premise of full schema observability. Supervised fine-tuning approaches such as OmniSQL (Li et al., 2025), STAR (He et al., 2025), and ROUTE (Qin et al., 2024) internalize generation capabilities but rely on static context. Similarly, single-turn reinforcement learning (RL) methods (Ma et al., 2025; Yao et al., 2025; Zhang et al., 2025; Pourreza et al., 2025) optimize execution accuracy using terminal rewards while assuming access to the complete database structure. Constrained to a single-turn interaction paradigm, these models function as passive translators and struggle under Unknown Schema setting requiring active database exploration.

Tool-Augmented Approaches for Text-to-SQL. To handle complex or hidden databases, recent works introduce tool-augmented methods. Training-free frameworks (Wang et al., 2024, 2025a,d) leverage frozen language models to query metadata, but remain susceptible to parametric hallucinations and lack reliable verification mechanisms. More recently, multi-turn RL approaches (Xu et al., 2025; Hua et al., 2026; Guo et al., 2025) incorporate SQL execution into the training loop to refine queries. While promising, these methods lack clear separation between metadata retrieval and SQL generation, relying on conflated terminal rewards that fail to distinguish errors from schema retrieval and SQL generation.

Credit Assignment in Multi-Turn RL. A central challenge in multi-turn RL is attributing final outcomes to individual actions in long trajectories. Existing solutions explore trajectory-level optimization (Wang et al., 2025c; Xue et al., 2025), reward decomposition (Liu et al., 2025), tree-structured search (Ji et al., 2025), and intrinsic motivation (Kumar et al., 2024; Wan et al., 2025). These techniques are primarily designed for homogeneous

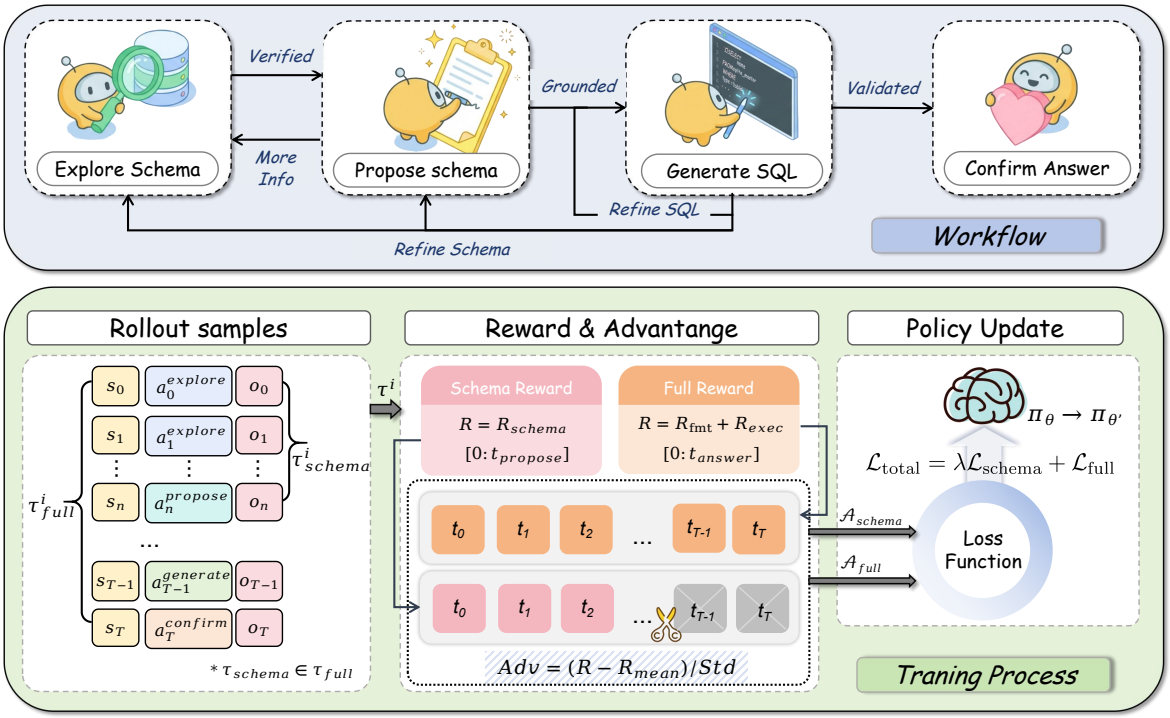


Figure 2: Overview of the **TRUST-SQL** framework. **(Top)** The four-phase workflow comprising *Explore*, *Propose*, *Generate*, and *Confirm*, with non-linear transitions enabling iterative schema refinement. **(Bottom)** The Phase-Aware GRPO training pipeline, where trajectories are decomposed into a Schema Track τ_{schema} and a Full Track τ_{full}, each optimized with independent rewards and masked advantages.

action spaces where each step contributes similarly to the final objective. However, Text-to-SQL introduces a heterogeneous decision process where schema retrieval and SQL generation play distinct roles, making standard credit assignment formulations insufficient. TRUST-SQL addresses this by introducing Phase-Aware GRPO to disentangle optimization across phases.

3 Methodology

We present **TRUST-SQL** for Text-to-SQL over unknown schemas. As illustrated in Figure 2, it consists of a four-phase interaction protocol and Phase-Aware GRPO training strategy. We formulate the task as a sequential decision-making process, followed by reward design and RL optimization.

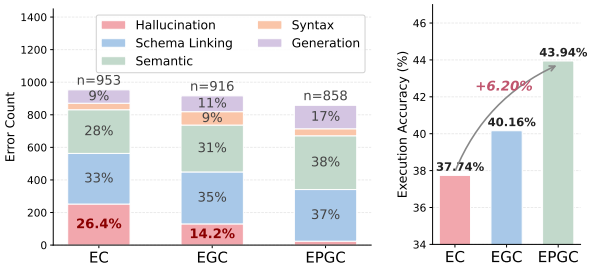
3.1 Motivation: Why a Four-Phase Protocol?

To empirically justify our interaction protocol design and identify key bottlenecks of Text-to-SQL under the **Unknown Schema setting**, we conduct a pilot study on the **BIRD-Dev** dataset with **Qwen3-8B** as the base model. We construct **three agent variants** with incremental structural constraints on interaction behavior and classify failure cases to derive design principles for our framework.

Protocol Variants. **EC** (Explore-Confirm) is a minimal baseline where the agent freely queries metadata and directly submits a SQL answer without intermediate verification. **EGC** (Explore-Generate-Confirm) introduces an **explicit Generate phase**, requiring the agent to execute a candidate SQL and observe its result before finalizing. **EPGC** (Explore-Propose-Generate-Confirm) further adds **the Propose phase** as a mandatory cognitive checkpoint, compelling the agent to commit to a verified schema before SQL generation.

Error Taxonomy. We classify failures into five categories: **(1) Hallucination**: the model fabricates non-existent tables or columns based on parametric priors; **(2) Schema Linking**: the model selects wrong or missing tables and columns despite correct exploration; **(3) Semantic**: the model correctly identifies the relevant schema but generates logically incorrect SQL; **(4) Syntax**: the SQL contains malformed statements that fail to execute; **(5) Generation**: the agent fails to produce a complete SQL, typically due to reaching the maximum turn limit.

As shown in Figure 3, three observations emerge from the results.



(a) Stacked error distribution. (b) EX accuracy.

Figure 3: Pilot study results on BIRD-Dev (Qwen3-8B).

Obs. 1: Schema verification is critical to suppress hallucination. In EC, hallucination accounts for 26.4% of all failures. The *Generate* phase partially alleviates this via execution feedback (14.2%), but the most significant reduction occurs with the *Propose* in EPGC, driving hallucination to just 2.8%, a **9.4× reduction** over EC.

Obs. 2: Schema linking is the persistent bottleneck. Schema linking errors remain consistently high across all variants, motivating our Phase-Aware GRPO to provide an independent optimization signal for schema exploration.

Obs. 3: Suppressing hallucination reveals semantic errors. As hallucination decreases, semantic errors increase from 268 to 330, reflecting a distributional shift: once schema is correctly identified, complex query logic becomes the dominant challenge, motivating joint optimization of schema grounding and SQL generation.

These observations motivate the core designs of our work: the *Propose* checkpoint to suppress hallucination, and Phase-Aware GRPO to co-optimize schema exploration and SQL generation. The *Propose* checkpoint contributes at two levels. At the workflow level, it grounds the agent's reasoning in verified metadata. At the optimization level, it provides a credit-assignment boundary for Phase-Aware GRPO. We disentangle these two effects in Appendix B.4.

3.2 Problem Formulation

Based on the EPGC protocol validated in Section 3.1, we formalize the Text-to-SQL task under the Unknown Schema setting as a Partially Observable Markov Decision Process **(POMDP)**, which is defined as $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{Z}, \gamma)$ over discrete steps $t = 0, 1, \ldots, T$.

State and Observation Spaces. The true environment state $s_t \in \mathcal{S}$ represents the complete database schema and remains hidden from the agent. Consequently, the agent only receives partial observations $o_t \in \Omega$ dictated by the observation function $\mathcal{Z}$, which consist of tool execution feedback. To navigate this unobservable environment, the agent relies on an internal context state $c_t = (q, h_t, \mathcal{K}_t)$. This context integrates the user question q, the interaction history h_t, and the **Verified Schema Knowledge** $\mathcal{K}_t$, which stores only explicitly verified metadata and initializes as $\mathcal{K}_0 = \emptyset$.

Action Space. To prevent hallucination, the agent selects actions $a_t \in \mathcal{A}$ from four strict categories based on its current context c_t. The *Explore* action queries database metadata. The *Propose* action serves as a mandatory cognitive checkpoint at step t_{propose} to commit to the verified schema $\mathcal{K}_{t_{\text{propose}}}$. The *Generate* action produces a candidate SQL grounded in $\mathcal{K}_t$, and the *Confirm* action submits the final SQL query y at the terminal step T.

Transition and Objective. Upon executing a_t, the environment emits observation o_t and the agent updates its context state to c_{t+1}. A complete interaction sequence from the agent's perspective is represented as a trajectory $\tau = \{(c_t, a_t, o_t)\}_{t=0}^{T}$. The ultimate goal of the policy $\pi_\theta(a_t \mid c_t)$ is to maximize the expected cumulative reward $J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[\sum_{t=0}^{T} \gamma^t r(c_t, a_t)]$.

3.3 Reward Components

To evaluate the trajectory, we define three distinct reward signals. The specific mechanism for assigning these signals to individual tokens is detailed in Section 3.4.

Execution Reward (R_{exec}). This reward compares the result $\texttt{Exec}(y)$ of the predicted SQL against the ground truth $\texttt{Exec}(y^*)$, with $\perp$ denoting that a query fails to execute. The reward is assigned as

$$R_{\text{exec}}(y, y^*) = \begin{cases} 1.0 & \text{if } \texttt{Exec}(y) = \texttt{Exec}(y^*) \\ 0.2 & \text{if } \texttt{Exec}(y) \neq \perp \\ 0.0 & \text{if } \texttt{Exec}(y) = \perp \end{cases} \tag{1}$$

where the middle case provides partial credit for SQL that executes but yields an incorrect result.

Format Reward (R_{fmt}). This constitutes a trajectory-level signal requiring consistent protocol adherence. The reward is defined as

$$R_{\text{fmt}}(\tau) = \begin{cases} 0.1 & \text{if protocol is fully adhered to} \\ 0.0 & \text{otherwise} \end{cases} \tag{2}$$

Full adherence requires that every action a_t conforms to prescribed format, all four action categories in $\mathcal{A}$ appear at least once, and no execution errors occur in the observations o_t.

Schema Reward (R_{schema}). This reward evaluates the quality of the schema exploration phase. It is computed as

$$R_{\text{schema}}(\hat{\mathcal{K}}, \mathcal{K}^*) = f_{\text{match}}(\hat{\mathcal{K}}, \mathcal{K}^*) \tag{3}$$

where $\hat{\mathcal{K}}$ represents the schema proposed by the agent at step t_{propose}, and $\mathcal{K}^*$ represents the minimal ground truth schema extracted from y^*. The function f_{match} evaluates their structural overlap. We adopt the sparse execution-coupled form by default; see Appendix B.2 for the full definition of f_{match} and its variants.

3.4 Resolving Credit Assignment via Phase-Aware GRPO

Standard RL combines exploration and generation under a single reward, making it hard to attribute success or failure to specific actions in long trajectories. We thus leverage the structural boundary of the *Propose* checkpoint to introduce **Phase-Aware GRPO**, extending Group Relative Policy Optimization to clearly separate the learning signals for schema grounding and SQL generation. In this work, we instantiate Phase-Aware GRPO with a two-track decomposition (Schema Track and Full Track); the same formulation generalizes to any number of structural checkpoints in longer agentic workflows. A step-by-step contrast against standard GRPO is provided in Appendix B.3.

Track Formulation and Rewards. For each question q, we sample G trajectories and divide each τ^i into two optimization tracks $k \in \{\text{schema}, \text{full}\}$, where the Schema Track ends at $T_{\text{schema}} = t_{\text{propose}}$ and the Full Track spans the entire interaction up to $T_{\text{full}} = T$. Here t_{propose} is defined per rollout as the step of the **latest** *Propose* action in the trajectory (Appendix B.1). A dedicated reward R_k^i is assigned to each track

$$R_k^i = \begin{cases} R_{\text{schema}}(\hat{\mathcal{K}}_i, \mathcal{K}^*) & \text{if } k = \text{schema} \\ R_{\text{exec}}(y_i, y^*) + R_{\text{fmt}}(\tau^i) & \text{if } k = \text{full} \end{cases} \tag{4}$$

ensuring an independent optimization signal for exploration quality regardless of generation errors.

Masked Advantage Computation. Advantages are computed via group-relative normalization within each track

$$A_k^i = \frac{R_k^i - \mu_k}{\sigma_k + \epsilon} \tag{5}$$

where μ_k and σ_k are the mean and standard deviation of the group rewards. We apply strict token-level masking where the advantage A_k^i is broadcast exclusively to tokens generated within the active steps $t \in [0, T_k]$. This is strictly finer-grained than trajectory-level weighting, as it prevents exploration rewards from incorrectly crediting generation tokens and vice versa. Consequently, tokens generated after the *Propose* checkpoint receive zero schema advantage.

Phase-Aware Loss Function. Let $\mathcal{L}_k(\theta)$ denote the GRPO loss computed over the active tokens for track k using the masked advantage A_k^i. The total objective combines both tracks

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{full}}(\theta) + \lambda \cdot \mathcal{L}_{\text{schema}}(\theta) \tag{6}$$

where λ controls the relative contribution of the Schema Track. By unifying these components, Phase-Aware GRPO co-optimizes schema grounding and SQL generation without mixing their learning signals.

4 Experiments

4.1 Experimental Setup

Implementation Details. We adopt Qwen3-4B and Qwen3-8B as our base models and implement all experiments using the SLIME framework (Zhu et al., 2025), trained in two sequential stages of SFT warm-up followed by Phase-Aware GRPO optimization. An ablation on the necessity of the SFT warm-up stage is provided in Appendix E.1. Further details are provided in Appendix C.

Baselines. TRUST-SQL utilizes a highly efficient data recipe comprising 9.2k SFT samples and 11.6k RL samples. We compare our framework against recent strong baselines across the 3B to 8B parameter scales. Single-turn models include OmniSQL (Li et al., 2025) and SQL-R1 (Ma et al., 2025). Multi-turn RL methods include MTIR-SQL (Xu et al., 2025) and SQL-Trail (Hua et al., 2026). Full dataset construction and detailed baseline comparisons are provided in Appendix A.

Evaluation Benchmarks and Metrics. We evaluate on BIRD-Dev (Li et al., 2024) for large-scale schema grounding and Spider-Test (Yu et al.,

Table 1: Execution Accuracy (EX%) across multiple benchmarks. **Gre** denotes single-sample performance; **Maj** denotes majority voting. **Bold** indicates the best result and underline indicates the second best within each group.

Method	Schema Prefill	BIRD (dev)		Spider (test)		Spider-DK		Spider-Syn		Spider-Realistic	
		Gre	Maj	Gre	Maj	Gre	Maj	Gre	Maj	Gre	Maj
3B – 4B Models											
SQL-R1-3B	✓	–	54.6	–	78.9	–	70.5	–	66.4	–	71.5
SQL-Trail-3B	✓	50.1	55.1	77.7	84.3	–	–	–	–	–	–
MTIR-SQL-4B	✓	63.1	64.4	83.4	–	71.2	–	**78.6**	–	78.7	–
TRUST-SQL-4B	×	**64.9**	**67.2**	**82.8**	**85.0**	**71.6**	**73.8**	74.7	**77.3**	**79.9**	**82.5**
7B – 8B Models											
OmniSQL-7B	✓	63.9	66.1	**87.9**	**88.9**	76.1	77.8	69.7	69.6	76.2	78.0
SQL-R1-7B	✓	63.7	66.6	–	88.7	–	**78.1**	–	76.7	–	83.3
SQL-Trail-7B	✓	60.1	64.2	86.0	87.6	**76.8**	77.8	72.8	77.0	79.6	83.9
MTIR-SQL-8B	✓	–	64.6	83.4	–	72.9	–	77.2	–	77.4	–
TRUST-SQL-8B	×	**65.8**	**67.7**	83.9	86.5	72.1	75.7	**75.4**	**77.4**	**82.1**	**84.1**
14B Models											
OmniSQL-14B	✓	64.2	65.9	**88.3**	88.3	72.9	74.8	69.0	72.0	76.4	78.5
SQL-R1-14B	✓	–	67.1	–	88.1	–	**79.3**	–	78.5	–	86.2
SQL-Trail-14B	✓	63.6	66.7	86.8	**88.5**	–	–	–	–	–	–
MTIR-SQL-14B	✓	67.2	68.1	87.2	–	**76.3**	–	**81.0**	–	81.1	–
TRUST-SQL-14B	×	**68.1**	**70.1**	83.9	86.5	75.0	78.6	78.4	**79.7**	**82.7**	**86.5**

2018) for compositional generalization. To stress-test model robustness, we incorporate three challenging variants. Specifically, Spider-Syn (Gan et al., 2021a) evaluates lexical robustness via synonym substitution, Spider-DK (Gan et al., 2021b) probes for implicit domain knowledge, and Spider-Realistic (Deng et al., 2021) assesses ambiguity resolution. We measure Execution Accuracy where the predicted SQL must yield the exact same database result as the ground truth. We report single-sample performance via Greedy decoding at temperature zero and execution-based Majority voting across multiple sampled queries.

4.2 Main Results

Table 1 presents the execution accuracy across all benchmarks. For the majority voting evaluation, we sample trajectories at a temperature of 0.8 with a 15-turn inference budget, as analyzed in Section 5.2. Detailed token consumption and tool invocation statistics are provided in Appendix D.1.

Scaling Across 4B / 8B / 14B. TRUST-SQL performs competitively at all three scales while operating under the Unknown Schema setting. On BIRD-Dev, the 4B / 8B / 14B variants reach 64.9 / 65.8 / 68.1 with greedy decoding and 67.2 / 67.7 / 70.1 with majority voting; TRUST-SQL-14B outperforms every 14B baseline that uses full schema prefilling. On the robustness suite, TRUST-SQL leads Spider-Realistic at all three scales (82.5 / 84.1 / 86.5 majority) and Spider-Syn majority voting at all three scales (77.3 / 77.4 / 79.7); on Spider-DK, the 4B variant tops its group (71.6 / 73.8) while the 8B and 14B variants remain within a few points of the schema-prefilled best. On the standard Spider-Test set, where schema-prefilled baselines benefit from explicit mapping cues, TRUST-SQL remains within a few points without any pre-loaded schema. These results suggest that the four-phase protocol and Phase-Aware GRPO continue to provide gains as scale grows.

The Value of Autonomous Exploration. TRUST-SQL achieves these scores under the strict Unknown Schema setting, while all baselines rely on full schema prefilling that consumes substantial context and assumes perfect database observability. Further comparisons against strong proprietary agentic models and a serving-cost analysis are reported in Appendices E.5 and D.2.

4.3 Can Schema Prefill Boost Performance?

While TRUST-SQL operates without any pre-loaded schema, a natural question arises as to whether injecting the complete schema would further boost performance. We thus introduce a Schema Prefill variant where the full schema is delivered as a single synthetic *Explore* turn at the beginning of the interaction, providing all table

and column information at once. The case study is shown in Appendix G.

Table 2: Effect of Schema Prefill (greedy decoding). Arrows denote absolute accuracy changes compared to the Unknown Schema (×) setting.

Prefill	BIRD	Spider	S-DK	S-Syn	S-Realistic
Qwen3-4B					
×	29.3	51.2	43.7	47.4	49.2
✓	46.3 ↑17.0	67.6 ↑16.4	57.0 ↑13.3	62.6 ↑15.2	65.9 ↑16.7
TRUST-SQL-4B					
×	64.9	82.8	71.6	74.7	79.9
✓	64.8 ↓0.1	83.1 ↑0.3	69.2 ↓2.4	72.5 ↓2.2	80.1 ↑0.2
Qwen3-8B					
×	47.9	67.4	56.3	58.4	66.3
✓	49.9 ↑2.0	68.3 ↑0.9	57.6 ↑1.3	64.5 ↑6.1	68.1 ↑1.8
TRUST-SQL-8B					
×	65.8	83.9	72.1	75.4	82.1
✓	65.5 ↓0.3	84.0 ↑0.1	74.4 ↑2.3	75.4	80.5 ↓1.6

As shown in Table 2, the base Qwen3 models rely heavily on pre-loaded metadata. Without schema prefilling, their performance drops sharply, evidenced by a 17.0% absolute drop for Qwen3-4B on BIRD. This suggests that standard models do not exhibit autonomous exploration capabilities. When equipped with our framework, TRUST-SQL closes this gap, with substantial gains over the base models. For instance, TRUST-SQL-4B yields a **35.6% absolute improvement** over Qwen3-4B on BIRD. Across all five benchmarks, the framework delivers an average absolute improvement of **30.6%** for the 4B variant and **16.6%** for the 8B variant compared to their respective base models under the Unknown Schema setting.

Furthermore, TRUST-SQL is largely insensitive to pre-loaded schemas. For both 4B and 8B variants, injecting the full schema upfront provides only negligible changes on BIRD and Spider. In some cases it slightly degrades performance on robustness benchmarks. Specifically, TRUST-SQL-4B drops by 2.4% on Spider-DK and TRUST-SQL-8B drops by 1.6% on Spider-Realistic. The iterative policy already retrieves the metadata necessary for SQL generation, making full schema injection largely redundant. Therefore, **active exploration is a viable alternative** to static prefilling.

5 Analysis

5.1 What Makes a Good Schema Reward?

We investigate two key design dimensions for f_{match} defined in Section 3.3: whether R_{schema} should be *coupled* with R_{exec}, and whether f_{match} should be *sparse* or *dense*. Specifically, **Sparse + Uncoupled** assigns R_{schema} regardless of R_{exec} with a binary f_{match}. **Sparse + Coupled** (**TRUST-SQL**) conditions R_{schema} on $R_{\text{exec}} = 1.0$ with the same binary criterion. **Dense + Coupled** further replaces f_{match} with a graduated function enforcing full recall as a hard gate before assigning partial precision-based rewards. Formal definitions of all three variants are given in Appendix B.2.

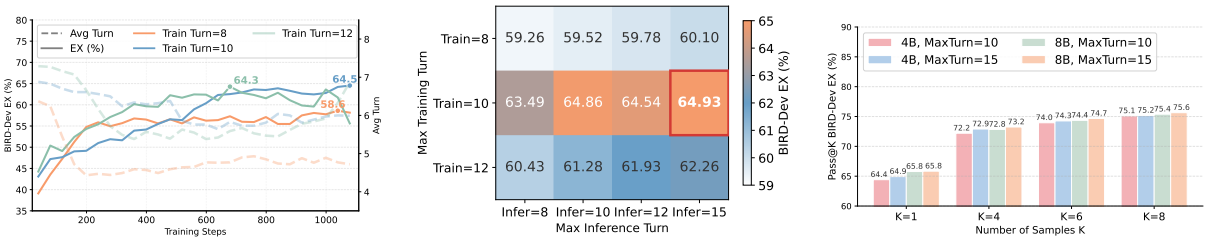
Figure 4: Ablation on schema reward formulation.

As shown in Figure 4, the three variants exhibit markedly different training dynamics. **Sparse + Uncoupled** achieves the lowest EX of 52.7% despite the highest turn count of 6.71, revealing that decoupling schema reward from execution incentivizes redundant exploration rather than precise grounding. **Dense + Coupled** reduces turns to 5.03 but converges to a suboptimal 64.0%, as the graduated f_{match} introduces conflicting gradients between maximizing recall and minimizing unnecessary columns. **Sparse + Coupled** achieves the best EX of 64.5% with a balanced turn count of 5.64, where the binary f_{match} provides an unambiguous optimization target and conditioning on $R_{\text{exec}} = 1.0$ establishes a direct causal chain between exploration quality and task success. These results indicate that *coupling* is a more critical design dimension than *reward density*.

5.2 Test-Time Scaling Behavior

We analyze the test-time scaling properties of TRUST-SQL across three dimensions.

Training Turn Budget. As illustrated in Figure 5a, expanding the training turn budget from 8 to 10 yields substantial gains on BIRD-Dev. However, further increasing to 12 turns causes severe training instability where the average turn count spikes and execution accuracy sharply declines, suggesting that an overly permissive horizon fails to penalize redundant exploration. Consequently, a 10-turn budget provides the optimal balance between accuracy and exploration efficiency.



(a) Training turn budget effect. (b) Train vs. inference turn budget. (c) Pass@K scaling.

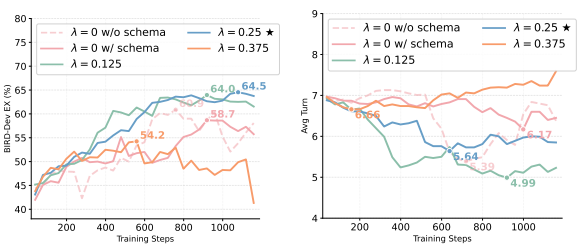
Figure 5: Test-time scaling analysis of TRUST-SQL across three dimensions: training turn budget, training vs. inference turn budget interaction, and Pass@K with repeated sampling.

Interaction Between Horizons. As shown in Figure 5b, a 10-turn training budget consistently yields the strongest baseline policy. Providing additional inference turns beyond the training horizon further improves performance, with the optimal configuration pairing a 10-turn training budget with a 15-turn inference budget to achieve a peak accuracy of **64.93%**. This demonstrates that the agent effectively utilizes extra test-time compute to recover from early exploration mistakes.

Scaling with Repeated Sampling. As shown in Figure 5c, all configurations exhibit monotonic accuracy improvements as the sample size K grows, driven by *exploration diversity* across independently sampled trajectories. The persistent gap between Pass@K and greedy performance indicates that the model can generate correct solutions but has not fully converged to a consistent policy, suggesting headroom for further RL training.

5.3 How to Balance Exploration and Generation?

In the Phase-Aware GRPO loss, λ controls the relative contribution of the Schema Track. We ablate $\lambda \in \{0.125, 0.25, 0.375\}$ against two single-track baselines where $\lambda = 0$. The first optimizes solely on execution outcome, and the second naively aggregates a schema reward weighted at 0.25 into the terminal reward without track separation.



(a) EX (%) during training. (b) Avg turns during training.

Figure 6: Effect of λ on training dynamics.

As shown in Figure 6a, naively mixing the schema reward into the terminal step yields 58.7%, worse than the 60.9% achieved by the pure execution baseline. This confirms that conflating exploration and generation obscures the reward signal. In contrast, the optimal Phase-Aware setting at $\lambda = 0.25$ peaks at **64.5%**, yielding a **+5.8%** gain over naive aggregation and a **+3.6%** gain over the pure execution baseline. Furthermore, λ dictates the balance between exploration and generation. While $\lambda = 0.125$ achieves a competitive 64.0%, an excessively large $\lambda = 0.375$ severely degrades performance to 54.2%. As shown in Figure 6b, this over-weighted schema reward incentivizes the agent to remain perpetually in the exploration phase, causing a sharp increase in average interaction turns and over-optimizing metadata retrieval at the expense of SQL generation.

6 Conclusion

In this work, we revisit the **Full Schema Assumption** that underlies Text-to-SQL research. By formalizing the task as a POMDP under the **Unknown Schema** setting, **TRUST-SQL** demonstrates that autonomous database exploration is both feasible and effective under partial schema observability, where the full database structure cannot be assumed to be available. The structured **four-phase protocol** grounds agent reasoning in actively verified metadata to suppress hallucinations, while its mandatory cognitive checkpoint provides a structural boundary for **Phase-Aware GRPO** to resolve the credit assignment bottleneck, yielding a **9.9%** relative improvement over standard GRPO. Experiments across five benchmarks at three scales confirm the framework's effectiveness. Despite operating without pre-loaded metadata, TRUST-SQL matches or surpasses schema-prefilled baselines on most benchmarks, advancing reliable Text-to-SQL in unobservable environments.

Limitations

While TRUST-SQL demonstrates strong performance under the Unknown Schema setting, several limitations remain.

Inference Overhead. The multi-turn interaction paradigm naturally incurs higher inference cost compared to single-turn methods, as each interaction step involves a live database call. However, as shown in Appendix D.1, this overhead remains modest in practice. Further optimizing inference efficiency for latency-critical deployments remains a practical direction for future work.

SQLite Dialect Only. Both training and evaluation are conducted on SQLite-based benchmarks, as BIRD and Spider exclusively use SQLite. Extending to other SQL dialects such as PostgreSQL or MySQL remains a valuable direction for future work.

Fixed Turn Budget. The maximum interaction turn T is fixed at training time, which may limit exploration thoroughness for databases with exceptionally complex schemas. Adapting the turn budget dynamically based on database complexity remains an interesting direction for future work.

Reproducibility Statement

To ensure full reproducibility, we release the complete source code at `https://anonymous.4open.science/r/TrustSQL-0902`. All dataset construction pipelines are detailed in Appendix A, training hyperparameters and hardware specifications are summarized in Appendix C. All experiments are conducted on NVIDIA A100 GPUs. Upon acceptance, we will publicly release the training datasets and model weights to further support the research community.

References

Google DeepMind. 2025. Gemini 2.5 pro. `https://deepmind.google/technologies/gemini/pro`.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

DeepSeek-AI. 2025. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2021. 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*, page 1337–1350. Association for Computational Linguistics.

Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R. Woodward, Jinxia Xie, and Pengsheng Huang. 2021a. Towards robustness of text-to-SQL models against synonym substitution. pages 2505–2515, Online. Association for Computational Linguistics.

Yujian Gan, Xinyun Chen, and Matthew Purver. 2021b. Exploring underexplored limitations of cross-domain text-to-sql generalization. *Preprint*, arXiv:2109.05157.

Taicheng Guo, Hai Wang, ChaoChun Liu, Mohsen Golalikhani, Xin Chen, Xiangliang Zhang, and Chandan K. Reddy. 2025. Mtsql-r1: Towards long-horizon multi-turn text-to-sql via agentic training. *Preprint*, arXiv:2510.12831.

Mingqian He, Yongliang Shen, Wenqi Zhang, Qiuying Peng, Jun Wang, and Weiming Lu. 2025. Star-sql: Self-taught reasoner for text-to-sql. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 24365–24375.

Harper Hua, Zhen Han, Zhengyuan Shen, Jeremy Lee, Patrick Guan, Qi Zhu, Sullam Jeoung, Yueyan Chen, Yunfei Bai, Shuai Wang, and 1 others. 2026. Sql-trail: Multi-turn reinforcement learning with interleaved feedback for text-to-sql. *arXiv preprint arXiv:2601.17699*.

Yuxiang Ji, Ziyu Ma, Yong Wang, Guanhua Chen, Xiangxiang Chu, and Liaoni Wu. 2025. Tree search for llm agent reinforcement learning. *arXiv preprint arXiv:2509.21240*.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, and 1 others. 2024. Training language models to self-correct via reinforcement learning, 2024. *URL https://arxiv.org/abs/2409.12917*, 2(3):4.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. 2025. Llms get lost in multi-turn conversation. *arXiv preprint arXiv:2505.06120*.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, and 1 others. 2024. Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows. *arXiv preprint arXiv:2411.07763*.

Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, and 1 others. 2025. Omnisql: Synthesizing high-quality text-to-sql data at scale. *arXiv preprint arXiv:2503.02240*.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, and 1 others. 2024. 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.

Xiaoqian Liu, Ke Wang, Yongbin Li, Yuchuan Wu, Wentao Ma, Aobo Kong, Fei Huang, Jianbin Jiao, and Junge Zhang. 2025. Epo: Explicit policy optimization for strategic reasoning in llms via reinforcement learning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15371–15396.

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

Meituan LongCat Team. 2025. Longcat-flash technical report. *Preprint*, arXiv:2509.01322.

OpenAI. 2024. Gpt-4o system card. `https://openai.com/index/gpt-4o-system-card`.

OpenAI. 2025. Gpt-4.1. `https://openai.com/index/gpt-4-1`.

Mohammadreza Pourreza, Shayan Talaei, Ruoxi Sun, Xingchen Wan, Hailong Li, Azalia Mirhoseini, Amin Saberi, Sercan Arik, and 1 others. 2025. Reasoning-sql: Reinforcement learning with sql tailored partial rewards for reasoning-enhanced text-to-sql. *arXiv preprint arXiv:2503.23157*.

Yang Qin, Chao Chen, Zhihang Fu, Ze Chen, Dezhong Peng, Peng Hu, and Jieping Ye. 2024. Route: Robust multitask tuning and collaboration for text-to-sql. *arXiv preprint arXiv:2412.10138*.

Qwen. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

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

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

Yanming Wan, Jiaxing Wu, Marwa Abdulhai, Lior Shani, and Natasha Jaques. 2025. Enhancing personalized multi-turn dialogue with curiosity reward. *arXiv preprint arXiv:2504.03206*.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, LinZheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2025a. MAC-SQL: A multi-agent collaborative framework for text-to-SQL. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 540–557, Abu Dhabi, UAE. Association for Computational Linguistics.

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

Zhongyuan Wang, Richong Zhang, Zhijie Nie, and Jaein Kim. 2024. Tool-assisted agent on sql inspection and refinement in real-world scenarios. *Preprint*, arXiv:2408.16991.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, Yiping Lu, Kyunghyun Cho, Jiajun Wu, Li Fei-Fei, Lijuan Wang, Yejin Choi, and Manling Li. 2025c. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *Preprint*, arXiv:2504.20073.

Ziyang Wang, Yuanlei Zheng, Zhenbiao Cao, Xiaojin Zhang, Zhongyu Wei, Pei Fu, Zhenbo Luo, Wei Chen, and Xiang Bai. 2025d. Autolink: Autonomous schema exploration and expansion for scalable schema linking in text-to-sql at scale. *Preprint*, arXiv:2511.17190.

Zekun Xu, Siyu Xia, Chuhuai Yue, Jiajun Chai, Mingxue Tian, Xiaohan Wang, Wei Lin, Haoxuan Li, and Guojun Yin. 2025. Mtir-sql: Multi-turn tool-integrated reasoning reinforcement learning for text-to-sql. *arXiv preprint arXiv:2510.25510*.

Zhenghai Xue, Longtao Zheng, Qian Liu, Yingru Li, Xiaosen Zheng, Zejun Ma, and Bo An. 2025. Simpletir: End-to-end reinforcement learning for multi-turn tool-integrated reasoning. *arXiv preprint arXiv:2509.02479*.

Cehua Yang, Dongyu Xiao, Junming Lin, Yuyang Song, Hanxu Yan, Shawn Guo, Wei Zhang, Jian Yang, Mingjie Tang, and Bryan Dai. 2025. Agro-sql: Agentic group-relative optimization with high-fidelity data synthesis. *Preprint*, arXiv:2512.23366.

Haojin Yang, Ai Jian, Xinyue Huang, Yiwei Wang, Weipeng Zhang, Ke Zeng, Xunliang Cai, and Jingqing Ruan. 2026. Harmonizing dense and sparse signals in multi-turn rl: Dual-horizon credit assignment for industrial sales agents. *Preprint*, arXiv:2603.01481.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. *Preprint*, arXiv:2210.03629.

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

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. 2018. 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*, Brussels, Belgium. Association for Computational Linguistics.

Tianshu Zhang, Kun Qian, Siddhartha Sahai, Yuan Tian, Shaddy Garg, Huan Sun, and Yunyao Li. 2026. Evoschema: Towards text-to-sql robustness against schema evolution. *Preprint*, arXiv:2603.10697.

Yuxin Zhang, Meihao Fan, Ju Fan, Mingyang Yi, Yuyu Luo, Jian Tan, and Guoliang Li. 2025. Reward-sql: Boosting text-to-sql via stepwise reasoning and process-supervised rewards. *arXiv preprint arXiv:2505.04671*.

Yifei Zhou, Song Jiang, Yuandong Tian, Jason Weston, Sergey Levine, Sainbayar Sukhbaatar, and Xian Li. 2025. Sweet-rl: Training multi-turn llm agents on collaborative reasoning tasks. *arXiv preprint arXiv:2503.15478*.

Zilin Zhu, Chengxing Xie, Xin Lv, and slime Contributors. 2025. slime: An llm post-training framework for rl scaling. https://github.com/THUDM/slime. GitHub repository. Corresponding author: Xin Lv.

A Data Construction

A.1 Baseline Training Data Comparison

To contextualize the data efficiency of TRUST-SQL, we summarize the training data configurations of all evaluated baselines in Table 3. While recent single-turn models rely on massive synthetic datasets containing millions of samples, and multi-turn RL frameworks utilize large portions of standard benchmarks, TRUST-SQL achieves superior performance using a highly constrained and curated data recipe.

Table 3: Comparison of training data volume and sources.

Model	SFT Data	RL Data
OmniSQL	2.5M (SynSQL)	–
SQL-R1	2.5M (SynSQL)	5k (SynSQL)
MTIR-SQL	–	18.1k (Spider+BIRD)
SQL-Trail	0.8k (SynSQL)	1k (Spider)
TRUST-SQL	**9.2k (SynSQL)**	**11.6k (Spider+BIRD)**

A.2 SFT Training Data Construction

To warm up the agent prior to RL training, we construct a supervised fine-tuning dataset of high-quality exploration trajectories. The source questions are sampled from the training split of **SynSQL-2.5M** (Li et al., 2025). We filter this corpus to retain only questions of *Moderate*, *Complex*, and *Highly Complex* difficulty, as simpler questions provide insufficient training signal for multi-turn exploration. This yields 9,217 unique source questions, whose difficulty distribution is shown in Table 4.

Table 4: Difficulty distribution of source questions.

Difficulty	Count	Proportion
Moderate	3,821	41.5%
Complex	3,243	35.2%
Highly Complex	2,153	23.3%
Total	**9,217**	**100%**

Annotation Pipeline. We employ a multi-model annotation strategy using **GPT-4.1-mini** (OpenAI, 2025), **GPT-4o-mini** (OpenAI, 2024), and **DeepSeek-R1** (DeepSeek-AI, 2025). Each model is prompted to generate complete four-phase interaction trajectories following the TRUST-SQL protocol with a maximum output length of 2,048 tokens per response. A trajectory is retained if and only if it satisfies two strict conditions. First, the final SQL execution result must match the ground truth answer. Second, every turn must pass the format check described in Appendix F. This execution-verified filtering ensures that the SFT model learns from trajectories that are both correct and structurally well-formed.

Dataset Statistics. Table 5 summarizes the retained samples contributed by each annotation model.

Table 5: SFT training data statistics by annotation model. *Samples* denotes the total number of retained trajectories and *Unique IDs* denotes the number of distinct source questions covered by each model.

Annotation Model	Samples	Unique IDs
DeepSeek-R1	9,972	1,442
GPT-4.1-mini	43,803	5,575
GPT-4o-mini	17,195	2,263
Total	**70,970**	**9,280**

The majority of retained samples are contributed by GPT-4.1-mini (43,803 samples, 61.7%), reflecting its stronger instruction-following capability in generating well-formatted trajectories. DeepSeek-R1 contributes 9,972 samples across 1,442 unique questions, providing diverse chain-of-thought reasoning styles that complement the GPT-annotated data.

A.3 RL Training Data Construction

Question Selection. For RL training, we adopt the source questions from the training sets of **BIRD** (Li et al., 2024) and **Spider** (Yu et al., 2018) with identical difficulty filtering. To ensure effective RL exploration, we apply a difficulty-based filtering strategy where each question is rolled out 8 times using the SFT-initialized policy. Only questions with a pass rate strictly below 6/8 are retained. This criterion excludes questions that are already too easy for the current policy, as they provide negligible learning signal.

Ground Truth Schema Extraction. To compute the schema reward R_{schema} during RL training, we require the ground truth schema $\mathcal{K}^* = (\mathcal{K}^*_{\text{table}}, \mathcal{K}^*_{\text{col}})$ for each training instance. We extract $\mathcal{K}^*$ directly from the ground truth SQL y^* via rule-based parsing using standard SQLite utilities, which yields the referenced table names and column names without additional cost or noise. This

Table 6: RL training data filtering statistics.

Statistic	Value
Total Candidate Questions	18,078
Retained Questions	11,642
Rejected Questions	6,436
Keep Rate	64.4%
Pass Rate Threshold	$< 6/8$

procedure introduces no extra human annotation and scales straightforwardly to new domains.

B Phase-Aware GRPO: Detailed Formalization

This section consolidates the formal details of Phase-Aware GRPO summarized in Section 3.4. We clarify (i) the rollout-dependent definition of the *Propose* checkpoint t_{propose}; (ii) the schema matching function f_{match} and its three variants used in the ablation study; (iii) an explicit algorithmic contrast against standard GRPO; and (iv) the two-level role—workflow-level and optimization-level—played by the *Propose* stage.

B.1 Rollout-Dependent Definition of the *Propose* Step

The checkpoint step t_{propose} is not a fixed hyperparameter. For each rollout τ^i, we define it dynamically as the step index of the **latest** `propose_schema` action in that trajectory:

$$t^i_{\text{propose}} = \max\{\, t : a^i_t = \texttt{propose_schema} \,\}. \quad (7)$$

If the agent revises its schema multiple times during exploration, only the schema committed at this final checkpoint is used to compute the schema reward. Let $\hat{\mathcal{K}}^i$ denote the schema proposed at t^i_{propose}; the schema reward then evaluates

$$R^i_{\text{schema}} = f_{\text{match}}(\hat{\mathcal{K}}^i, \mathcal{K}^*), \quad (8)$$

and the Schema Track in Section 3.4 is the trajectory prefix up to this latest checkpoint. The schema-track advantage is broadcast only to tokens within this prefix, while the global rollout reward continues to supervise the overall protocol behavior.

B.2 Schema Matching Function and Its Variants

Let P_t, P_c denote the proposed table and column sets in $\hat{\mathcal{K}}$, and G_t, G_c the corresponding ground-truth sets in $\mathcal{K}^*$. We consider three matching variants in Section 5.1; the **sparse execution-coupled** form is the default used by TRUST-SQL throughout the main experiments.

(i) Sparse + Uncoupled. Exact-match reward on both tables and columns:

$$f_{\text{match}} = \mathbf{1}[P_t = G_t \wedge P_c = G_c]\,. \quad (9)$$

(ii) Sparse + Coupled (default). The same exact-match criterion, additionally gated by execution correctness so that schema credit is given only when the resulting SQL also executes correctly:

$$R_{\text{schema}} = \mathbf{1}[R_{\text{exec}} = 1] \cdot f_{\text{match}}. \quad (10)$$

(iii) Dense + Coupled. First enforce full recall on both tables and columns ($r_t = r_c = 1$); then compute a discretized reward based on a weighted combined precision

$$\pi = w_t\, p_t + w_c\, p_c, \quad (11)$$

with reward levels $\{1.0, 0.8, 0.6, 0.4, 0.2, 0.1\}$ at thresholds $\{1.0, 0.9, 0.8, 0.7, 0.6\}$, again gated by execution success. Sparse variants reward only exact schema identification, whereas the dense coupled variant strongly penalizes missing schema elements while still assigning partial credit to over-selected columns, provided that the final SQL executes correctly.

B.3 Algorithmic Contrast with Standard GRPO

Phase-Aware GRPO differs structurally from standard GRPO along three dimensions, isolated by the three controlled variants used in Section 5.3.

(i) Standard GRPO ($\lambda = 0$, no schema reward). A single trajectory-level reward

$$R = R_{\text{exec}} + R_{\text{fmt}} \quad (12)$$

yields one normalized advantage $A = (R - \mu)/(\sigma + \epsilon)$, which is broadcast uniformly to every token in the rollout.

(ii) Naive single-track with schema reward ($\lambda = 0$, w/ schema). The schema reward is added directly into the same terminal scalar,

$$R = R_{\text{exec}} + R_{\text{fmt}} + 0.25\, R_{\text{schema}}, \quad (13)$$

while still using a single shared advantage broadcast across the entire rollout. This conflates the two learning signals.

(iii) Phase-Aware GRPO (ours). The rollout is partitioned at t_{propose} into a Schema Track and a Full Track. Each track is normalized into its own advantage independently, and the schema-track advantage is applied *only* to tokens satisfying $t \leq t_{\text{propose}}$.

Summary. Standard GRPO uses *one reward, one advantage, one global mask*, whereas Phase-Aware GRPO uses *two rewards, two independently normalized advantages, and two disjoint masks defined by the explicit schema checkpoint*. The masked-advantage formulation is strictly finer-grained than trajectory-level reward shaping, since it prevents exploration credit from leaking into generation tokens and vice versa.

B.4 Two-Level Role of the Propose Checkpoint

The *Propose* stage contributes at two distinct levels, which we separate for clarity.

Workflow-level effect. At inference, the *Propose* stage forces the agent to explicitly commit to verified metadata before generation. This commitment reduces the chance of fabricating non-existent schema elements during SQL generation. To isolate this effect from any RL-induced gain, we conduct a controlled SFT-only experiment on Qwen3-4B in which the entire *Propose* phase is removed from the training trajectories and the data is correspondingly rewritten to maintain a consistent workflow structure (Table 7). The explicit *Propose* stage already yields a moderate gain (46.2 vs. 45.1) before any RL is applied, indicating that the checkpoint itself improves schema grounding independently of Phase-Aware GRPO.

Setting	Workflow	EX (%)
w/o *Propose*	E → G → C	45.1
w/ *Propose*	E → P → G → C	**46.2**

Table 7: SFT-only ablation isolating the workflow-level effect of the explicit *Propose* stage on Qwen3-4B.

Optimization-level effect. Structurally, the *Propose* stage provides a natural boundary for schema-specific credit assignment during RL. Without this explicit checkpoint, schema and SQL-generation tokens would have to share a single global advantage, as in the naive single-track variant of Appendix B.3. The checkpoint enables Phase-Aware GRPO to localize schema rewards to the exploration/proposal prefix instead of the entire rollout, which is the source of the additional improvement reported in Section 5.3 on top of the workflow-level gain.

C Implementation Details

We train three model scales, namely Qwen3-4B, Qwen3-8B, and Qwen3-14B, each passing through a supervised fine-tuning warm-up stage followed by Phase-Aware GRPO optimization. RL training adopts synchronous rollouts under the SLIME framework (Zhu et al., 2025) for the 4B variant, while the 8B and 14B variants adopt asynchronous training to accommodate the larger compute footprint. Table 8 summarizes the hardware configuration, key hyperparameters, and estimated training cost for each stage.

D Cost and Efficiency Analysis

D.1 Cost Analysis

Table 9 presents a comprehensive inference cost analysis on BIRD-Dev, comparing accuracy, latency, token consumption, interaction turns, and tool call frequency across all methods.

Accuracy vs. Cost Trade-off. Training-free pipeline methods such as CHESS achieve competitive accuracy at a high cost of 251.3 seconds and 320.8K tokens per query, which makes them costly to deploy at scale. In contrast, TRUST-SQL-4B achieves a higher accuracy of 64.9% under the Unknown Schema setting with only 0.6 seconds latency and 2.83K tokens, representing a **500× reduction in latency** and a **113× reduction in token consumption** compared to CHESS.

Efficiency of Active Exploration. Compared to schema-prefilled baselines of similar scale, TRUST-SQL maintains competitive inference efficiency. TRUST-SQL-4B without prefilling consumes only 2.83K tokens and completes interactions in 5.89 average turns, comparable to MTIR-SQL-4B which consumes 2.9K tokens under full schema access. This suggests that the active exploration policy retrieves the metadata it needs without substantial overhead.

Impact of Schema Prefilling on Base Models. We observe an asymmetric effect of schema prefilling on base models versus TRUST-SQL. For Qwen3-4B, removing prefilling increases token consumption from 1.82K to 4.93K and degrades accuracy from 46.3% to 29.3%, indicating strong dependence on pre-loaded metadata. In contrast, TRUST-SQL-4B without prefilling consumes only

Table 8: Training setup for TRUST-SQL across model scales and training stages.

Model	Stage	GPUs	Mode	LR	Batch Size	Epochs	Rollout	λ	Turns	Time (hrs)
Qwen3-4B	SFT	16×A100	sync	1×10^{-5}	256	2	–	–	–	6.5
	RL	8×A100	sync	1×10^{-6}	32	3	8	0.25	10	60
Qwen3-8B	SFT	16×A100	sync	1.5×10^{-6}	256	2	–	–	–	12
	RL	32×A100	async	8×10^{-7}	32	3	8	0.25	10	40
Qwen3-14B	SFT	8×A100	sync	7.5×10^{-7}	256	2	–	–	–	24
	RL	48×H100	async	2×10^{-6}	32	2	8	0.25	10	40

2.83K tokens while maintaining 64.9% accuracy, suggesting that Phase-Aware GRPO yields efficient and targeted exploration behavior.

D.2 Cost-Accuracy Positioning

To complement the accuracy-only comparison in Appendix E.5, we estimate the serving cost of each model under the same Unknown Schema setting using publicly available pricing: Claude Opus 4.6 at \$15/M input + \$75/M output, Qwen3-14B hosting at approximately \$0.30/M input + \$0.50/M output, and Qwen3-8B hosting at approximately \$0.15/M input + \$0.25/M output. We assume a 70/30 input-output token ratio and an amortized cost of 2K tokens per interaction turn. The resulting cost estimates are shown in Table 10.

Two observations follow. First, TRUST-SQL-14B achieves slightly higher EX than Claude Opus 4.6 under the same 4S-US setting (68.10 vs. 67.34), while reducing estimated serving cost by roughly two orders of magnitude (\$3.4 vs. \$339 per 1K queries). Second, although ReAct-US prompting with Claude Opus 4.6 is markedly cheaper than 4S-US prompting with the same model, it still costs more than 25× TRUST-SQL-14B and lags by roughly 3 EX points. These results indicate that simply scaling model size or prompting style does not remove the value of structured schema exploration combined with specialized training for the partially observable setting.

Table 9: Inference cost analysis on BIRD-Dev.

Method	Prefill	Acc (%)	Latency (s)	OutputTokens (K)	Turns	Tool Calls
CHESS (Talaei et al., 2024)	✓	61.5	251.3	320.8	–	–
SQL-R1-7B (Ma et al., 2025)	✓	63.7	0.4	3.1	–	–
MTIR-SQL-4B (Xu et al., 2025)	✓	63.1	0.5	2.9	–	1.34
MTIR-SQL-8B (Xu et al., 2025)	✓	63.6	0.4	2.0	–	1.31
Qwen3-4B (Qwen, 2025)	✓	46.3	0.4	1.82	2.34	1.64
Qwen3-4B (Qwen, 2025)	✗	29.3	1.2	4.93	7.66	4.42
Qwen3-8B (Qwen, 2025)	✓	49.9	0.4	2.15	2.14	2.92
Qwen3-8B (Qwen, 2025)	✗	47.9	1.0	3.85	6.34	4.41
TRUST-SQL-4B	✓	**64.8**	**0.4**	**1.75**	**4.23**	**2.89**
TRUST-SQL-4B	✗	**64.9**	**0.6**	**2.83**	**5.89**	**3.66**
TRUST-SQL-8B	✓	**65.5**	**0.5**	**2.00**	**4.69**	**3.61**
TRUST-SQL-8B	✗	**65.8**	**0.5**	**2.03**	**5.62**	**3.45**

Table 10: Estimated serving cost under the Unknown Schema setting on BIRD-Dev. TRUST-SQL rows are highlighted. TRUST-SQL-14B reaches slightly higher EX than Claude Opus 4.6 under the same 4S-US setting at roughly two orders of magnitude lower serving cost.

Model	Setting	EX	Avg Turns	Cost / query	Cost / 1K queries
Claude Opus 4.6	4S-US	67.34	5.14	$0.339	$339.2
Claude Opus 4.6	ReAct-US	65.19	1.32	$0.087	$87.1
TRUST-SQL-14B	4S-US	**68.10**	4.67	**$0.0034**	**$3.4**
TRUST-SQL-8B	4S-US	65.80	5.62	$0.0020	$2.0

E Additional Empirical Results

E.1 Is Cold-Start SFT Necessary?

TRUST-SQL adopts a two-stage training pipeline in which Phase-Aware GRPO is preceded by an SFT warm-up phase. To assess its necessity, we compare three training configurations on TRUST-SQL-4B (Table 11). Applying Phase-Aware GRPO directly without SFT warm-up yields 59.9% on BIRD and 79.6% on Spider, both substantially below the full pipeline. These numbers are largely illusory: without SFT initialization, the model quickly learns to **hack the reward by exhaustively querying all tables and columns in the first turn**, completing the interaction in roughly four actions. This degenerates the Unknown Schema setting into a disguised Full Schema scenario, bypassing genuine active exploration. SFT alone achieves reasonable performance, confirming that the warm-up phase instills structured exploration behavior, while the full two-stage pipeline consistently achieves the best results. This indicates that Phase-Aware GRPO provides gains that cannot be attributed to supervised learning alone.

Table 11: Ablation on cold-start SFT for TRUST-SQL-4B (greedy decoding).

Configuration	SFT	RL	BIRD (dev)	Spider (test)
SFT Only	✓	✗	46.2	66.7
RL Only	✗	✓	59.9	79.6
SFT + RL	✓	✓	**64.9**	**82.8**

E.2 Pass@K Results on Additional Benchmarks

Section 5.2 reports Pass@K scaling behavior on BIRD-Dev. Here we extend this analysis to the remaining four benchmarks to verify that the monotonic scaling trend generalizes across different evaluation settings. Table 12 reports Pass@K results for $K \in \{1, 4, 6, 8\}$ under a 15-turn inference budget.Consistent with the BIRD-Dev results reported in the main paper, all benchmarks exhibit monotonic accuracy improvements as K grows. The persistent gap between Pass@K and greedy performance indicates that the model possesses the capability to generate correct solutions but has not fully converged to a consistent policy, suggesting headroom for further training.

Table 12: Pass@K results across all benchmarks (temperature = 0.8, max turns = 15).

Size	Benchmark	Pass@1	Pass@4	Pass@6	Pass@8
4B	Spider (test)	82.8	86.5	86.9	87.1
	Spider-DK	71.6	78.8	80.3	81.2
	Spider-Syn	74.7	81.6	82.3	83.1
	Spider-Realistic	79.9	85.4	86.2	86.6
8B	Spider (test)	83.9	86.5	87.1	87.5
	Spider-DK	72.1	79.2	80.5	81.3
	Spider-Syn	75.4	81.0	83.0	84.0
	Spider-Realistic	82.1	85.5	86.4	87.0

E.3 Performance on Complex Benchmark (Spider 2.0)

To evaluate TRUST-SQL under more challenging real-world conditions, we conduct additional experiments on the SQLite subset of Spider 2.0 (Lei et al., 2024), comprising 135 questions over large-scale databases featuring significantly more complex schemas and larger table counts than standard Spider. This setting is particularly well-suited for assessing the Unknown Schema framework, as the increased schema complexity makes full schema prefilling even more impractical.

Table 13 reports execution accuracy alongside representative baselines. Notably, strong proprietary models such as GPT-4o (OpenAI, 2024) and DeepSeek-V3 (DeepSeek-AI, 2025) achieve only 15.6% on this benchmark, while specialized Text-to-SQL models like OmniSQL-7B (Li et al., 2025)

Table 14: Trajectory-level statistics across training checkpoints of TRUST-SQL-8B on BIRD-Dev. As accuracy improves, exploration shifts earlier in the trajectory rather than becoming longer.

Metric	**Pre**	**Post**	
	(iter 39)	(iter 799)	(iter 1119)
Total Accuracy (%)	43.1	63.6	**64.5**
Avg Rounds	7.23	5.80	6.16
Avg Explore Turns	4.01	2.36	2.50
Avg Propose Turns	0.97	1.16	1.38
Avg Propose First Turn	3.84	2.15	2.24
Avg Propose Last Turn	4.05	2.58	3.07
Avg Propose Tokens	438	682	754

reach 10.4%, reflecting the substantial difficulty of this setting.

Table 13: Execution accuracy on the Spider 2.0 SQLite subset (135 questions). Baselines use full schema prefilling. Pass@8 is computed over 8 sampled trajectories.

Method	**Prefill**	**Greedy**	**Pass@8**
OmniSQL-7B	✓	10.4	–
GPT-4o	✓	15.6	–
DeepSeek-V3	✓	15.6	–
OpenSearchSQL+Qwen2.5-7B-Instruct	✓	4.4	7.4
OpenSearchSQL+Arctic-Text2SQL-R1-7B	✓	14.1	20.7
TRUST-SQL-8B	✗	**14.8**	**24.9**

Despite operating entirely without pre-loaded metadata, TRUST-SQL-8B achieves 14.8% greedy accuracy and 24.9% Pass@8, surpassing OpenSearchSQL paired with the specialized Arctic-7B (Yao et al., 2025) model. The non-saturating Pass@8 curve further suggests substantial headroom for improvement with increased sampling budgets, validating the generalizability of our framework beyond standard academic benchmarks.

E.4 Training Dynamics

To examine whether stronger performance is driven by longer exploration or by a more efficient interaction profile, we track a set of trajectory-level statistics across three checkpoints of the RL training of TRUST-SQL-8B on BIRD-Dev. Table 14 reports execution accuracy along with the average number of interaction rounds, the average number of *Explore* and *Propose* actions per trajectory, the first and last step at which a *Propose* action is issued, and the average number of tokens spent inside each *Propose* action.

Two trends emerge as training progresses. First, accuracy increases from 43.1% to 64.5%, while the average number of interaction rounds and *Explore* actions decreases substantially. Second, schema proposals shift noticeably earlier in the trajectory: the first *Propose* turn moves from step 3.84 down to about step 2.2, and the per-*Propose* token budget grows from 438 to 754. Taken together, these results indicate that stronger performance comes from a more decisive exploration–proposal pattern, not from extending the exploration phase. The model gradually learns where to place the schema-consolidation boundary, allowing it to commit to relevant metadata earlier in the reasoning process.

E.5 Comparison with Large Agentic Models

To assess whether stronger proprietary backbones can substitute for the proposed training framework, we additionally evaluate several large agentic foundation models on BIRD-Dev under three settings: (i) **4S-US**, the four-stage protocol under the Unknown Schema setting; (ii) **ReAct-US**, ReAct-style prompting (Yao et al., 2023) under the Unknown Schema setting; and (iii) **4S-SP**, the four-stage protocol with schema prefill. For each setting we report execution accuracy, the average number of interaction rounds, and the Success@15 rate, defined as the percentage of samples that terminate with a valid final answer within the 15-turn interaction budget (Table 15).

Three findings emerge. First, the Unknown Schema setting is substantially harder than schema-prefilled prompting, and the gap widens for weaker backbones: removing the prefill reduces EX consistently across all three models, with the largest drop on GLM-5.1 (63.23 → 47.07). This indicates that Unknown Schema is not merely a prompt-variation, but a fundamentally harder partially observable interaction setting.

Second, the structured four-stage protocol consistently outperforms ReAct-style prompting across every backbone we tested. Even for strong proprietary agentic models, explicit interaction structure still contributes beyond raw model capability.

Third, prompting alone does not fully substitute for specialized RL training. Under the same 4S-US setting, Claude Opus 4.6 reaches 67.34 with 5.14 average rounds, which remains below TRUST-SQL-14B (68.10 / 4.67) despite relying on a substantially stronger proprietary backbone. We further observe two characteristic failure modes

Table 15: Large agentic foundation models on BIRD-Dev under three settings. EX denotes execution accuracy (%), Turns denotes the average number of interaction rounds, and S@15 denotes Success@15. **Bold** highlights the best setting per row. For reference, TRUST-SQL-14B reaches **68.10** EX with 4.67 average rounds under the same Unknown Schema setting.

Model	4S-US			ReAct-US			4S-SP		
	EX	Turns	S@15	EX	Turns	S@15	EX	Turns	S@15
Claude Opus 4.6	67.34	5.14	97.8	65.19	1.32	99.4	**69.30**	1.19	99.9
Claude Sonnet 4.6	56.39	1.68	99.9	54.89	1.47	99.8	**59.78**	1.05	100
GLM-5.1	47.07	9.67	73.3	43.74	5.98	69.7	**63.23**	3.48	98.5

when adapting general-purpose agentic models to Unknown Schema purely through prompting: (i) *under-exploration*, typical of Claude Sonnet 4.6, which frequently terminates within 1–2 interaction rounds and effectively skips schema exploration; and (ii) *inefficient exploration*, typical of GLM-5.1, which issues many ineffective tool calls under partial observability (avg 9.67 rounds) and exhibits a much lower Success@15 rate (73.3%).

F Agent Configuration

F.1 Tool Overview

The agent interacts with the database environment through four structured tools, each corresponding to one phase of the TRUST-SQL protocol. Table 16 provides an overview of their roles and output tags.

Table 16: Overview of the four tools in the TRUST-SQL action space.

Action	Phase	Output Tag	Description
`explore_schema`	Explore	`<tool_call>`	Query database metadata
`propose_schema`	Propose	`<schema>`	Commit to verified schema
`generate_sql`	Generate	`<tool_call>`	Execute candidate SQL
`confirm_answer`	Confirm	`<answer>`	Submit final SQL answer

F.2 Format Check Rules

At each turn, the agent's output must conform to a strict structural protocol. The `FormatCheck` function validates each turn by enforcing the following rules:

1. **Think tag**: The output must contain exactly one `<think>...</think>` block.

2. **Action tag**: The output must contain exactly one `<action>...</action>` block, whose content must be one of the four valid action types.

3. **Content tag**: Each action type requires a corresponding content tag, as specified in Table 16. Specifically, `explore_schema` and `generate_sql` require a `<tool_call>` block; `propose_schema` requires a `<schema>` block; and `confirm_answer` requires an `<answer>` block.

A turn is considered valid if and only if all three conditions are satisfied, yielding a format score of 0.1. Any violation results in a format score of 0.0 and terminates the format reward for the entire trajectory.

F.3 Prompt Template

The following presents the complete prompt used in TRUST-SQL, comprising a system prompt that defines the agent's role, action protocol, and output format, followed by a user prompt that provides the task-specific context.

G Case Study

We present a case study on BIRD-Dev instance `dev_4` (database `california_schools`, Qwen3-4B, greedy decoding) to qualitatively examine how schema availability shapes model behavior. The full interaction traces are shown in Figure 7.

Task. The question requires retrieving phone numbers of directly charter-funded schools opened after January 1, 2000. Answering correctly demands grounding the funding-type predicate in the actual column values stored in the database, information absent from both the question and the external knowledge hint.

Unknown Schema Setting (6 turns). Without any prior schema knowledge, the model adopts a systematic bottom-up exploration strategy. In T1 and T2, it queries `sqlite_master` to discover available tables and retrieve their schema definitions. In T3, it probes the actual values of `Charter_Funding_Type` in the `frpm` table, uncovering the critical predicate value `Directly_funded`. Only after this value-level ver-

ification does the model commit to a schema proposal in T4 and generate the correct SQL in T5, which is subsequently confirmed in T6.

Schema Prefill Setting (4 turns). When the full schema is injected as a synthetic explore turn in T1, the model skips exploratory interactions and moves directly to schema proposal in T2 and SQL generation in T3. However, reasoning solely from structural metadata without inspecting actual column values, the model fails to discover the `Directly_funded` predicate. The generated SQL filters only on `Charter_School_(Y/N) = 1`, retrieving all charter schools regardless of funding type and producing a semantically broader answer that does not fully satisfy the question.

Discussion. The contrast reveals that schema prefilling accelerates inference but sacrifices value-level grounding. Interactive exploration enables the model to adaptively acquire the precise data-level knowledge needed for accurate SQL generation. This suggests that the benefit of the Unknown Schema setting lies not merely in schema discovery, but in fostering a more thorough and evidence-driven reasoning process.

System Prompt

 # Role
You are an expert SQL assistant working on an *unknown* database. You must **never hallucinate** tables or columns.
All schema knowledge MUST come from metadata queries only.
You must operate strictly through the **Action Protocol**.

 # Action Protocol
You must follow this sequence (can loop back if needed):

1. `explore_schema` — Query database metadata
2. `propose_schema` — Document verified schema
3. `generate_sql` — Create SQL query
4. `confirm_answer` — Output final SQL

ACTION: `explore_schema`
Used to query database metadata (tables, columns, foreign keys, etc.).

- Only metadata queries allowed.
- No user-intent SQL here.
- Verify relationships between tables when multi-table queries are needed.

ACTION: `propose_schema`
Used to output the current verified schema knowledge.

- Include ONLY tables/columns actually verified through `explore_schema`.
- Do NOT hallucinate or assume any unverified structures.
- `joins` is optional; include only when relationships are explicitly verified.
- Supports both single-table and multi-table structures.

ACTION: `generate_sql`
Used to generate the SQL answer based on the latest `<schema>`.

- Use ONLY verified schema from `propose_schema`.
- If required tables/columns are missing, switch back to `explore_schema` or `propose_schema` in the next message.
- SQL must be syntactically valid and executable.
- Consider query optimization (indexes, joins, filters).
- Validate the SQL logic matches user intent.

ACTION: `confirm_answer`
Used when you have validated the generated SQL and confirmed it meets user requirements.

- Execute this action ONLY after `generate_sql` validation.
- You MUST NOT return or describe any query results.
- You MUST NOT output anything other than SQL inside `<answer>`.
- The final output must be ONLY the SQL query in proper format.

 # Output Format

EVERY response must follow this exact structure:

<think>[Your reasoning process here]</think>

<action>[one of: explore_schema | propose_schema | generate_sql | confirm_answer]</action>

[Action-specific content below]

explore_schema

```
<think>reasoning</think>
<action>explore_schema</action>
<tool_call>
{"name": "execute_sql_query",
 "arguments": {"db_id": "...", "sql": "..."}}
</tool_call>
```

propose_schema

```
<think>reasoning</think>
<action>propose_schema</action>
<schema>
{
 "tables": ["tableA"],
 "columns": { "tableA": ["col1", "col2"] },
 "joins": []
}
</schema>
```

generate_sql

```
<think>reasoning</think>
<action>generate_sql</action>
<tool_call>
{"name": "execute_sql_query",
 "arguments": {"db_id": "...", "sql": "..."}}
</tool_call>
```

confirm_answer

```
<think>reasoning</think>
<action>confirm_answer</action>
<answer>
SELECT ... FROM ... WHERE ...
</answer>
```

 # Tools

You may call one or more functions to assist with the user query.

```
{
 "type": "function",
 "function": {
```

```
    "name": "execute_sql_query",
    "description": "Execute SQL query and return partial
    results containing column names.
    (maximum 30 records)",
    "parameters": {
     "type": "object",
     "properties": {
      "db_id": {
       "type": "string",
       "description": "The name of the database to query"
      },
      "sql": {
       "type": "string",
       "description": "The SQL query to execute"
      }
     },
     "required": ["db_id", "sql"]
    }
   }
  }
```

For each function call, return a json object with function name and arguments within `<tool_call></tool_call>` XML tags:

```
<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call>
```

User Prompt

Task Configuration

Database Engine: SQLite
Database: {db_id}
External Knowledge: {external_knowledge}
User Question: {question}?

Database: `california_schools` **Decoding:** Greedy **Max Turns:** 10
External Knowledge: Charter schools refers to `'Charter School (Y/N)' = 1` in `frpm`
Question (dev_4): *Please list the phone numbers of the direct charter-funded schools that are opened after 2000/1/1.*

Unknown Schema (Qwen3-4B, 6 turns)

T1 [explore_schema]

```
SELECT name AS table_name
FROM sqlite_master
WHERE type IN ('table');
```

```
table_name
frpm  satscores  schools
```

T2 [explore_schema]

```
SELECT sql FROM sqlite_master
WHERE type IN ('table')
  AND name IN ('frpm','schools');
```

[environment feedback omitted]

T3 [explore_schema]

```
SELECT DISTINCT "Charter_Funding_Type"
FROM frpm;
```

[environment feedback omitted]

T4 [propose_schema]

```
tables: [frpm, schools]
frpm:   [Charter School (Y/N),
         Charter Funding Type, CDSCode]
schools:[Phone, OpenDate, CDSCode]
join:   frpm.CDSCode = schools.CDSCode
```

T5 [generate_sql]

```
SELECT s.Phone FROM frpm f
JOIN schools s ON f.CDSCode=s.CDSCode
WHERE f."Charter_School_(Y/N)"=1
  AND f."Charter_Funding_Type"
      ='Directly_funded'
  AND s.OpenDate > '2000-01-01';
```

[environment feedback omitted]

T6 [confirm_answer]

```
SELECT s.Phone FROM frpm f
JOIN schools s ON f.CDSCode=s.CDSCode
WHERE f."Charter_School_(Y/N)"=1
  AND f."Charter_Funding_Type"
      ='Directly_funded'
  AND s.OpenDate > '2000-01-01';
```

Schema Prefill (Qwen3-4B, 4 turns)

T1 [explore_schema] (synthetic prefill)

```
CREATE TABLE frpm (CDSCode TEXT,
  `Charter School (Y/N)` INTEGER,
  `Charter Funding Type` TEXT, ...)
CREATE TABLE schools (CDSCode TEXT,
  Phone TEXT, OpenDate DATE, ...)
```

T2 [propose_schema]

```
tables: [schools, frpm]
schools:[CDSCode, Phone, OpenDate]
frpm:   [CDSCode,Charter School (Y/N)]
join:   schools.CDSCode=frpm.CDSCode
```

T3 [generate_sql]

```
SELECT DISTINCT s.Phone
FROM schools s
INNER JOIN frpm f
  ON s.CDSCode = f.CDSCode
WHERE f."Charter_School_(Y/N)" = 1
  AND s.OpenDate > '2000-01-01'
```

[environment feedback omitted]

T4 [confirm_answer]

```
SELECT DISTINCT s.Phone
FROM schools s
INNER JOIN frpm f
  ON s.CDSCode = f.CDSCode
WHERE f."Charter_School_(Y/N)" = 1
  AND s.OpenDate > '2000-01-01'
```

Figure 7: Case study on BIRD-Dev instance dev_4 (database `california_schools`, Qwen3-4B, greedy decoding). **Left:** Unknown Schema setting with interactive metadata exploration. **Right:** Schema Prefill variant where the complete schema is injected as a synthetic *Explore* turn.