SQL-Trail: Multi-Turn Reinforcement Learning with Interleaved Feedback for Text-to-SQL

Harper Hua[1*], Zhen Han[2], Zhengyuan Shen[2], Meng-Chieh Lee[2], Sheng Guan[2], Qi Zhu[2], Sullam Jeoung[2], Yueyan Chen [2], Yunfei Bai[2], Shuai Wang[2], Vassilis Ioannidis[2], Huzefa Rangwala[2]
[1]Stanford University, [2]Amazon Web Services

Abstract

While large language models (LLMs) have substantially improved Text-to-SQL generation, a pronounced gap remains between AI systems and human experts on challenging benchmarks such as BIRD-SQL. We argue this gap stems largely from the prevailing single-pass paradigm, which lacks the iterative reasoning, schema exploration, and error-correction behaviors that humans naturally employ. To address this limitation, we introduce **SQL-Trail**, a multi-turn reinforcement learning (RL) agentic framework for Text-to-SQL. Rather than producing a query in one shot, **SQL-Trail** interacts with the database environment and uses execution feedback to iteratively refine its predictions. Our approach centers on two key ideas: (i) an adaptive turn-budget allocation mechanism that scales the agent's interaction depth to match question difficulty, and (ii) a composite reward panel that jointly incentivizes SQL correctness and efficient exploration. Across benchmarks, **SQL-Trail** sets a new state of the art and delivers strong data efficiency—up to **18×** higher than prior single-pass RL state-of-the-art methods. Notably, our 7B and 14B models outperform substantially larger proprietary systems by **5%** on average, underscoring the effectiveness of interactive, agentic workflows for robust Text-to-SQL generation.

1 Introduction

Text-to-SQL enables intuitive access to structured databases by automatically converting natural language questions into executable SQL queries, thereby democratizing data retrieval for non-expert users (Qin et al., 2022; Liu et al., 2025b). Recent large language models (LLMs) have achieved remarkable progress in this task, advancing both research frontiers and real-world applications (OpenAI, 2023; Anthropic, 2025). Current state-of-the-art approaches for LLM adaptation primarily rely on chain-of-thought prompting (Pourreza and Rafiei, 2023; Dong et al., 2023; Li et al., 2024b), supervised fine-tuning (SFT) (Li et al., 2025a; Wang et al., 2024), or reinforcement learning (RL) (Ma et al., 2025; Pourreza et al., 2025). However, despite these sophisticated methods, a substantial performance gap remains between humans and the best AI systems on challenging leaderboards such as BIRD-SQL (Blier and Ollivier, 2021).

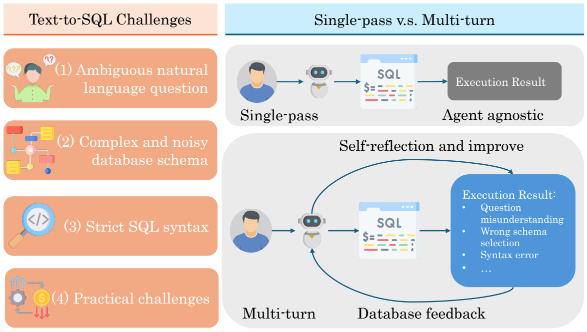

Figure 1: Illustration of core Text-to-SQL challenges and the shift from single-pass to multi-turn generation.

This gap largely stems from the single-pass paradigm underlying most existing methods: given a natural language question and a database schema, the model directly generates a SQL query without leveraging feedback from the database environment, such as execution results or error messages, into its reasoning process (Chen et al., 2025; Elgohary et al., 2020). Moreover, as shown in Figure 1, Text-to-SQL poses several intrinsic challenges: (1) natural language questions are often ambiguous ; (2) database schemas can be large, complex, and filled with noisy or semantically ambiguous entity names; and (3) SQL's strict syntactic structure leaves little room for error; (4) practical constraints regarding high computational costs and the scarcity of large-scale, high-quality public training datasets (Liu et al., 2025b). As noted by Pourreza and Rafiei (2024a), most model failures arise from

*Work done during internship at AWS. Corresponding to shuohua@stanford.edu.

incorrect schema linking, namely difficulty identifying the most relevant tables and columns, and from hard examples involving nested subqueries, multi-hop joins, and complex aggregations. In contrast, human experts excel at this task because they can iteratively interact with the database to better understand its structure, decompose complex problems into smaller subqueries to test intermediate results, and refine or debug their queries based on execution feedback (Liu et al., 2025b).

Recent work on tool-augmented RL has shown that LLM agents can improve substantially by interacting with external environments over multiple turns (e.g., search, UI, code execution) rather than acting in a single pass (Jin et al., 2025; Wei et al., 2025a; Hu et al., 2025; Wei et al., 2025b). Such interaction enables agents to gather missing information, refine intermediate hypotheses, and self-correct using feedback, yielding better accuracy and robustness across domains (Liu et al., 2025a; Li et al., 2025b; Cao et al., 2025). Motivated by this paradigm, we propose **SQL-Trail**, a multi-turn RL-trained Text-to-SQL agent. Unlike single-pass RL baselines such as SQL-R1 (Ma et al., 2025), **SQL-Trail** performs iterative database probing, schema exploration, and execution-based self-correction within a closed loop, with difficulty-aware turn budgeting to avoid overthinking on easy queries while allocating more interaction steps to harder ones. We further introduce a composite reward panel that provides dense step-wise guidance, jointly encouraging execution correctness and efficient long-horizon behavior, enabling smaller models to reliably converge to accurate SQL programs.

Our main contributions are as follows: (1) **Unified Multi-turn RL Training Framework:** We present the first end-to-end study of multi-turn LLM training strategies for Text-to-SQL, integrating a novel adaptive turn-budget allocation mechanism that allows the agent to expend more interaction turns on complex queries while remaining concise on simple ones. (2) **Systematic Investigation of Multi-turn Agent Behavior:** We provide a comprehensive comparison between single-pass and multi-turn RL and detailed ablation study of our composite reward panel, revealing how interactive execution fundamentally reshapes reasoning trajectories. We also conduct the first systematic investigation into agent reasoning efficiency under a multi-turn setting. (3) **State-of-the-Art Data Efficiency and Generalization:** Our experiments demonstrate that multi-turn RL achieves state-of-the-art out-of-distribution performance with exceptional data efficiency. Specifically, while prior single-pass methods such as OmniSQL yield a marginal accuracy improvement of 0.005% per 1,000 in-distribution training samples, **SQL-Trail** delivers a 4.6% gain per 1,000 samples using out-of-distribution data. This high-potential learning paradigm enables our 7B model to outperform significantly larger proprietary models by an average of 5% across all benchmarks, establishing new performance records for open-source models at both the 7B and 14B scales.

2 Related Work

Text-to-SQL has progressed from rule-based and template systems to neural semantic parsers such as Seq2SQL (Zhong et al., 2017) and SQLNet (Xu et al., 2017), and more recently to LLM-based approaches. Modern methods leverage in-context learning (Zhang et al., 2023; Agarwal et al., 2024; Sun et al., 2023), improved schema linking (Li et al., 2023; Cao et al., 2024; Snell et al., 2024; Eyal et al., 2023), and constrained decoding (e.g., PICARD) (Scholak et al., 2021) to improve validity and robustness. State-of-the-art systems largely rely on supervised fine-tuning and multi-step self-correction with execution feedback (Li et al., 2025a; Wang et al., 2023; Pourreza et al., 2024; Pourreza and Rafiei, 2024b), but SFT-centric training can still struggle to generalize to unseen schemas and complex queries (Pourreza et al., 2025; Ma et al., 2025).

In parallel, reinforcement learning has emerged as a powerful mechanism for training autonomous agents capable of complex, multi-turn reasoning (Wei et al., 2022; Jaech et al., 2024; Plaat et al., 2024). Recent frameworks such as DeepSeek-R1 (Guo et al., 2025) have demonstrated that algorithms like Group Relative Policy Optimization (GRPO) can incentivize models to internalize intermediate reasoning steps without explicit supervision, achieving strong performance in mathematics and logic. Furthermore, domain-specific implementations like Llama3-SWE-RL (Wei et al., 2025a) and Search-R1 (Jin et al., 2025) have proven that RL agents can self-improve on real-world software engineering tasks and tool-augmented retrieval by interacting dynamically with their environments (Li et al., 2025b; Jiang et al., 2025; Zheng et al., 2025). Despite these advancements in general code generation and agentic workflows, there remains a notable gap in applying deep reasoning RL frameworks to the multi-turn Text-to-SQL setting (Liu et al., 2025b).

3 SQL-TRAIL

Overview We introduce **SQL-TRAIL**, a multi-turn Text-to-SQL agent fine-tuned through a staged training pipeline.

3.1 Generation with Multi-Turn SQL Engine Calling

We design a tool-augmented ReAct-style agent workflow (Yao et al., 2023) in which an LLM agent interacts with the database environment by calling a SQL execution tool and receives structured feedback from the database, enabling it to solve Text-to-SQL tasks through multi-turn reasoning and execution. The framework consists of two core components: *actions*, where the LLM produces a reasoning trace and proposes SQL queries, and *observations*, which are the execution outputs returned to the model.

Formally, for the i-th example, we represent the multi-turn interaction as a trajectory

$$\tau_i = \{(o_{i,t}, a_{i,t})\}_{t=1}^{T_i}, \qquad (1)$$

where $a_{i,t}$ denotes the agent action at turn t (e.g., a reasoning trace and a SQL query to execute), and $o_{i,t}$ denotes the corresponding observation returned by the environment (e.g., execution results, error messages, or a truncated dataframe preview). Here T_i is the number of interaction turns for instance i. A trajectory is thus the ordered sequence of alternating actions and observations induced by iterative SQL engine calls.

Our approach adopts an iterative reason–execute–observe loop, where the LLM alternates between natural-language reasoning and external SQL execution under a strict, token-delimited interface (Cao et al., 2025). At each turn, the model outputs a reasoning block <reasoning>...</reasoning> followed by a SQL action <sql>...</sql>; the system extracts and executes the SQL, then appends the engine output as an observation <observation>...</observation> for the next turn. The process repeats until a turn budget is reached or the model emits the final solution in <solution>...</solution>. Full system prompt details and formatting specifications are provided in Appendix A.3.

3.2 Training phase 1: Supervised Fine-Tuning

Supervised fine-tuning (SFT) provides a crucial initialization step for our agent. Smaller open-weight models often struggle with the long, highly structured prompts required by the above multi-turn agent workflows, frequently producing formatting mistakes, prematurely invoking tools, or entering undesired action loops (Hui et al., 2024b). In contrast, large proprietary models demonstrate strong instruction-following and reliably adhere to the agent interface (Anthropic, 2025). To bridge this gap, we distill the instruction-following behavior of a high-capacity closed-source teacher model into smaller open-source student models, teaching them to execute the agent loop correctly and consistently. This distillation equips the students with a strong prior over the agent's operational structure, enabling stable early rollouts and providing dense formative signals that significantly improve downstream reinforcement learning.

3.3 Training Phase 2: Multi-turn Reinforcement Learning with a SQL Engine

In the second training phase, we apply reinforcement learning (RL) to optimize the LLM as a flexible multi-turn agent that develops a deeper understanding of how to use the SQL execution tool to solve tasks precisely and efficiently. RL encourages the agent to avoid unnecessary detours and to generalize more robustly under distribution shifts in database schemas and unseen domains. To train this multi-turn Text-to-SQL agent, we adopt an RL framework built upon Grouped Reinforcement Policy Optimization (GRPO) (Shao et al., 2024) and extend it with a detailed reward panel tailored for multi-turn reasoning.

3.3.1 Reinforcement Learning Formulation

For each training instance sampled from the dataset $\mathcal{D}$, the input consists of a natural language question q and its associated database schema d.

We distinguish *single-turn* RL, where the policy emits one SQL query in a single step and receives a terminal execution-based reward, from our *multi-turn* RL setting, where the policy interacts with a SQL execution environment over multiple steps. In the multi-turn case, conditioned on (q, d), the rollout engine (Section 3.1) induces a trajectory $\tau = (o_t, a_t)_{t=1}^{T}$, where each action a_t proposes a SQL (or control) step and each observation o_t is structured engine feedback (e.g., results or er-

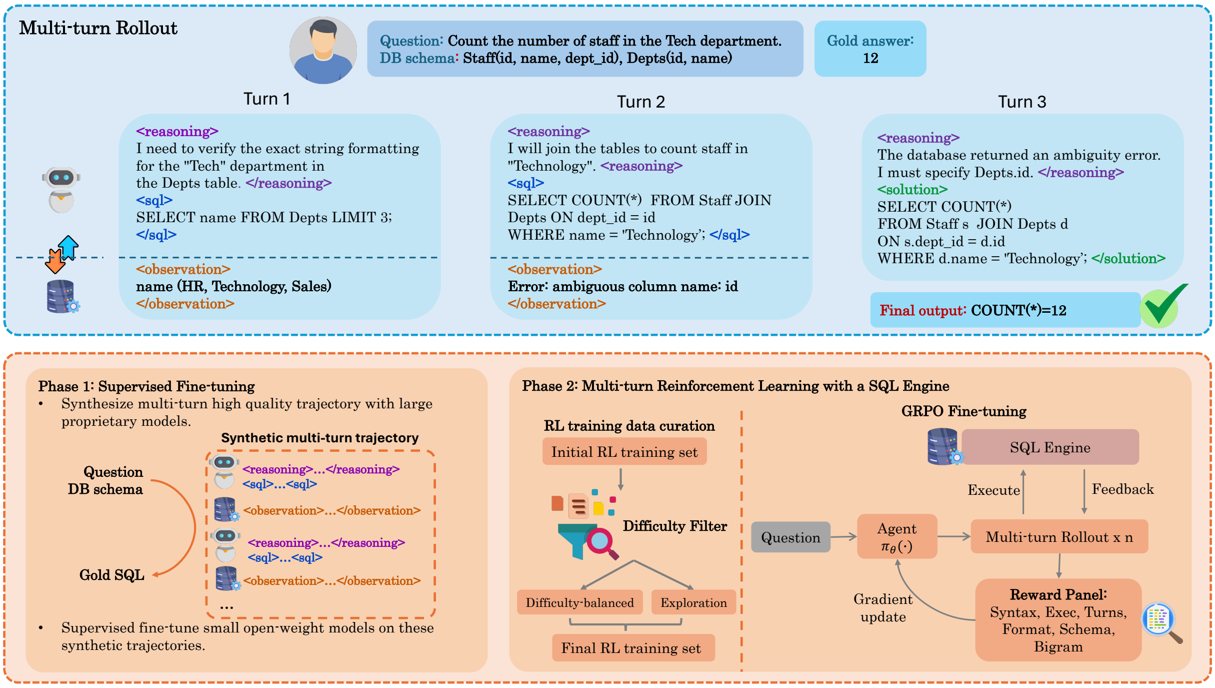

Figure 2: Overview of **SQL-TRAIL**. The top panel demonstrates the multi-turn Text-to-SQL interaction process, and the bottom panel outlines the unified RL training pipeline.

rors); learning is driven by a trajectory-level reward $R(\tau; q, d)$ that combines final execution correctness with intermediate behavioral signals.

To better encourage exploration in the multi-turn setting, we remove the KL regularization term and modify the clipping mechanism used in the original GRPO. Standard GRPO applies symmetric PPO-style clipping (Yu et al., 2025), enforcing identical upper and lower bounds on the likelihood ratio $\rho_i(\theta) = \frac{\pi_\theta(\tau_i|q,d)}{\pi_{\theta_{\text{old}}}(\tau_i|q,d)}$, where $\pi_{\theta_{\text{old}}}$ is the policy before the update step. In contrast, we adopt a clip-higher variant, which keeps the conservative lower bound to maintain training stability but raises the upper bound to allow larger update steps on promising but initially low-likelihood trajectories. Formally, we constrain $\rho_i(\theta)$ in between $1 - \epsilon_{\text{low}}$ and $1 - \epsilon_{\text{high}}$ with $\epsilon_{\text{low}} < \epsilon_{\text{high}}$. By expanding the upper clipping range, the clip-higher strategy increases probability mass on diverse exploratory rollouts while still maintaining controlled updates.

Our objective maximizes the modified GRPO return:

$$\begin{aligned}&\mathcal{J}_{GRPO}(\theta)\\&= \mathbb{E}_{q,d\sim\mathcal{D},\{\tau_i\}_{i=1}^{G}\sim\pi_{\theta_{old}}(\cdot|q,d)}\left[\frac{1}{G}\sum_{i=1}^{G}\min\Big(\rho_i(\theta)A_i,\right.\\&\left.\qquad\mathrm{clip}\big(\rho_i(\theta),\ 1-\epsilon_{\text{low}},\ 1+\epsilon_{\text{high}}\big)A_i\Big)\right].\end{aligned} \quad (2)$$

This formulation preserves GRPO's group-relative optimization while providing stronger encouragement for upward, exploration-driven policy updates.

3.3.2 Multi-turn Reward Design

Reward design is the main driver of effective RL. Prior multi-turn agent RL training largely rely on a binary execution reward (Liu et al., 2025a; Jin et al., 2025), but we find this signal is overly sparse: small mistakes collapse the reward to zero and offer little direction for improvement, especially when training data is limited. To better exploit the available data, we introduce a six-term, rule-based reward panel that provides fine-grained SQL-structural feedback and shapes long-horizon multi-turn behavior.

Final execution reward. Our primary objective is execution correctness, so we use a binary execution reward r_{exec}: we extract the final SQL from `<solution>...</solution>`, execute it, and compare the result to the gold query's execution result,

$$r_{\text{exec}} = \mathbb{1}[\text{exec}(\text{pred\_sql}) = \text{exec}(\text{gold\_sql})] \quad (3)$$

Turn number reward. To encourage efficient multi-turn behavior, we add a turn-budget reward r_{turns} that favors solving within a small number of steps (with thresholds tied to difficulty) and discourages redundant interactions. The exact thresholding rule is provided in Appendix A.4.

Auxiliary shaping rewards. We further include four lightweight rewards to densify supervision and enforce valid behavior: bi-gram overlap (r_{bigram}) and schema grounding (r_{schema}) as Jaccard-based similarity signals, plus binary syntax validity (r_{syntax}) and interface compliance (r_{format}). Full definitions and implementation details are deferred to Appendix A.4.

Total reward. We combine terms with a simple weighted sum,

$$R = 5r_{\text{exec}} + 2r_{\text{turns}} + r_{\text{schema}} + r_{\text{bigram}} + r_{\text{syntax}} + r_{\text{format}}, \tag{4}$$

using larger weights on r_{exec} (end-task objective) and r_{turns} (explicitly shaping multi-turn efficiency), while keeping auxiliary shaping terms at unit weight (stabilization without dominating optimization). The weighting rationale and ablations are included in Appendix A.4.

4 Experiment Setup

4.1 Model setup

We use Qwen2.5-Coder base models (3B/7B/14B) (Hui et al., 2024a) following existing works and a two-stage pipeline: supervised fine-tuning (SFT) followed by RL from the SFT checkpoints. For RL, we sample $G = 6$ rollouts per query (temperature 1.0) with a 10-turn trajectory cap to enable iterative planning and revision. Additional training details are provided in Appendix A.5.

4.2 Training data curation

We use a two-stage pipeline (SFT $\rightarrow$ RL) and deliberately curate a small but informative training set to study data efficiency. Full sampling, filtering, and hyperparameter details are provided in Appendix A.6.

SFT. We sample 3,000 Spider-train questions (Yu et al., 2018), generate multi-turn trajectories with Claude-Sonnet-3.7 using our agent template (Section 3.1), and retain 1,000 trajectories with correct final SQL, priortized toward medium/hard difficulty. These demonstrations distill instruction-following and multi-turn behaviors into the student models. Appendix A.6 lists the exact selection criteria and prompts.

RL. We train on 1,027 questions split into: (i) a **difficulty-balanced set** (700) chosen to maximize informative GRPO advantages by favoring "hard-but-solvable" items (non-degenerate pass@6), and (ii) an **exploration set** (327) of consistently difficult cases to target persistent failure modes (including post-SFT failures and pass@6=0 subsets from SynSQL and Spider). Appendix A.6 details the candidate pools, pass@6 estimation, scoring/ranking, and composition breakdown.

4.3 Benchmarks and Evaluation

Benchmarks and Metrics. We evaluate on Spider (Yu et al., 2018) and BIRD (Li et al., 2024a). For Spider, we report dev and test results to measure in-domain gains, since they match our SFT training distribution; for out-of-distribution generalization, we evaluate on the BIRD dev set, which is held out from all training phases and contains complex, value-centric questions over real-world schemas across 37 domains. We further assess robustness on Spider-DK (Gan et al., 2021b), Spider-Syn (Gan et al., 2021a), and Spider-Realistic (Deng et al., 2021), which probe external-knowledge reliance, lexical shift, and more ambiguous user queries. We report Execution Accuracy (EX), i.e., the fraction of predictions whose execution result matches the ground truth, and measure training efficiency via Data Efficiency, defined as EX gain (percentage points) per 1,000 training examples:

$$\text{efficiency}_{\text{pp/1k}} = \frac{\Delta\text{pp}}{N} \times 1000 \tag{5}$$

where $\Delta\text{EX}_{\text{pp}}$ is the EX improvement in pp and N is the number of training examples.

Inference Strategy. To ensure a fair comparison with prior work, we follow standard test-time evaluation practices: greedy decoding (Pass@1, zero-temperature) and majority voting over 8 sampled candidates using execution-result consensus. We also report Pass@k to characterize the model's best-of-k capability.

5 Results

5.1 Main results

Baseline Comparison Table 1 reports the performance of **SQL-TRAIL** across Spider and BIRD benchmarks, compared against several state-of-the-art Text-to-SQL finetuning systems. We include strong single-pass RL baselines such as SQL-R1 (Ma et al., 2025), trained on 5k synthetic chal-

Text2SQL Method	Training Set (Size)	Spider (dev)		Spider (test)		BIRD (dev)		Data Efficiency (Gre)↑	
		Gre	Maj	Gre	Maj	Gre	Maj	Spider-test	BIRD-dev
Sonnet-3.7									
Single-pass	-	78.3	78.9	82.0	83.2	58.5	60.1	-	-
Multi-turn Agent	-	77.2	77.9	81.9	82.0	60.0	60.8	-	-
Qwen2.5-Coder-3B-Instruct									
Single-pass	-	72.8	77.0	75.1	77.2	45.2	50.5	-	-
SQL-R1-3B	SynSQL(5k)	71.9	78.1	76.5	78.9	48.4	54.6	0.28	0.64
SQL-TRAIL-3B	SynSQL(0.8k)+Spider(1k)	**76.3**	**83.1**	**77.7**	**84.3**	**50.1**	**55.1**	**1.3**	**2.4**
Qwen2.5-Coder-7B-Instruct									
Single-pass	-	73.4	77.1	82.2	85.6	50.9	61.3	-	-
Reasoning-SQL-7B	BIRD-train(8026)	-	-	78.7	-	**64.0**	-	-0.44	1.63
SQL-R1-7B	SynSQL(5k)	81.9	84.5	83.5	86.1	58.9	63.1	0.26	1.6
OminiSQL-7B	SynSQL(2.5M)+BIRD(9.4k)+Spider(8.7k)	81.2	81.6	**87.9**	88.9	63.9	**66.1**	0.002	0.005
SQL-TRAIL-7B	SynSQL(0.8k)+Spider(1k)	**85.2**	**86.8**	86.0	**87.6**	60.1	64.2	**1.90**	**4.60**
Qwen2.5-Coder-14B-Instruct									
Single-pass	-	78.1	80.6	86.6	88.0	61.5	66.1	-	-
Reasoning-SQL-14B	BIRD-train(8k)	-	-	81.4	-	**65.3**	-	-0.1	0.34
SQL-R1-14B	SynSQL(5k)	83.4	86.7	86.1	88.1	63.2	**67.1**	-0.65	0.47
Omini-SQL-14B	SynSQL(2.5M)+BIRD(9.4k)+Spider(8.7k)	81.4	82.0	**88.3**	88.3	64.2	65.9	0.0006	0.001
SQL-TRAIL-14B	SynSQL(0.8k)+Spider(1k)	**85.1**	**87.1**	86.8	**88.5**	63.6	66.7	**0.1**	**1.05**

Table 1: Main EX(%) and data efficiency results. The table is organized into blocks, each headed by the corresponding base model. "Multi-turn Agent" refers to untuned base models initiated with a multi-turn system template; see more details in Appendix A.8. “Gre” denotes greedy decoding and “Maj” denotes majority voting.

lenging SynSQL (Li et al., 2025a) examples, and Reasoning-SQL (Pourreza et al., 2025), trained on over 8k BIRD-train samples. We further compare to OminiSQL (Li et al., 2025a), which performs large-scale SFT on 2.5M synthetic examples and chain-of-thought augmented Spider and BIRD datasets.

Overall, **SQL-TRAIL** reaches state-of-the-art accuracy with an order-of-magnitude gain in data efficiency, trained on just 1,873 examples. On BIRD-dev, **SQL-TRAIL**-7B achieves 60.1% (Greedy) and 64.2% (Majority), outperforming the much larger Sonnet-3.7 by +1.6 and +4.1 points under identical evaluation. Compared with same-scale single-pass RL baselines (SQL-R1, Reasoning-SQL), **SQL-TRAIL** is far more data-efficient: at 7B, it delivers 7–18× higher efficiency on both Spider-test and BIRD-dev.

Generalization and robustness analysis Multi-turn RL substantially improves out-of-distribution generalization. Reasoning-SQL-7B trains on $>$ 8k in-domain BIRD examples yet transfers poorly to Spider-test (78.7% EX), 9.2 points below OminiSQL-7B (87.9%). In contrast, **SQL-TRAIL**-7B is trained only on Spider + SynSQL (never on BIRD) but transfers strongly to BIRD-dev, trailing OminiSQL-7B by just 3.9 points (64.2% vs. 68.1%). This asymmetry suggests BIRD supervision does not reliably transfer to Spider, whereas our multi-turn agent trained on Spider transfers robustly to BIRD.

Model	Spider (Dev)	Spider (Test)	BIRD (Dev)
single-pass RL	82.8	85.1	56.3
multi-turn RL	84.5	86.1	59.3

Table 2: Execution accuracy(%) comparison between single-turn RL and multi-turn RL under identical training configurations. Results are reported using majority voting.

Against SQL-R1-7B (same SynSQL source), **SQL-TRAIL**-7B is more cross-domain robust on BIRD-dev (+3.7 Greedy, +1.1 Majority) while using less than half the data. Compared with OminiSQL—which trains on both Spider and BIRD—**SQL-TRAIL** still leads by $>$ 5 points on Spider-dev at 7B (86.8% vs. 81.6%) and 14B (87.1% vs. 82.0%), and matches or exceeds OminiSQL on BIRD-dev at the 14B scale despite BIRD being unseen for **SQL-TRAIL**. We attribute this transferability to active environment probing, which helps the agent infer schema structure and user intent across unseen domains.

For robustness, **SQL-TRAIL**-7B matches or outperforms other 7B baselines on Spider-DK, Spider-Syn, and Spider-Realistic (Appendix A.7, Table 5), with the largest gains on Spider-Syn, reinforcing the advantage of multi-turn RL under schema perturbations and noisy queries.

Single-pass and Multi-turn RL comparison In this section, we analyze why multi-turn RL outperforms single-pass RL. We train both settings on the same RL dataset with identical hyperparameters

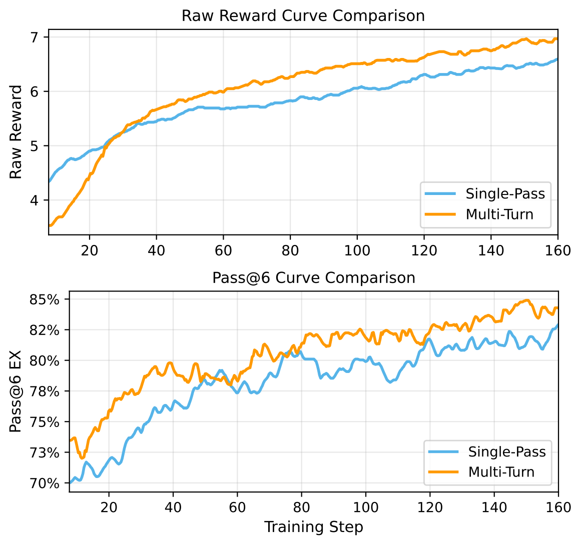

Figure 3: Training dynamics for Single-Pass vs. Multi-Turn RL. Curves represent the 10-step moving average of raw rewards (top) and Pass@6 accuracy (bottom).

and the same Qwen2.5-Coder-7B base model (no SFT), and evaluate majority-vote EX on Spider-dev/test and BIRD-dev (Table 2). Because single-pass RL lacks interaction, we disable the turn-count reward in both settings and keep all other reward components fixed. Multi-turn RL consistently wins across benchmarks, with the largest gain on cross-domain BIRD-dev (+3pp, 56.3%→59.3%), indicating stronger OOD generalization.

Figure 3 sheds light on the mechanism. Single-pass RL improves rapidly but saturates early, suggesting a lower capability ceiling. Multi-turn RL continues to scale, consistent with richer feedback from interactive execution and a longer optimization horizon. Notably, Pass@6 favors multi-turn RL throughout training, even when its reward curve is initially lower, implying that multi-turn training induces more diverse and exploratory trajectories rather than optimizing a single deterministic chain.

Qualitative examples in Appendix A.7 further support this: multi-turn agents handle ambiguous questions and complex schemas by iteratively revising hypotheses and verifying via execution, while single-pass RL lacks these self-correction behaviors. Together, these results explain the higher headroom, broader search, and stronger generalization of multi-turn RL.

5.2 Post-training analysis

In this section, we examine how post-training improves the base model through a series of extensive ablation studies. We analyze three key components: (1) the contribution of each reward in the RL stage, (2) how RL enhances the model's reasoning ability, and (3) the effect of SFT as a cold-start initializer.

5.2.1 Ablation study

We perform a comprehensive reward and post-training stage ablation study on BIRD-dev in Appendix A.8 Table 6 to isolate the contribution of each component in our reinforcement learning framework. For SQL quality evaluation, we report average syntax accuracy, schema similarity, and bi-gram similarity, with calculation details the same as presented in Appendix A.4. To assess the multi-turn behavior of the agents, we also track the average number of conversation turns required to reach a solution. We compare the performance of our method against the Qwen2.5-Coder-7B base model, the post-SFT checkpoint, and Sonnet 3.7 as a large closed-source reference.

Reward ablation The reward ablation results show that the bi-gram similarity reward provides the largest individual gain among partial rewards. Beyond encouraging surface overlap, bi-gram matching supplies a dense learning signal at the token-transition level, which reduces reward variance and steers decoding toward stable SQL skeletons early in optimization. In contrast, the execution reward is more sparse and less informative. As a result, improving bi-gram overlap—especially around high-impact transitions such as column references, JOIN–ON patterns, and GROUP BY/aggregation templates—more directly corrects the structural failure modes that dominate execution errors. Counterintuitively, format and execution rewards contribute only marginal gains once strong bi-gram, schema, and syntax constraints are present, suggesting substantial redundancy among these signals in the high-constraint regime.

Post-training ablation Post-training ablations show scale-dependent behaviors. Larger models (e.g., Sonnet) are syntactically strong but weak at schema linking, and both Sonnet and base Qwen generate overly long trajectories. Without multi-turn post-training, models lack robust tool-use and multi-step planning: Sonnet wastes turns probing the schema, while base Qwen repeatedly revises flawed SQL due to weaker syntax. Comparing base-Qwen → SFT → RL highlights a clear progression: SFT improves syntax accuracy and shortens trajectories, but schema linking remains difficult; RL further improves schema identification and drives convergence to correct solutions within a tight turn range, indicating more efficient schema linking and reliable SQL generation. Appendix A.8 reports matched-condition test-time compute comparisons (diversity vs. consistency) and a cold-start SFT ablation across scales.

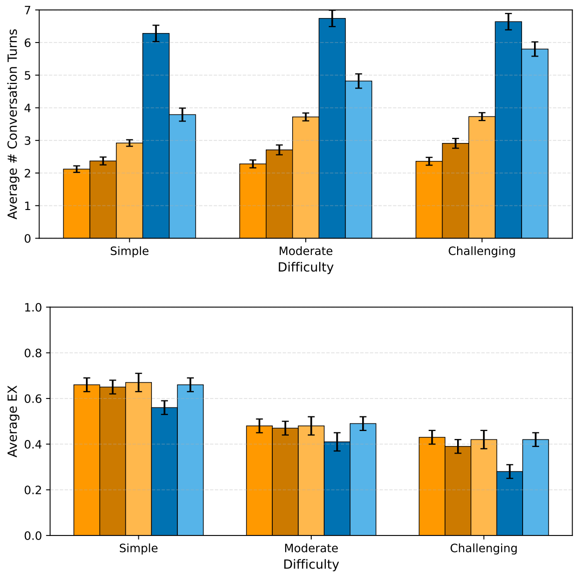

Figure 4: Performance comparison on the BIRD-dev set across three difficulty levels. The top panel reports the average number of turns, while the bottom panel displays the Average Execution Accuracy (EX). The models evaluated are **SQL-Trail RL**, **SQL-Trail SFT**, **SQL-Trail RL w/o turn reward**, **Qwen2.5-Coder Agent**, and **Sonnet Agent**.

Turn-Efficiency We further investigate how RL and the turn-efficiency reward demonstrate a clear impact on interaction behavior. As shown in Appendix A.8 Table 6, removing it causes the model to engage in longer, unnecessary reasoning chains without yielding higher accuracy, confirming that this signal suppresses over-thinking and encourages concise trajectories. Beyond simple turn reduction, Figure 4 shows that the reward enables adaptive turn budgeting: the agent takes more turns on harder queries and fewer on easier ones. It's also worth noticing that RL-trained **SQL-Trail** shows advantages in solving challenging problems exceeding Sonnet-3.7 with significantly fewer turns. This difficulty-aware allocation improves execution accuracy by ensuring that the model expends effort only where needed, rather than uniformly across tasks. In Appendix A.8 Figure 10, we present a side-by-side comparison where **SQL-Trail** solves the problem efficiently, whereas Sonnet exhibits significant overthinking, yielding unnecessarily long trajectories that ultimately result in incorrect predictions.

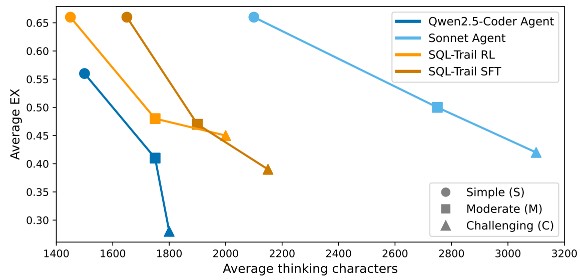

Figure 5: Reasoning efficiency analysis on BIRD-dev. We plot Execution Accuracy against the average quantity of reasoning (measured in characters within <reasoning>tags) across different difficulty levels

5.2.2 Reasoning analysis

To assess reasoning efficiency, we analyze the relationship between thinking-token length and execution accuracy in Figure 5. As expected, harder questions generally require longer reasoning traces to recover the correct SQL logic. However, base models fail to navigate this trade-off: Sonnet tends to overthink with diminishing returns, while Qwen-base frequently underthinks. In contrast, **SQL-Trail** achieves balanced reasoning efficiency—reaching the highest execution accuracy with substantially fewer reasoning tokens, especially after RL optimization. This conclusion is further supported by our analysis of reasoning length versus schema-link complexity in Appendix A.8 Figure 7. We further include analysis of reasoning efficiency evolution in Appendix A.8 Figure 8.

6 Conclusion

We introduce **SQL-Trail**, a unified multi-turn RL framework that reframes Text-to-SQL from static translation into interactive reasoning. Using difficulty-aware rewards and targeted data selection, **SQL-Trail** enables open-source models to explore schemas, correct errors via execution feedback, and adapt their turn budget to query complexity. **SQL-Trail** sets a new state of the art in data efficiency and out-of-distribution generalization, matching or outperforming larger proprietary systems on BIRD-dev with fewer than 2,000 training samples. These results highlight the importance of iterative environment interaction for robust, self-correcting database agents.

7 Limitations

Despite strong gains, **SQL-TRAIL** has several important limitations. First, it assumes an interactive execution environment: the agent must be able to run (possibly multiple) SQL queries against the target database to obtain errors and results. This requirement may be infeasible in settings with restricted connectivity, strict privacy controls, or expensive query execution. Second, multi-turn interaction increases inference cost and latency (more tokens, more database calls). While difficulty-aware turn budgeting reduces unnecessary steps, worst-case overhead remains higher than single-pass systems. Third, parts of the training recipe rely on supervision signals that may not be available at scale in new domains. Reward shaping can also introduce inductive biases (e.g., favoring syntactic/structural similarity over alternative but equivalent SQL), and RL optimization may exploit spurious correlations in the training distribution. Finally, our empirical study is centered on Spider/BIRD and related robustness suites; these benchmarks do not fully represent production constraints and real-world use cases. As a result, additional validation is needed to establish reliability and cost-quality trade-offs in deployed environments.

References

Rishabh Agarwal, Avi Singh, Lei M Zhang, Bernd Bohnet, Luis Rosias, Stephanie C.Y. Chan, Biao Zhang, Aleksandra Faust, and Hugo Larochelle. 2024. Many-shot in-context learning. In *ICML 2024 Workshop on In-Context Learning*.

Anthropic. 2025. Claude 3.7 Sonnet System Card. `https://www.anthropic.com/claude-3-7-sonnet-system-card`. Accessed: November 24, 2025.

Léonard Blier and Yann Ollivier. 2021. Unbiased methods for multi-goal reinforcement learning. *arXiv preprint arXiv:2106.08863*.

Shiyi Cao, Sumanth Hegde, Dacheng Li, Tyler Griggs, Shu Liu, Eric Tang, Jiayi Pan, Xingyao Wang, Akshay Malik, Graham Neubig, Kourosh Hakhamaneshi, Richard Liaw, Philipp Moritz, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. 2025. Skyrl-v0: Train real-world long-horizon agents via reinforcement learning.

Zhenbiao Cao, Yuanlei Zheng, Zhihao Fan, Xiaojin Zhang, Wei Chen, and Xiang Bai. 2024. Rsl-sql: Robust schema linking in text-to-sql generation. *arXiv preprint arXiv:2411.00073*.

Albert Chen, Manas Bundele, Gaurav Ahlawat, Patrick Stetz, Zhitao Wang, Qiang Fei, Donghoon Jung, Audrey Chu, Bharadwaj Jayaraman, Ayushi Panth, and 1 others. 2025. Text-to-sql for enterprise data analytics. *arXiv preprint arXiv:2507.14372*.

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*, pages 1337–1350, Online. Association for Computational Linguistics.

Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Jinshu Lin, Dongfang Lou, and 1 others. 2023. C3: Zero-shot text-to-sql with chatgpt. *arXiv preprint arXiv:2307.07306*.

Ahmed Elgohary, Saghar Hosseini, and Ahmed Hassan Awadallah. 2020. Speak to your parser: Interactive text-to-SQL with natural language feedback. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2065–2077, Online. Association for Computational Linguistics.

Ben Eyal, Amir Bachar, Ophir Haroche, Moran Mahabi, and Michael Elhadad. 2023. Semantic decomposition of question and sql for text-to-sql parsing. *arXiv preprint arXiv:2310.13575*.

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. *arXiv preprint arXiv:2109.05157*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Xueyu Hu, Tao Xiong, Biao Yi, Zishu Wei, Ruixuan Xiao, Yurun Chen, Jiasheng Ye, Meiling Tao, Xiangxin Zhou, Ziyu Zhao, Yuhuai Li, Shengze Xu, Shenzhi Wang, Xinchen Xu, Shuofei Qiao, Zhaokai Wang, Kun Kuang, Tieyong Zeng, Liang Wang, and 10 others. 2025. OS agents: A survey on MLLM-based agents for computer, phone and browser use. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7436–7465, Vienna, Austria. Association for Computational Linguistics.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun

Zhang, Bowen Yu, Kai Dang, and 1 others. 2024a. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024b. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Pengcheng Jiang, Jiacheng Lin, Lang Cao, Runchu Tian, SeongKu Kang, Zifeng Wang, Jimeng Sun, and Jiawei Han. 2025. Deepretrieval: Hacking real search engines and retrievers with large language models via reinforcement learning. *arXiv preprint arXiv:2503.00223*.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*.

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

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 13067–13075.

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

Zhishuai Li, Xiang Wang, Jingjing Zhao, Sun Yang, Guoqing Du, Xiaoru Hu, Bin Zhang, Yuxiao Ye, Ziyue Li, Rui Zhao, and 1 others. 2024b. Pet-sql: A prompt-enhanced two-stage text-to-sql framework with cross-consistency. *arXiv preprint arXiv:2403.09732*.

Zhuofeng Li, Haoxiang Zhang, Seungju Han, Sheng Liu, Jianwen Xie, Yu Zhang, Yejin Choi, James Zou, and Pan Lu. 2025b. In-the-flow agentic system optimization for effective planning and tool use. *arXiv preprint arXiv:2510.05592*.

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. 2025a. Skyrl-sql: Matching gpt-4o and o4-mini on text2sql with multi-turn rl. Notion Blog.

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

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*.

OpenAI. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Aske Plaat, Annie Wong, Suzan Verberne, Joost Broekens, Niki van Stein, and Thomas Back. 2024. Reasoning with large language models, a survey. *arXiv preprint arXiv:2407.11511*.

Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O Arik. 2024. Chase-sql: Multi-path reasoning and preference optimized candidate selection in text-to-sql. *arXiv preprint arXiv:2410.01943*.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *arXiv preprint arXiv:2304.11015*.

Mohammadreza Pourreza and Davood Rafiei. 2024a. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems*, 36.

Mohammadreza Pourreza and Davood Rafiei. 2024b. Dts-sql: Decomposed text-to-sql with small large language models. *arXiv preprint arXiv:2402.01117*.

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*.

Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, Binhua Li, Ruiying Geng, Rongyu Cao, Jian Sun, Luo Si, and 1 others. 2022. A survey on text-to-sql parsing: Concepts, methods, and future directions. *arXiv preprint arXiv:2208.13629*.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. Picard: Parsing incrementally for constrained auto-regressive decoding from language models. *arXiv preprint arXiv:2109.05093*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.

Ruoxi Sun, Sercan Ö Arik, Rajarishi Sinha, Hootan Nakhost, Hanjun Dai, Pengcheng Yin, and Tomas Pfister. 2023. Sqlprompt: In-context text-to-sql with minimal labeled data. *arXiv preprint arXiv:2311.02883*.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2024. Mac-sql: A multi-agent collaborative framework for text-to-sql. *Preprint*, arXiv:2312.11242.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Qian-Wen Zhang, Zhao Yan, and Zhoujun Li. 2023. Mac-sql: Multi-agent collaboration for text-to-sql. *arXiv preprint arXiv:2312.11242*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I Wang. 2025a. Swe-rl: Advancing llm reasoning via reinforcement learning on open software evolution. *arXiv preprint arXiv:2502.18449*.

Zhepei Wei, Wenlin Yao, Yao Liu, Weizhi Zhang, Qin Lu, Liang Qiu, Changlong Yu, Puyang Xu, Chao Zhang, Bing Yin, Hyokun Yun, and Lihong Li. 2025b. Webagent-r1: Training web agents via end-to-end multi-turn reinforcement learning. *Preprint*, arXiv:2505.16421.

Xiaojun Xu, Chang Liu, and Dawn Song. 2017. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Zhewei Yao, Guoheng Sun, Lukasz Borchmann, Gaurav Nuti, Zheyu Shen, Minghang Deng, Bohan Zhai, Hao Zhang, Ang Li, and Yuxiong He. 2026. Arctic-text2sql-r1: Simple rewards, strong reasoning in text-to-sql. *Preprint*, arXiv:2505.20315.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, YuYue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, LingJun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, and 17 others. 2025. DAPO: An open-source LLM reinforcement learning system at scale. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, and 1 others. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*.

Hanchong Zhang, Ruisheng Cao, Lu Chen, Hongshen Xu, and Kai Yu. 2023. Act-sql: In-context learning for text-to-sql with automatically-generated chain-of-thought. *Preprint*, arXiv:2310.17342.

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. 2025. Deepresearcher: Scaling deep research via reinforcement learning in real-world environments. *Preprint*, arXiv:2504.03160.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

A Appendix

A.1 Potential Risks

Deploying multi-turn Text-to-SQL agents introduces privacy, security, and operational risks. Because **SQL-TRAIL** explicitly queries the database during inference, it may surface sensitive information via returned rows, error messages, or intermediate tool traces. This risk is amplified by multi-turn exploration, which can adaptively probe schemas and values. To mitigate this, deployments should enforce least-privilege access (ideally read-only), redact or aggregate sensitive outputs, and log/access-audit queries and results with appropriate retention policies.

There are also security and misuse risks: adversarial schema/table names or database contents can act as prompt-injection vectors that steer the agent toward unsafe behavior, and the agent could generate destructive or exfiltrative SQL if permissions allow (e.g., `UPDATE`/`DROP`, wide table scans). Practical safeguards include strict SQL allow-lists (e.g., `SELECT`-only), static query analysis, query cost estimation, timeouts, rate limits, and sandboxed execution. Operationally, multi-turn agents can increase database load and may inadvertently issue expensive queries, creating denial-of-service-style failures under concurrency. Finally, as with many LLM systems, performance may vary across domains and languages underrepresented in training/evaluation, potentially creating unequal reliability across user groups; careful domain-specific evaluation and monitoring are required before high-stakes use.

A.2 Dataset statistics

Table 3 summarizes the dataset sizes used in this work. Spider and BIRD provide standard train/dev/test splits for supervised training and evaluation, with BIRD's test split typically kept hidden for leaderboard evaluation. SynSQL-2.5M is a large-scale synthetic corpus released primarily as a training set, which we use for training but do not treat as a canonical benchmark dev/test split.

Dataset	Train	Dev	Test	Total
Spider	7,000	1,034	2,147	10,181
BIRD	9,428	1,534	1,789	12,751
SynSQL-2.5M	2,536,035	–	–	2,536,035

Table 3: Dataset sizes (question–SQL pairs). SynSQL-2.5M is released primarily as a large synthetic training corpus in the public release we use, without a canonical dev/test split.

A.3 Multi-turn agentic system design.

Our approach follows an iterative framework where LLM can alternate between natural-language reasoning and external SQL execution in a closed loop (Cao et al., 2025). The system instruction enforces a strict interface: at each intermediate turn, after receiving the current environment input, the model must begin its response with a block of reasoning enclosed in `<reasoning>` and `</reasoning>`. It must then place its proposed SQL query for that step between `<sql>` and `</sql>` at the end of the response. Once these tokens appear, the system extracts the enclosed SQL and forwards it to the SQL engine. The resulting execution output is wrapped between `<observation>` and `</observation>` and appended to the conversation as user input for the next iteration. At every turn, the LLM conditions on the full history of past actions and observations to produce its next action, consisting of a new reasoning trace and SQL query. This loop continues until either a maximum turn limit is reached or the model outputs a final answer by enclosing its completed SQL solution between `<solution>` and `</solution>`. We guide the initial LLM to follow our predefined instructions using a detailed system prompt template, as shown in Appendix A.3. This generation procedure serves as the central inference scaffold for both rollout during training and evaluation. By enforcing strict formatting rules, it constrains the model's behavior and ensures accurate reward assignment during the RL training stage.

As shown in System Prompt A.8, we explicitly instruct the LLM to first identify all relevant tables and columns from the provided database schema before issuing any SQL query. In particular, the model must list the tables and fields it plans to use in its initial <think> block, ensuring that subsequent reasoning and tool calls are grounded in the schema rather than hallucinated structures. The prompt further enforces that the final `SELECT` clause only includes columns explicitly requested in the natural language question, preventing over-selection and spurious attributes. Together, these constraints encourage disciplined schema selection and promote faithful, schema-aware SQL generation throughout the multi-turn interaction.

We provide our multi-turn generation workflow in Algorithm 1. The procedure iteratively rolls out the policy's token-level decisions, alternates between model reasoning and SQL tool calls, and injects execution feedback back into the dialogue history to guide subsequent turns. Notably, when returning SQL execution results, we deliberately serialize data frames with column headers included, ensuring the agent receives richer schema context during database probing and enabling more reliable schema linking in later turns.

During the prompt and workflow optimization stage, we evaluate our prompt-design choices using Sonnet-3.7 on BIRD-dev (Table 4). Our ablation progressively introduces database schema details, column headers in SQL execution observations, and explicit schema-selection instructions in the system prompt. The results highlight two key findings: (i) providing rich schema context through interactive database exploration substantially improves grounding, and (ii) enforcing schema-selection constraints keeps the model's choices consistent throughout the trajectory.

Agent prompt design	BIRD (Dev)
w/ Schema + DFCols + Selection guiding	59.7
w/ Schema + DFCols	58.8
w/ Schema	55.9
w/o Schema	28.4

Table 4: Execution accuracy (%) on BIRD-dev under different agent prompt designs. "Schema" indicates whether the full database schema is included in the initial user prompt. "DFCols" denotes whether column headers are shown in the SQL execution return. "Selection guiding" refers to adding instructions in the system prompt to enforce strict selection of relevant tables and columns.

A.4 Detailed Reward Formulations

In this section, we provide the specific definitions and calculation methods for the four reward terms introduced in the main text.

Final execution reward The primary objective of the text-to-SQL task is to generate a query that retrieves the correct answer from the database. To reflect this, we introduce execution reward (r_{exec}). For each generated trajectory, we extract the final SQL solution from `<solution>...</solution>` tokens, and then we execute this query and compare the generated results with the gold SQL query execution results. If they are identical, we assign 1 to the execution reward term. This is the formal definition of it:

$$r_{\text{exec}} = \mathbb{1}[\text{exec}(\text{pred\_sql}) = \text{exec}(\text{gold\_sql})] \quad (6)$$

Turn number reward The length of conversation is critical to long-horizon multi-turn agent behavior. To encourage efficiency and discourage the agent from engaging in redundant reasoning steps, we introduce a turn-based reward (r_{turns}). This term penalizes the agent for exceeding predetermined turn budget T . The reward is conditional on staying within the budget and, for harder queries, achieving correct execution. The formulation is given by:

$$r_{\text{turns}} = \begin{cases} 1, & d = \text{simple} \land t \le 2, \\ 1, & d = \text{medium} \land t \le 3, \\ 1, & d \in \{\text{hard}, \text{extra}\} \\ & \land r_{\text{exec}} = 1 \land t < T, \\ 0, & \text{otherwise.} \end{cases} \quad (7)$$

where t is the finishing turn number of the current trajectory.

Syntax Correctness Reward A fundamental prerequisite for any generated code is that it must be executable by the database engine. To provide an early learning signal that distinguishes between valid SQL queries and hallucinated strings that violate SQL grammar, we utilize a syntax correctness reward r_{syntax}. This is a binary indicator function that returns 1 if the predicted SQL is valid and executable, and 0 otherwise:

$$r_{\text{syntax}} = \mathbb{1}[\text{pred\_sql is executable}] \quad (8)$$

N-gram Similarity Reward Binary execution rewards are often too sparse as a generated query might be semantically close to the solution but fail to execute due to minor token mismatches. To provide a denser training signal, we incorporate an N-gram similarity reward r_{ngram}. This metric measures the lexical overlap between the bigrams ($n = 2$) of the predicted query set B_{pred} and the gold query set B_{gold} using Jaccard similarity:

$$r_{\text{ngram}} = \frac{|B_{\text{pred}} \cap B_{\text{gold}}|}{|B_{\text{pred}} \cup B_{\text{gold}}|} \quad (9)$$

For example, consider a gold query `SELECT name FROM student` and a predicted query `SELECT name FROM teacher`. The set of bigrams for the gold query is {`SELECT name`, `name FROM`, `FROM` student}, while the prediction yields {SELECT name, name FROM, FROM teacher}. The intersection contains two bigrams and the union contains four distinct bigrams, resulting in a reward score of $2/4 = 0.5$.

Schema Linking Reward A common failure mode in text-to-SQL generation is the hallucination of table or column names. To enforce strict grounding, we calculate a schema linking reward r_{schema} based on the Jaccard similarity between the set of schema items (tables and columns) appearing in the prediction S_{pred} and the gold label S_{gold}:

$$r_{\text{schema}} = \frac{|S_{\text{pred}} \cap S_{\text{gold}}|}{|S_{\text{pred}} \cup S_{\text{gold}}|} \tag{10}$$

For instance, if the gold query retrieves data from the table Employees and column Salary, the gold schema set is {Employees, Salary}. If the model correctly identifies Employees but hallucinates a column Wages, the predicted set is {Employees, Wages}. The intersection is {Employees} and the union is {Employees, Salary, Wages}, yielding a reward of $1/3$.

Format Reward To ensure the model adheres to the formatting constraints specified in the system prompt, which is crucial for supporting the multi-turn agent workflow and downstream parsing, we grant reward only when the output contains well-formed tags. Specifically, the output must include \think and \thinkend to delimit the reasoning process, as well as \sol and \solend to encapsulate the final SQL solution:

$$r_{\text{format}} = \mathbb{1}[\text{final output has correct format}] \tag{11}$$

Total Reward The final reward R is a weighted sum:

$$\begin{aligned} Reward = &5r_{\text{exec}} + 2r_{\text{turns}} \\ &+ r_{\text{schema}} + r_{\text{bigram}} + r_{\text{syntax}} + r_{\text{format}} \end{aligned} \tag{12}$$

We set these weights empirically, with a significantly larger weight on execution since execution correctness is the primary objective and the most reliable learning signal for end-task performance. We assign a slightly larger weight to the turn-budget term to explicitly shape multi-turn agent behavior (encouraging concise yet sufficient interactions). All remaining auxiliary shaping terms are given unit weight, reflecting no strong prior preference among them; they serve mainly to stabilize training and provide additional guidance without dominating optimization.

A.5 Training Configuraiton.

For the SFT stage, we train with a batch size of 128 for two epochs and adopt an optimizer configuration consisting of a learning rate of 1×10^{-5}, betas of $(0.9, 0.95)$, a weight decay of 0.01, a warmup ratio of 0.1, gradient clipping at 1.0, and a cosine learning-rate schedule. For the RL stage, we initialize from the SFT checkpoint and continue training with the same batch size of 128, using a learning rate of 1×10^{-6} and top-$p = 0.99$ sampling for rollouts. Each question is expanded with a rollout group size of six under a maximum turn budget of ten, and we report evaluation results at step 108.

A.6 Training data curation

In our training pipeline, we applied a two-stage process: Supervised Fine-Tuning (SFT) followed by Reinforcement Learning (RL). To maximally utilize information from a relatively small dataset and investigate the data efficiency of multi-turn RL compared to other data-hungry fine-tuning methods, we deliberately filtered for informative training samples and carefully balanced their difficulty.

SFT For supervised fine-tuning, we begin by randomly sampling 3,000 training examples from Spider-train (Yu et al., 2018). We then run inference with Claude-Sonnet-3.7 (Anthropic, 2025) using our multi-turn agent template from Section 3.1, allowing up to 10 turns per example. From these synthetic trajectories, we select 1,000 that produce correct final SQL predictions, prioritizing examples with moderate to high difficulty under Spider's difficulty classification. These curated trajectories serve as high-quality demonstrations for SFT, enabling the student base models to distill strong instruction-following and multi-turn reasoning behaviors from a more capable closed-source model.

RL For the RL stage, we constructed a dataset of 1,027 samples. To ensure data-effective learning and robust reasoning, we divided this data into two strategic categories: a difficulty-balanced set and an expanded exploration set.

The **Difficulty-balanced Set** (700 samples) is curated to prioritize "hard yet solvable" instances. As detailed in the GRPO formulation in Sec 3.1, the optimization process relies on the relative advantage within a group of sampled outputs. If a training input yields all correct or all incorrect responses across the sampling group, the reward variance is zero. Consequently, these samples provide no useful gradient signal, failing to utilize the dataset effectively. Furthermore, a high prevalence of such samples decreases the valid batch size, leading to training instability. To mitigate this, we sampled 2,800 candidates from the synthetic text-to-SQL corpus SynSQL-2.5M, prioritizing samples from challenging difficulty level, and evaluated the SFT model using $G = 6$ stochastic generations per question. We computed a difficulty score, $\mathcal{S}(q)$, to identify items that are neither trivial nor impossible:

$$\mathcal{S}(q) = \begin{cases} \text{pass@}G(q) & \text{if } 0 < \text{pass@}G(q) < 1 \\ 1 & \text{otherwise} \end{cases} \tag{13}$$

By ranking samples in ascending order of $\mathcal{S}(q)$ and selecting the top 700, we focus on problems that are hard yet solvable where the model succeeds occasionally (e.g., pass@6 $\approx 1/6$) but does not solve them reliably. This filtering removes both overly easy and unsolvable cases, ensuring dense, informative advantage signals that improve optimization stability and data efficiency.

The **Exploration Set** (327 samples) targets challenging edge cases that encourage exploration and help correct persistent error modes. It is constructed by combining three sources of difficult instances: Post-SFT failures—127 questions from the original SFT training set that the SFT model still fails to solve; a subset of 100 challenging SynSQL examples with pass@6 = 0; and an additional 100 extra-hard Spider-train questions with pass@6 = 0. Together, these samples capture diverse but reliably difficult behaviors that the model must learn to overcome during RL training.

A.7 Multi-turn v.s. Single-pass

To further evaluate generalization under distribution shift, we assess **SQL-TRAIL** on three robustness benchmarks: **Spider-DK**, which injects domain knowledge into a subset of databases and stresses joint schema–knowledge reasoning; **Spider-Syn**, which rewrites questions with synonyms to test robustness to lexical variation; and **Spider-Realistic**, which removes schema-name leakage to approximate more natural user phrasing. We compare against strong single-pass baselines, including SQL-R1 and Reasoning-SQL—both trained with single-pass GRPO—as well as OminiSQL, which relies on single-pass SFT.

As shown in Table 5, **SQL-TRAIL** consistently matches or surpasses all single-pass baselines across all robustness settings. The largest

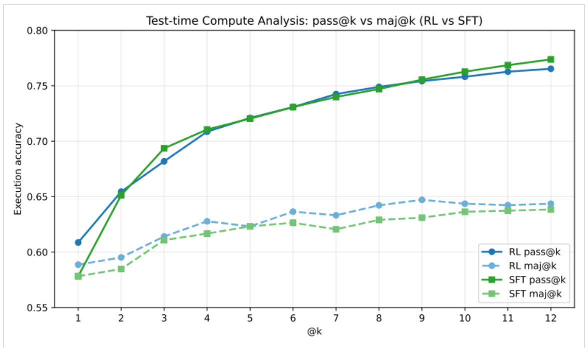

Figure 6: Test-time compute analysis

gains arise on **Spider-Syn**, where the multi-turn formulation improves accuracy by a substantial margin, indicating superior resilience to lexical perturbations. Notably, **SQL-TRAIL** also achieves the best performance on **Spider-Realistic**, demonstrating strong grounding when explicit schema cues are removed. These results collectively highlight that multi-turn RL provides intrinsic advantages over single-pass formulations, particularly when the model must reason over noisy, perturbed, or partially obscured database schemas.

As shown in Figure 10, we provide side-by-side examples of a single-pass LLM and a multi-turn agent. The qualitative comparison illustrates that multi-turn agents handle difficult questions and noisy or ambiguous databases far more effectively. Notably, in the multi-turn trajectory, the agent initially makes the same mistake as the single-pass model on the second turn. However, instead of committing to this incorrect answer, it reflects on the question, conducts additional probing, and carefully re-examines the schema. This iterative refinement allows the multi-turn agent to correct its earlier error and ultimately arrive at the correct solution—behavior that single-pass LLMs are unable to exhibit.

A.8 Additional experiments

Test-time compute. We analyze the impact of test-time compute scaling in Figure 6, comparing the SFT baseline against the RL-tuned model. As the number of samples k increases, the SFT model eventually overtakes the RL model on Pass@k, indicating that supervised fine-tuning preserves greater sample diversity. Conversely, the RL model consistently outperforms SFT on Majority@k, suggesting that reinforcement learning encourages the model to converge on correct solutions with higher consistency, albeit at the cost of diversity.

Setup		Spider-DK		Spider-Syn		Spider-Realistic	
Base LLM	**Method**	**Gre**	**Maj**	**Gre**	**Maj**	**Gre**	**Maj**
Qwen2.5-Coder-7B	Multi-turn Agent	67.5	73.6	63.1	66.9	66.7	70.5
	SQL-R1	–	**78.1**	–	76.7	–	83.3
	Reasoning-SQL	73.3	–	69.3	–	–	–
	Omini-SQL	76.1	77.8	69.7	69.6	76.2	78.0
	SQL-TRAIL	**76.8**	**78.1**	**72.8**	**77.0**	**79.6**	**83.9**

Table 5: Robustness Benchmark experiment. "Multi-turn Agent" means non-finetuned base LLM initiated with multi-turn system prompt.

Setup		Quality Metrics (%)			Conversation Turns		BIRD-dev EX (%)	
Model	**Reward**	**Syntax Acc↑**	**Schema Sim↑**	**Bi-gram Sim↑**	**Avg**	**Std**	**Gre↑**	**Maj↑**
Multi-turn Sonnet 3.7	–	**98.6**	81.9	63.3	4.24	2.22	60.0	60.8
Multi-turn Qwen2.5-Coder-7B	–	89.6	78.3	**70.3**	6.44	3.81	49.1	51.2
Post-SFT **SQL-TRAIL**-7B	–	97.8	85.1	66.7	2.53	1.51	57.8	58.7
SQL-TRAIL-7B	all rewards	97.8	**86.2**	67.7	2.26	0.65	**60.1**	**64.2**
	- w/o turns	97.4	85.5	67.5	3.19	2.12	59.3	63.2
	- w/o ngram	97.3	85.1	66.9	2.35	0.97	57.2	61.6
	- w/o schema	96.4	84.7	66.5	2.28	0.69	58.5	62.6
	- w/o syntax	96.9	85.7	67.5	2.28	0.72	59.1	63.8
	- w/o format	97.4	85.3	67.1	2.28	0.91	59.8	62.9
	- w/o execution	97.5	85.1	67.3	2.21	0.64	57.9	62.8

Table 6: Ablation study. We compare variants of **SQL-TRAIL**-7B fine-tuned with different post-training stages and reward configurations.

Model-Size	w/ SFT	w/ RL	Spider (Dev)	Spider (Test)	BIRD (Dev)
SQL-TRAIL-3B	✓	✗	83.1	82.9	55.7
	✗	✓	84.6	83.1	55.2
	✓	✓	84.6	84.3	55.2
SQL-TRAIL-7B	✓	✗	83.6	83.5	58.7
	✗	✓	85.8	86.8	61.7
	✓	✓	86.5	87.0	64.2
SQL-TRAIL-14B	✓	✗	83.3	86.1	64.8
	✗	✓	86.9	87.2	65.4
	✓	✓	87.1	88.5	66.7

Table 7: Execution accuracy (%) of models with different cold-start strategies (SFT and RL).

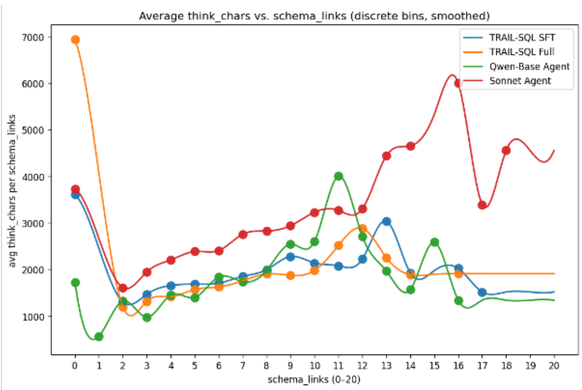

Figure 7: Reasoning efficiency v.s schema links

Cold-start ablation Additionally, we examine the efficacy of initialization strategies in Table 7. The results demonstrate that RL with SFT cold-start yields the highest performance, surpassing both the RL model trained without cold-start and the SFT-only baseline.

Reasoning scaling with schema complexity Figure 7 characterizes the relationship between reasoning length (average characters within <think> tags) and the number of schema links in the predicted SQL. Generally, a higher density of schema links necessitates increased reasoning volume. However, we observe distinct resource allocation patterns across models: Sonnet tends to generate excessive reasoning chains, whereas Qwen-base often exhibits insufficient reasoning depth. In contrast,

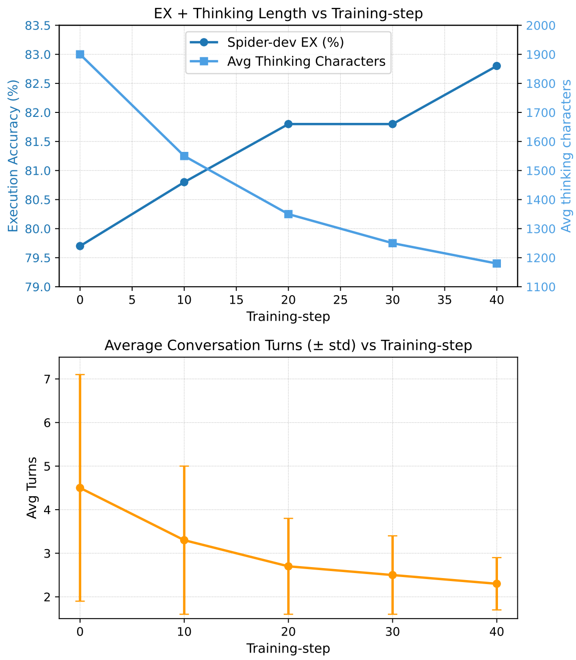

Figure 8: Changes in reasoning efficiency during training, shown alongside the curves for execution accuracy and the number of conversation turns.

SQL-TRAIL demonstrates an adaptive allocation, scaling its reasoning effort effectively to resolve schema dependencies without incurring unnecessary computational overhead.

Evolution of reasoning efficiency We further analyze the training dynamics by monitoring execution accuracy alongside generation costs in Figure 8. As training progresses, we observe that execution accuracy steadily improves, while both the reasoning character length and the total trajectory length decrease. This inverse correlation suggests that the model learns to optimize its thought process, pruning redundant reasoning steps to achieve correct solutions with greater computational efficiency.

Cost–accuracy trade-off We further analyze the cost–accuracy trade-off of different inference and training paradigms on BIRD-Dev, with results summarized in Table 8. For each method, we report the average token usage per query, including **Total** tokens (input + output), **Input** tokens (user prompts together with per-turn database feedback), and **Output** tokens (model generations), while computing Execution Accuracy (EX) using majority voting at inference time; all multi-turn methods are capped at 10 turns per question. As shown in Table 8, our multi-turn RL framework improves accuracy through effective exploration without incurring substantial token overhead: SQL-TRAIL-7B with the full RL objective achieves the best EX of 64.2% while using 3.2K total tokens on average, which is comparable to the 3.1K tokens of the single-pass RL baseline. In contrast, naïve multi-turn prompting increases both interaction length and token consumption while performing substantially worse. We also find that the turn-level reward improves efficiency: compared to the RL variant without turn reward, the full reward reduces the average number of turns from 3.2 to 2.3 and lowers total token usage from 3.6K to 3.2K, while simultaneously improving EX from 63.2% to 64.2%. Overall, these results suggest that our turn-aware multi-turn RL objective improves both final accuracy and interaction efficiency.

More recent baselines We further compare against more recent single-pass RL baselines in Table 9. Specifically, we include the 7B and 14B results for Arctic-Text2SQL-R1 (Yao et al., 2026), which are identical to those reported in the XYZ-Text2SQL-R1 paper; 3B results are omitted because they were not reported by the authors. We evaluate all methods using Execution Accuracy (EX) and Data Efficiency (DE). According to the paper, Arctic/XYZ-Text2SQL-R1 is trained on 27,785 samples filtered from BIRD-train, Spider-train, Spider-dev, and Gretel-Synth Filtered. Notably, BIRD-train and Spider-dev are in-distribution with respect to the BIRD-dev and Spider-test evaluation sets, respectively. In contrast, SQL-TRAIL is trained on only 1,873 examples drawn from Spider-train and SynSQL, which are fully out-of-distribution relative to these evaluation benchmarks. As shown in Table 9, although Arctic/XYZ-Text2SQL-R1 achieves higher absolute EX, SQL-TRAIL attains substantially higher data efficiency, particularly on BIRD-dev, demonstrating that our multi-turn RL framework extracts much more performance per training example under a significantly more challenging training setup.

Model	Paradigm	Setting	Avg. Turn(s)	Total (K)	Input (K)	Output (K)	EX (%)
Sonnet 3.7	Single-pass	Pure Inference	1.0	2.8	2.3	0.5	60.1
Sonnet 3.7	Multi-turn	Pure Inference	4.2	3.8	3.0	0.8	60.8
Qwen-2.5-Coder-7B	Single-pass	Pure Inference	1.0	2.5	2.3	0.2	61.3
Qwen-2.5-Coder-7B	Single-pass	RL	1.0	3.1	2.3	0.8	62.1
Qwen-2.5-Coder-7B	Multi-turn	Pure Inference	6.4	4.3	3.4	0.9	51.2
SQL-TRAIL-7B	Multi-turn	SFT	2.5	3.3	2.8	0.5	58.7
SQL-TRAIL-7B	Multi-turn	RL (w/o Turn Reward)	3.2	3.6	2.9	0.7	63.2
SQL-TRAIL-7B	Multi-turn	RL (Full Reward)	2.3	3.2	2.7	0.5	64.2

Table 8: Cost–accuracy trade-off on BIRD-Dev. We report average conversation turns, token usage, and execution accuracy (EX).

Method	EX: Spider-test (%)	EX: BIRD-dev (%)	DE: Spider-test	DE: BIRD-dev
Qwen2.5-Coder-7B	82.2	50.9	–	–
Arctic/XYZ-Text2SQL-R1-7B	88.8	68.9	0.24	0.65
SQL-TRAIL-7B	86.0	60.1	2.0	4.9
Qwen2.5-Coder-14B	86.6	61.5	–	–
Arctic/XYZ-Text2SQL-R1-14B	89.4	70.1	0.10	0.31
SQL-TRAIL-14B	86.8	63.6	0.11	1.12

Table 9: Comparison with recent baselines on Spider-test and BIRD-dev. We report Execution Accuracy (EX) and Data Efficiency (DE). Arctic/XYZ-Text2SQL-R1 is trained on 27,785 filtered samples, including in-distribution data from BIRD-train and Spider-dev, while SQL-TRAIL is trained on only 1,873 out-of-distribution samples from Spider-train and SynSQL.

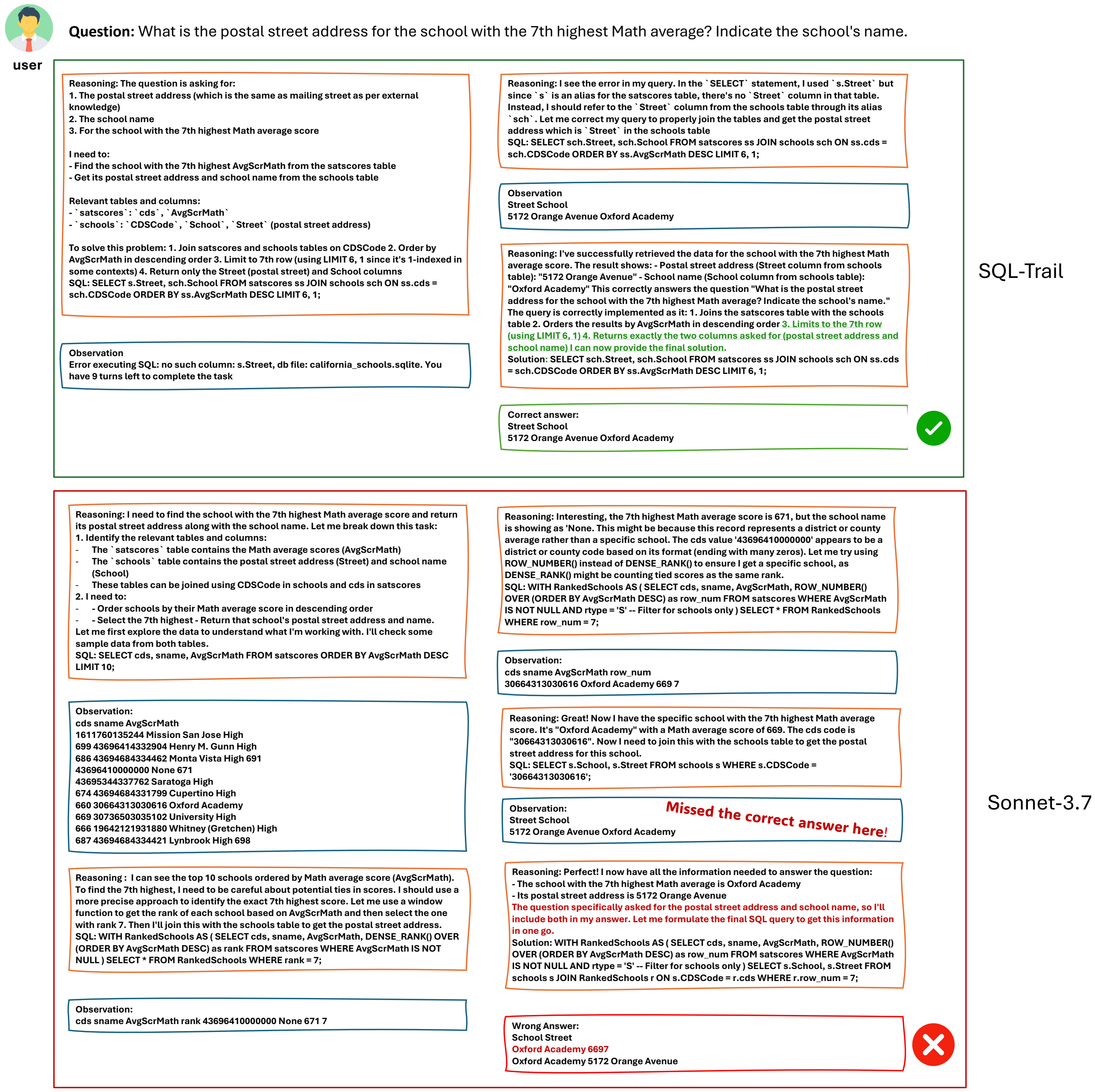

Figure 9: **SQL-TRAIL** and Sonnet-3.7 agent overthinking behavior comparison

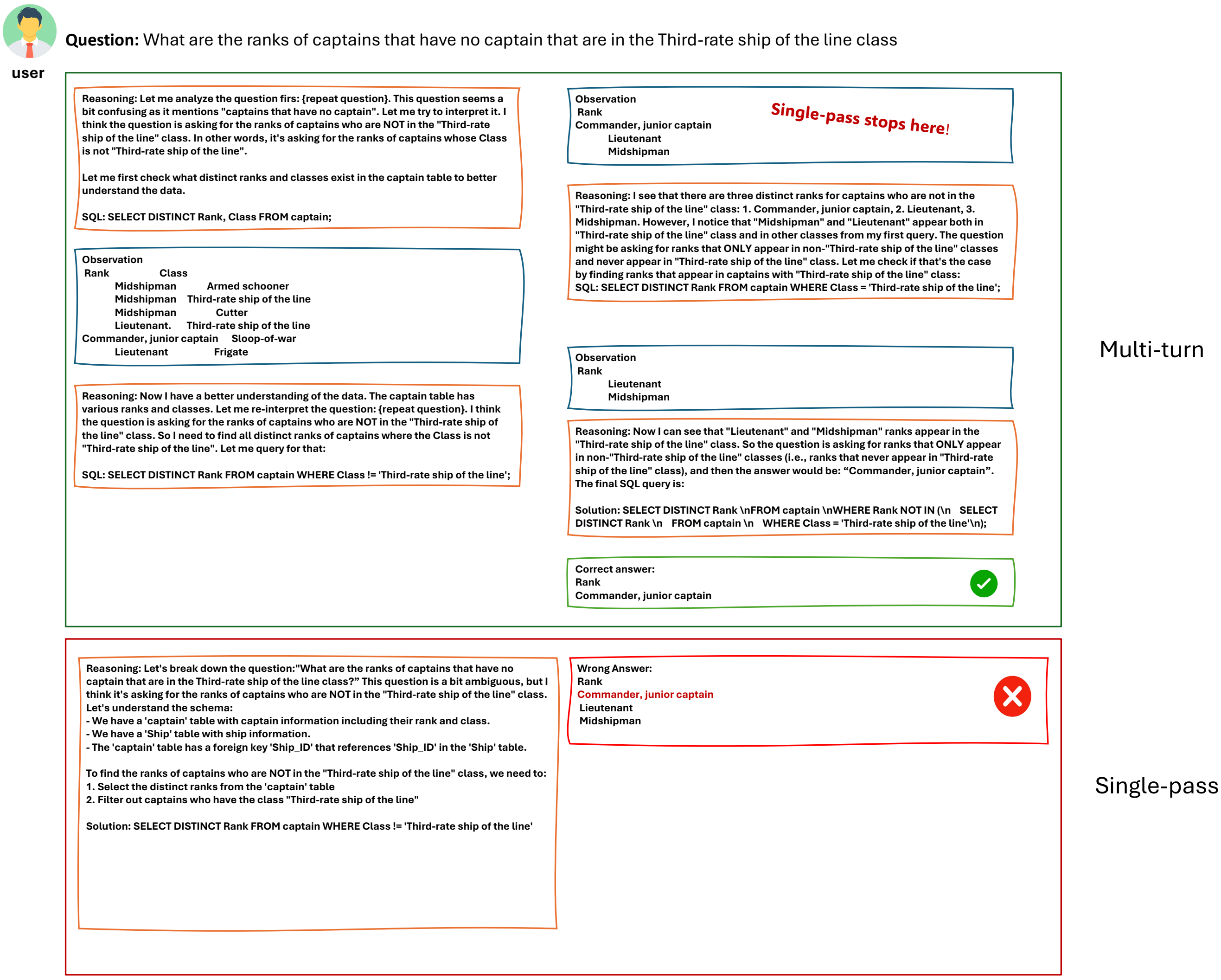

Figure 10: Example where multi-turn succeeds single-pass

Algorithm 1 LLM Response Rollout with Multi-Turn SQL Tool Calls

Require: Input question x, policy model π_θ, maximum turn budget T
Ensure: Final SQL query $q^\star$

Initialize dialogue history $h \leftarrow x$
Initialize turn counter $t \leftarrow 0$
while $t < T$ **do**
  Initialize current turn response $y_t \leftarrow \varnothing$
  while True **do**
    Generate next token $u \sim \pi_\theta(\cdot \mid x, h, y_t)$
    Append token $y_t \leftarrow y_t + u$
    if $u \in$ [</reasoning>, </sql>, </solution>, <eos>] **then**
      break
    end if
  end while
  Append assistant turn $h \leftarrow h + y_t$
  if <solution>detected in y_t **then**
    Extract final SQL $q^\star$ from <solution>block
    return $q^\star$
  else if <sql>detected in y_t **then**
    Parse SQL query $\hat{q} \leftarrow \text{PARSESQL}(y_t,$ <sql>$)$
    Execute query to obtain result $r \leftarrow \text{EXECSQL}(\hat{q})$
    if r is invalid **then**
      $o \leftarrow$ "Your previous action is invalid. Think and try again."
    else
      $o \leftarrow \text{TOSTRING}(\text{DATAFRAME}(r)$ with column headers$)$
    end if
    $z \leftarrow$ <observation> o+"You have $(T - t)$ turns left to complete the task."</observation>
    Append user observation $h \leftarrow h + z$
  end if
  $t \leftarrow t + 1$
end while
Generate final <solution>...</solution>block $y^{\text{sol}} \sim \pi_\theta(\cdot \mid x, h)$
Extract and return final SQL $q^\star$ from y^{sol}

System Prompt for Multi-turn Text-to-SQL Agent

Task Overview: You are a helpful text-to-SQL expert assistant. 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.

Database Engine: {engine}

Database Schema: {schema}

External Knowledge: {external_knowledge}

Question: {question}

Instructions:

1. 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.
2. The generated query should return all of the information asked in the question without any missing or extra information.
3. In your first response, you must identify all tables and columns from the database schema that are relevant to answering the question.
4. Before generating the final SQL query you should reflect through previous steps and refine any errors previous steps.

Format:

1. Conduct thinking inside <reasoning>...</reasoning> blocks every time you get new observation or information. nothing more.
2. You can use SQL tool written within a single <sql>your sql</sql> block to explore or verify. SQL tool output will be shown as dataframe inside <observation>...</observation>. Based on this observation, you can think again and refine.
3. The returned dataframe will be truncated in 50 rows if observation is too long.
4. If you find no further exploration is needed or reaches max turns, directly provide the final sql in <solution>...</solution>

```
------------------------ START OF EXAMPLE ------------------------
Question: what is the average price of all products?

Database Schema:
    Table: products
    - id (INTEGER, PRIMARY KEY)
    - name (TEXT)
    - price (REAL)

<think>I need to find the average price of all items listed in the products table.

Relavent tables and columns:
```

```
- 'products':'price'

To solve this problem I should issue a simple aggregate query using
AVG(price).</think>
<sql>SELECT AVG(price) FROM products;</sql>
<observation>
   AVG(price)
0       24.75
</observation>
<think>The result shows that the average product price is 24.75. The question asks
for the average price, so I can directly return this SQL query as the final
solution.</think>
<solution>SELECT AVG(price) FROM products;</solution>
------------------------ END OF EXAMPLE ------------------------
```