

# 000 001 002 003 004 005 006 007 MTIR-SQL: MULTI-TURN TOOL-INTEGRATED REA- 008 SONING REINFORCEMENT LEARNING FOR TEXT-TO- 009 SQL 010 011 012

013 **Anonymous authors**  
014 Paper under double-blind review  
015  
016  
017  
018  
019  
020  
021  
022  
023  
024  
025  
026  
027  
028  
029  
030

## ABSTRACT

031 As large language models (LLMs) are increasingly used in Text-to-SQL tasks,  
032 Reinforcement Learning (RL) has become a common method for improving per-  
033 formance. Existing methods primarily rely on static execution feedback, which  
034 restricts real-time error correction. However, integrating multi-turn tool invoca-  
035 tion along with dynamic feedback could significantly improve adaptability and  
036 robustness, ultimately enhancing model performance. To address these issues,  
037 we propose **MTIR-SQL**, an innovative Multi-turn Tool-Integrated Reasoning  
038 reinforcement learning framework for Text-to-SQL. Our approach introduces  
039 an execution-aware multi-turn reasoning paradigm that seamlessly incorporates  
040 database execution feedback at each reasoning step, enabling context-sensitive  
041 query generation and progressive refinement throughout the reasoning process.  
042 The framework extends the GRPO algorithm to accommodate complex multi-turn  
043 interaction scenarios. Considering the training instability characteristics of MTIR  
044 and the potential for significant Deviation of model distribution from the initial  
045 model, we enhance the GRPO algorithm by adding a trajectory filtering mech-  
046 anism and removing KL loss constraints. Experimental results demonstrate that  
047 MTIR-SQL, with 4B parameters, achieves 64.4% accuracy in the BIRD Dev and  
048 84.6% execution accuracy in the SPIDER Dev, significantly outperforming exist-  
049 ing approaches.

## 1 INTRODUCTION

050 Text-to-SQL, the task of automatically translating natural language questions into executable SQL  
051 queries, is a key technique for lowering the barrier to database access (Liu et al., 2025). By enabling  
052 non-technical users to query structured data in natural language, it has found wide applications in  
053 business intelligence, data analytics, and interactive question answering (Hong et al., 2025).

054 Existing approaches to Text-to-SQL generally fall into three paradigms: (i) supervised fine-tuning  
055 (SFT) of domain-specific open-source models (Li et al., 2025b; 2024b); (ii) prompting closed-source  
056 large language models (LLMs) with advanced reasoning strategies such as chain-of-thought (Li  
057 et al., 2025a; Zhai et al., 2025; Pourreza et al., 2024); and (iii) reinforcement learning (RL)-based  
058 methods that optimize model reasoning with algorithms such as PPO and GRPO (Pourreza et al.,  
059 2025; Ma et al., 2025; Yao et al., 2025; Dai et al., 2025). In particular, RL methods leverage final  
060 SQL execution results as reward signals for policy optimization. However, current practices treat  
061 execution feedback merely as scalar rewards, wasting rich tool information and leaving static LLMs  
062 unable to adapt their reasoning dynamically.

063 Recently, Multi-turn Tool-Integrated Reasoning (MTIR) has emerged as a promising paradigm for  
064 enhancing the reasoning capabilities of LLMs. By interleaving model reasoning with external  
065 tools—such as search engines, Python interpreters, and SQL executors—MTIR enables LLMs to  
066 overcome intrinsic limitations in computation, retrieval, and structured manipulation. Several studies  
067 have demonstrated the benefits of RL-based optimization in tool use: Search-R1 (Jin et al., 2025)  
068 explores dynamic tool invocation for question answering, while ToRL (Li et al., 2025c) and Effec-  
069 tive CIR (Bai et al., 2025) design tailored RL recipes for mathematical reasoning. More recently,

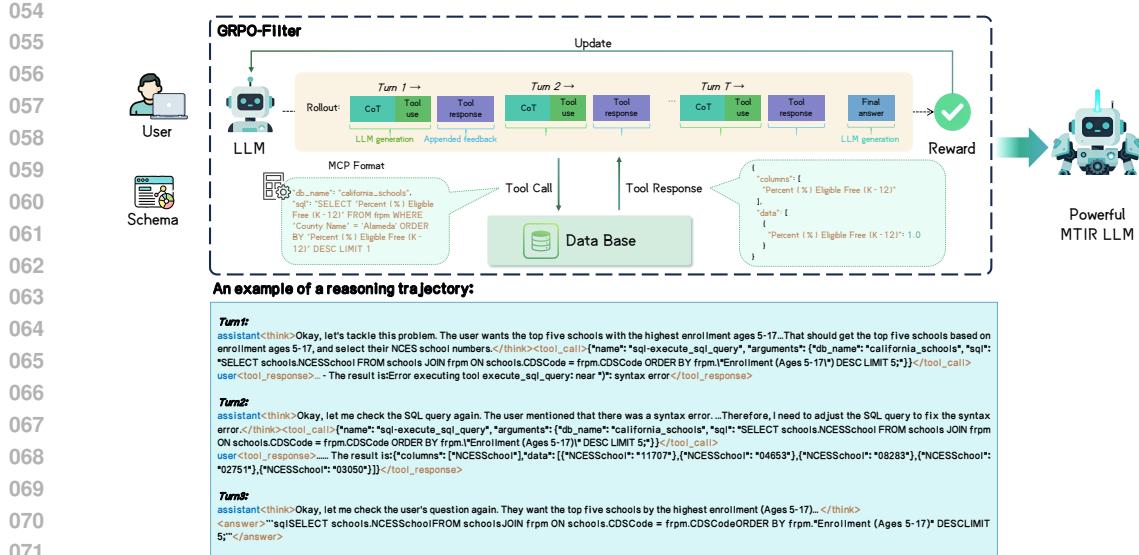


Figure 1: Overview of the MTIR-SQL framework. The framework integrates multi-turn reasoning with execution feedback and extends GRPO with trajectory filtering to enable dynamic correction and stable training, thereby enhancing SQL generation accuracy in complex scenarios.

the ReEx-SQL (Dai et al., 2025) framework extends TIR to Text-to-SQL, providing early evidence of its potential in structured query tasks.

Despite this progress, three fundamental challenges remain. On the **tool side**, SQL-oriented MTIR remains underexplored; existing efforts often rely on ad-hoc token mechanisms for tool invocation, limiting interoperability, extensibility, and compatibility with diverse database operations (Jin et al., 2025; Dai et al., 2025). On the **RL side**, dominant methods such as GRPO suffer from reward collapse and difficulty in modeling long-horizon dependencies, leading to instability in multi-turn tool interaction. On the **framework side**, current Text-to-SQL studies adopt heterogeneous, non-standardized implementations that lack modularity and generality (Dai et al., 2025; Ma et al., 2025; Yao et al., 2025; Gajjar et al., 2025).

To address these issues, we propose **MTIR-SQL**, a reinforcement learning framework for Multi-turn Tool-Integrated Reasoning in Text-to-SQL (Figure 1). MTIR-SQL extends GRPO to handle complex multi-turn interactions and introduces two key modifications: (i) a trajectory filtering mechanism to discard invalid rollouts and (ii) the removal of KL regularization to mitigate distributional collapse during training. Built on top of RL-Factory (Chai et al., 2025) with standardized MCP-compatible tool invocation, our framework ensures extensibility and interoperability.

Our contributions are summarized as follows:

- **MTIR-SQL Framework.** We introduce a novel RL framework for Text-to-SQL that enables LLMs to reason interactively and directly optimize via SQL execution feedback. It incorporates **tool response tokens masking** for stable training and supports multi-turn iterative reasoning and execution verification.
- **GRPO Extensions.** We extend GRPO with SQL execution rollout expansion and trajectory filtering to stabilize training in multi-turn tool-use scenarios, effectively mitigating reward collapse.
- **Strong Empirical Results.** On the BIRD dataset, MTIR-SQL trained on Qwen-3-4B achieves a 16% absolute improvement over baselines, matching the performance of recent 7B-coder models. It attains competitive execution accuracy, demonstrating its robustness and effectiveness.

108 **2 RELATED WORK**109 **2.1 RL FOR TOOL-INTEGRATED REASONING**

110 Tool-Integrated Reasoning (TIR) has emerged as a key paradigm for augmenting large language  
 111 models by enabling interaction with external tools and APIs (Zhang et al., 2025). Early work fo-  
 112 cused on single-turn tool invocation through supervised fine-tuning approaches, demonstrating ef-  
 113 fectiveness across domains, including mathematical reasoning, code generation (Mai et al., 2025),  
 114 and search integration (Jin et al., 2025). Multi-turn TIR enables iterative reasoning through se-  
 115 quential tool interactions, where models repeatedly generate tool calls, execute it, and refine based on  
 116 results (Mai et al., 2025; Shang et al., 2025; Wang et al., 2025; Zeng et al., 2025; Zhao et al., 2025).  
 117 However, training stability remains a critical challenge due to distribution drift from external tool  
 118 outputs and error accumulation across reasoning rounds, often leading to training instability and  
 119 entropy collapse. Recent work like SimpleTIR (Xue et al., 2025) addresses these issues by filtering  
 120 empty rounds in multi-turn reasoning, achieving state-of-the-art performance on mathematical tasks.  
 121 Despite these advances, optimizing Multi-turn TIR for complex tasks remains challenging (Lin &  
 122 Xu, 2025; Dong et al., 2025a;b; Yu et al., 2025). We apply recent MTIR advancements, includ-  
 123 ing filtering and handling invalid turns, to the Text-to-SQL domain, improving execution feedback  
 124 management, multi-table relationship handling, and ensuring SQL semantic correctness across iter-  
 125 ations.

126 **2.2 TEXT-TO-SQL**

127 Text-to-SQL aims to automatically convert natural language questions into executable SQL query  
 128 statements, enabling natural language interfaces for databases. The field has evolved through three  
 129 main paradigms: supervised fine-tuning methods that train specialized models on domain-specific  
 130 datasets (Li et al., 2024b; 2025b; Yang et al., 2024; Qin et al., 2025), using closed-source large mod-  
 131 els with prompt engineering and chain-of-thought reasoning to handle complex multi-table joins and  
 132 nested queries (Li et al., 2025a; Zhai et al., 2025; Pourreza et al., 2024; 2025; Lyu et al., 2025; Pour-  
 133 reza & Rafiei, 2023; Xie et al., 2024; Cao et al., 2024), and reinforcement learning approaches using  
 134 algorithms such as GRPO to enhance reasoning capabilities and generalization (Dai et al., 2025).  
 135 Despite recent advancements, current RL-based methods exhibit significant limitations. They rely  
 136 on static context during generation and lack mechanisms for validating or correcting interme-  
 137 diate reasoning steps, resulting in errors that cannot be self-corrected (Ma et al., 2025; Yao et al.,  
 138 2025; Gajjar et al., 2025). Execution feedback is treated as a reward signal rather than dynamically  
 139 integrated, hindering the model’s ability to adapt to complex scenarios. The challenge persists in  
 140 incorporating execution feedback while managing database results, multi-table relationships, and en-  
 141 suring SQL semantic correctness. To address these issues, we introduce Multi-turn Tool-Integrated  
 142 Reasoning in the Text-to-SQL domain, enabling the model to improve performance through iterative  
 143 use of external tools.

144 **3 METHODOLOGY**

145 We propose an SQL-integrated reinforcement learning framework with GRPO-Filter, which com-  
 146 bines unconstrained optimization, selective rollout filtering, and multi-turn reasoning to improve  
 147 decision-making. The model dynamically interacts with SQL execution, refining its output through  
 148 iterative feedback. A reward mechanism focused on format, execution, and result correctness guides  
 149 the generation of high-quality SQL queries.

150 **3.1 SQL-INTEGRATED RL WITH GRPO-FILTER**

151 We formulate the reinforcement learning framework with SQL execution tool  $\mathcal{E}$  as follows:

$$\max_{\pi_\theta} \mathbb{E}_{x \sim D, y \sim \pi_\theta(\cdot|x; \mathcal{E})} [r_\phi(x, y)], \quad (1)$$

152 where  $\pi_\theta$  is the LLM policy and  $r_\phi$  is the reward function. Unlike prior reinforcement learning  
 153 methods that primarily rely on the policy LLM  $\pi_\theta(\cdot|x)$  to generate rollout sequences, our framework  
 154 explicitly incorporates SQL execution-guided reasoning via  $\pi_\theta(\cdot|x; \mathcal{E})$ , which can be formulated as

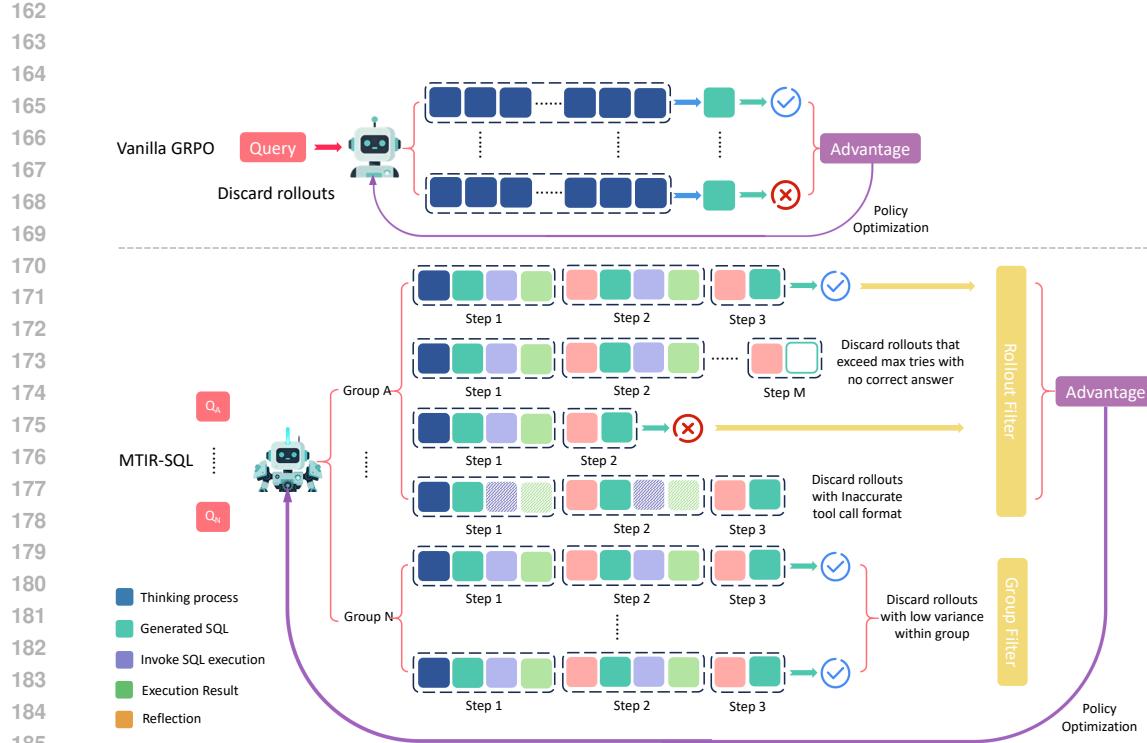


Figure 2: Compared to vanilla GRPO, our framework removes the KL constraint, introduces quality-aware rollout filtering, and extends to multi-turn reasoning with SQL execution feedback for more stable and accurate policy optimization.

$\pi_\theta(\cdot|x) \otimes \mathcal{E}$ , where  $\otimes$  denotes the interleaved SQL generation and execution feedback. This enables more effective decision-making in SQL generation tasks by leveraging real-time execution results to guide the model’s reasoning process.

Our approach introduces **GRPO-Filter**, an enhanced variant of Group Relative Policy Optimization (GRPO) specifically designed for complex multi-turn interactive scenarios. GRPO-Filter incorporates three key innovations:

**Unconstrained Optimization:** Unlike standard GRPO, we remove the KL divergence constraint between the policy and reference model, allowing for more flexible policy updates:

$$\mathcal{L}_{\text{GRPO-Filter}} = -\mathbb{E}_{(x,y) \sim \mathcal{D}} \left[ \frac{\pi_\theta(y|x; R)}{\pi_{\text{ref}}(y|x; R)} \cdot A(x, y) \right], \quad (2)$$

where  $A(x, y)$  represents the advantage function, eliminating the traditional KL penalty term  $\beta \cdot \text{KL}(\pi_\theta || \pi_{\text{ref}})$ .

**Selective Rollout Filtering:** To ensure the stability of gradient estimation and the efficiency of policy learning, we implement a multi-dimensional quality-aware filtering mechanism. Let  $G_x = \{y_1, y_2, \dots, y_K\}$  denote a group of  $K$  trajectories generated for input  $x$ . The set of retained trajectories is defined as:

$$\mathcal{T}_{\text{filtered}} = \{(x, y) \in \mathcal{T}_{\text{rollout}} : \mathcal{F}(x, y, G_x) > \tau\}, \quad (3)$$

where  $\tau$  serves as the binary acceptance threshold (set to 0 for strict boolean filtering). The filtering function  $\mathcal{F}(x, y, G_x)$  is a composite indicator designed to filter out noise and uninformative samples. It is formally defined as the conjunction of two criteria:

$$\mathcal{F}(x, y, G_x) = \mathbb{I}_{\text{valid}}(y) \cdot \mathbb{I}_{\text{div}}(G_x). \quad (4)$$

Specifically, these components address the following aspects:

- *Execution Validity* ( $\mathbb{I}_{\text{valid}}$ ): This term filters out invalid interaction patterns, specifically defined as trajectories where the model performs tool invocations for more than two turns without yielding a final answer, or encounters fatal execution errors. This ensures that the policy prioritizes efficient and conclusive reasoning paths.
- *Representation Diversity* ( $\mathbb{I}_{\text{div}}$ ): To prevent mode collapse and ensure meaningful advantage computation, we discard groups with insufficient variance. Specifically,  $\mathbb{I}_{\text{div}}(G_x) = 1$  if the standard deviation of rewards within group  $G_x$ , denoted as  $\sigma_R(G_x)$ , exceeds a minimum threshold  $\epsilon_\sigma$ .

**Multi-turn Extension:** GRPO-Filter extends the original framework to handle complex multi-turn interactions by maintaining conversation context and enabling iterative reasoning across multiple dialogue turns:

$$\pi_\theta(y_t|x, h_{<t}; R) = \pi_\theta(y_t|\text{concat}(x, h_{<t}); R), \quad (5)$$

where  $h_{<t}$  represents the conversation history up to turn  $t$ , and  $y_t$  is the response at turn  $t$ .

This multifaceted approach allows GRPO-Filter to effectively optimize policies for reasoning-intensive tasks while maintaining training stability and improving sample efficiency through selective learning from high-quality experiences.

### 3.2 INTERACTION WITH SQL EXECUTION ENVIRONMENT

The integration of SQL and its execution interface with large language models (LLMs), which are capable of comprehending and generating query intentions, can significantly enhance the automation of complex database operations. In an LLM-based SQL tool invocation environment, the system should exhibit human-like interactive and reasoning behaviors. These behaviors include generating syntactically correct and logically sound SQL queries from natural language questions, invoking database execution interfaces at appropriate moments, and executing queries safely. Additionally, the system should carefully interpret query results, verify their correctness, and refine subsequent problem decomposition or query generation strategies based on feedback. This capability is cultivated through guiding the model via multi-turn interaction and reflective learning with the SQL execution environment. Detailed prompt of sql Execution can be found in Appendix D.1.

With the support of SQL tools, the model dynamically incorporates database query results into the reasoning process through multi-turn execution, as illustrated in Figure 1. Unlike conventional methods that generate a complete SQL query until an end-of-sequence (EOS) token is produced, our approach constructs the full reasoning trajectory through continuous interaction with the SQL execution environment. The first interaction begins with a system prompt followed by the user’s question, with detailed content available in Appendix D.1. The model, acting as the assistant, generates an initial response until it outputs an EOS token. If no SQL tool call is detected, the process terminates. When an SQL query is identified, the environment service extracts and safely executes it, then appends the execution result to the dialogue context in the user role. The model subsequently continues its reasoning as the assistant based on the updated context, producing the next turn of response. This multi-turn process iterates until the model returns a final answer or a maximum number of turns, denoted as  $T$ , is reached. Detailed content can be found in Appendix B.

### 3.3 REWARD DESIGN

To optimize policy effectively, we introduce a streamlined reward mechanism that focuses on critical elements of SQL query quality. This framework incorporates three key factors—syntax validity, execution feasibility, and semantic precision—each providing distinct guidance to ensure the model generates SQL queries that are syntactically correct, executable, and semantically meaningful.

**Format Reward.** We guide the model to maintain a specific sequence of tags, ensuring a structured response. The response should follow a strict order: starting with `<think>...</think>`, followed optionally by `<tool_call>...</tool_response>`, and concluding with the `<answer>...</answer>` tag. Additionally, all tools must be used within `<tool_call>...</tool_call>` and `<tool_response>...</tool_response>` tags to maintain a standardized flow.

270

271

$$R_f = \begin{cases} 0.1, & \text{if the format is correct,} \\ -0.1, & \text{if the format is incorrect.} \end{cases} \quad (6)$$

272

274

**Execution Reward.** This reward evaluates the syntactic correctness and executability of the generated SQL. It prevents the model from producing invalid or overly complex queries. If the SQL statement fails to execute, the model will not receive subsequent rewards. Furthermore, the execution time is constrained to discourage the generation of unnecessarily complex queries:

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

## 4 EXPERIMENTS

294

295

### 4.1 EXPERIMENTAL SETUP

296

**Datasets.** We train and evaluate our model on two Text-to-SQL benchmarks, SPIDER (Yu et al., 2019) and BIRD (Li et al., 2023), which assess different aspects of the task. SPIDER is a large-scale, cross-domain benchmark focused on SQL complexity, with 10,181 questions and 5,693 unique queries across 200 databases. BIRD addresses real-world scenarios, featuring 12,751 question-SQL pairs on 95 large-scale databases with "dirty" data and evaluating both accuracy and efficiency. To ensure both training efficiency and SQL generation accuracy, this study follows the principles of "high quality, executable, and low redundancy" for data filtering and optimization. For the training of the BIRD and SPIDER benchmarks, we prioritize execution validity checks. Batch execution of reference SQL queries revealed that some samples returned empty results, which, if used for RL training, would fail to provide valid reward signals and could lead to learning biases or "reward hacking."

297

298

299

300

301

302

303

304

305

306

307

308

309

310

311

312

313

314

**Baselines.** We compare our MTIR-SQL framework against two primary categories of baseline methods. For supervised fine-tuning, we evaluate Qwen2.5-Coder-7B-Instruct (Hui et al., 2024), a state-of-the-art code generation model fine-tuned on Text-to-SQL datasets using standard cross-entropy loss. For reinforcement learning without tool integration, we implement GRPO on the Qwen3-4B model, using execution accuracy as the reward signal to optimize SQL generation through policy gradient methods. Both baselines use identical training procedures and computational budgets as our proposed framework but lack access to intermediate execution feedback during generation, allowing us to isolate the contribution of Multi-turn Tool-Integrated Reasoning.

315

316

317

318

319

320

321

322

323

**Experimental Details.** We conduct experiments using the Qwen3-Instruct model. During training and inference, we adopt database prompts from CodeS (Li et al., 2024b) and SQL-R1 (Ma et al., 2025), which provide curated schema components, values, and metadata, and have demonstrated competitive performance on the BIRD benchmark. We employ algorithms such as PPO and GRPO within the RL-Factory framework (Chai et al., 2025). The training configuration uses a batch size of 64 and a learning rate of 1e-6. During the rollout phase, we sample 5 outputs for each input at temperature  $T = 0.6$ , set the maximum sequence length to 8192, and the maximum number of interactions to  $N = 6$ . During inference, we apply greedy decoding ( $T = 0.0$ ). We use SQLite as the SQL executor to obtain execution feedback. The feedback includes column headers and cell values for up to 10 rows. All experiments are conducted on a system equipped with 8 NVIDIA A100 GPUs.

324 Table 1: Comprehensive comparison of BIRD Dev (EX%) scores. The “OSS” column indicates  
 325 whether the model is open-source (✓) or proprietary.  
 326

327 <b>Model</b>	328 <b>Size</b>	329 <b>OSS</b>	330 <b>BIRD Dev (EX%)</b>
<b>Models Under 10B Parameters</b>			
<i>Base Models</i>			
DPSK-Coder-6.7B-Instruct Guo et al. (2024)	6.7B	✓	43.1
Qwen3-4B	4B	✓	48.1
Qwen2.5-Coder-3B-Instruct Hui et al. (2024)	3B	✓	50.5
Qwen3-8B Ma et al. (2025)	8B	✓	50.8
Qwen2.5-Coder-7B-Instruct Hui et al. (2024)	7B	✓	50.9
OpenCoder-8B-Instruct Huang et al. (2025)	8B	✓	37.5
<i>SQL-Specific</i>			
SFT CodeS-7B Li et al. (2024b)	7B	✓	57.2
Think2SQL-3B Papicchio et al. (2025)	3B	✓	50.0
SQL-R1-3B Ma et al. (2025)	3B	✓	54.6
Think2SQL-7B Papicchio et al. (2025)	7B	✓	56.1
CogniSQL-R1-Zero-7B Gajjar et al. (2025)	7B		59.2
ReEx-SQL-7B Ma et al. (2025)	7B		64.9
SQL-R1-7B Ma et al. (2025)	7B	✓	66.6
Alpha-SQL+Qwen2.5-Coder-7B Gajjar et al. (2025)	7B	✓	66.8
Arctic-Text2SQL-R1-7B Yao et al. (2025)	7B	✓	68.9
<b>Models 10B–100B Parameters</b>			
<i>Base Models</i>			
Granite-20B-Code-Instruct Mishra et al. (2024)	20B	✓	34.0
Starcoder-15B-Instruct Lozhkov et al. (2024)	15B	✓	38.5
DPSK-Coder-V2-Instruct DeepSeek-AI et al. (2024)	16B	✓	44.6
Qwen3-14B Ma et al. (2025)	14B	✓	51.8
Codeniral-22B team (2024)	22B	✓	52.7
Qwen2.5-Coder-14B-Instruct Hui et al. (2024)	14B	✓	61.5
<i>SQL-Specific</i>			
SFT Code5-15B Li et al. (2024b)	15B	✓	58.5
Reasoning-SQL-14B Pourreza et al. (2025)	14B		64.2
SQL-R1-14B Ma et al. (2025)	14B	✓	67.1
Arctic-Text2SQL-R1-14B Yao et al. (2025)	14B	✓	70.1
Arctic-Text2SQL-R1-32B Yao et al. (2025)	14B	✓	70.5
<b>Large-scale Models (&gt; 100B or Proprietary)</b>			
<i>Base Models</i>			
Mistral Baseline Li et al. (2023)	123B	✓	53.5
DeepSeek-V3 DeepSeek-AI et al. (2025)	671B	✓	63.2
<i>SQL-Specific</i>			
SuperSQL (NLSQL-1360) Li et al. (2024a)	–		58.5
ChatGPT + CoT Li et al. (2023)	–		64.6
MCTS-SQL+GPT-4 Li et al. (2023)	–		69.4
OpenSearch-SQL+GPT-4oXie et al. (2025)	–		69.3
CHASE-SQL+Gemini 1.5 Pourreza et al. (2024)	–		73.1
<b>MTIR-SQL + Qwen3-4B (Ours)</b>	<b>4B</b>	✓	<b>64.4</b>
<b>MTIR-SQL + Qwen3-8B (Ours)</b>	<b>8B</b>	✓	<b>64.6</b>
<b>MTIR-SQL + Qwen3-14B (Ours)</b>	<b>14B</b>	✓	<b>68.1</b>

## 368 399 4.2 MAIN RESULT

370  
 371 **Performance on Main Benchmarks.** In Table 1, we present a comprehensive comparison of MTIR-  
 372 SQL against state-of-the-art baselines across varying parameter scales. In the compact model regime  
 373 (under 10B parameters), MTIR-SQL demonstrates exceptional parameter efficiency. Specifically,  
 374 our **MTIR-SQL (4B)** achieves an execution accuracy of **64.4%** on the BIRD Dev set. Remarkably,  
 375 despite having significantly fewer parameters, it outperforms robust open-source baselines such as  
 376 SFT CodeS-7B (57.2%) and Think2SQL-7B (56.1%), and matches the performance of proprietary  
 377 pipelines like ChatGPT + CoT (64.6%). While recent reinforcement learning-based models like  
 378 Arctic-Text2SQL-R1-7B achieve higher scores, our model offers a superior trade-off between com-

378 Table 2: Performance comparison of reasoning paradigms on benchmarks with pass@1.  
379

380 <b>Reasoning Paradigm</b>	381 <b>Training Type</b>	382 <b>BIRD Dev</b> 383 <b>EX (%)</b>	384 <b>SPIDER Dev</b> 385 <b>EX (%)</b>	386 <b>SPIDER Test</b> 387 <b>EX (%)</b>
388 Direct Output	389 –	390 46.9	391 69.2	392 70.8
393 Standard Reasoning	394 –	395 48.1	396 72.5	397 72.9
398 Tool-Integrated Reasoning	399 –	400 47.6	401 71.1	402 73.6
403 Standard Reasoning	404 GRPO	405 58.9	406 78.2	407 79.1
408 Multi-turn TIR	409 PPO	410 58.2	411 77.2	412 79.2
413 Multi-turn TIR	414 GRPO	415 60.3	416 80.1	417 81.4
418 <b>Multi-turn TIR</b>	419 <b>GRPO-Filter</b>	420 <b>63.1</b>	421 <b>82.4</b>	422 <b>83.4</b>

390  
391  
392 **putational cost and performance, effectively bridging the gap between lightweight deployment and**  
393 **high-precision reasoning.**

394 To assess the scalability of our framework, we extended MTIR-SQL to larger scales with 8B and  
395 14B parameters. As shown in the 10B-30B parameter section of Table 1, the performance of our  
396 method consistently improves with model size. Most notably, **MTIR-SQL (14B)** achieves **68.1%**,  
397 surpassing competitive peers including SQL-R1-14B (67.1%) and Reasoning-SQL (64.2%). This  
398 result highlights the effectiveness of our training strategy in eliciting complex SQL generation ca-  
399 pabilities, allowing our model to outperform other advanced RL-based methods within the same  
400 parameter class.

401 Finally, we compare MTIR-SQL with large-scale proprietary models. It is worth noting that **MTIR-**  
402 **SQL (4B)** already surpasses the massive DeepSeek-V3 (63.2%), illustrating that specialized training  
403 can yield better domain-specific results than general-purpose giant models. Furthermore, our 14B  
404 model approaches the performance of sophisticated multi-agent systems such as OpenSearch-SQL  
405 + GPT-4o (69.3%) and CHASE-SQL+Gemini 1.5 (73.1%).

406 **Performance on Reasoning Paradigms.** In Table 2, we evaluate the performance of different  
407 reasoning paradigms on Text-to-SQL benchmarks, specifically focusing on Pass@1 performance  
408 across the SPIDER and BIRD datasets. The results highlight the effectiveness of multturn tool-  
409 integrated reasoning. Among the reasoning paradigms, multi-turn TIR with GRPO-Filter leads to  
410 the highest performance on both the BIRD and SPIDER benchmarks. The BIRD Dev score of  
411 63.1% represents a significant improvement over standard reasoning and tool-integrated reasoning  
412 paradigms, which score 48.1% and 47.6%, respectively. In SPIDER Dev and Test, multi-turn TIR  
413 with GRPO filter also excels, achieving 82.4% and 83.4%, respectively, marking a clear advantage  
414 over other paradigms.

415 This reinforces the importance of incorporating execution feedback through multi-turn reasoning for  
416 enhancing performance in real-world Text-to-SQL tasks, particularly when dealing with complex  
417 databases like SPIDER and BIRD.

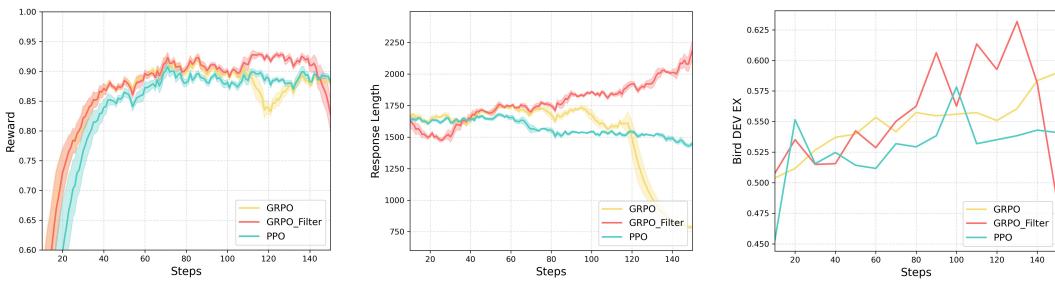
418 Table 3: Robustness Comparison on Spider-DK, Spider-Syn, Spider-Realistic, EHRSQ, and Sci-  
419 enceBenchmark.

420 <b>NL2SQL Method</b>	421 <b>Base Model</b>	422 <b>Spider-DK</b>	423 <b>Spider-Syn</b>	424 <b>Spider-Realistic</b>	425 <b>EHRSQ</b>	426 <b>Science</b> 427 <b>Benchmark</b>
<b>428 <i>Models Under 7B Parameters</i></b>						
429 SQL-R1 (Ma et al., 2025)	430 Qwen2.5-Coder-3B	431 70.5	432 66.4	433 71.5	434 -	435 -
<b>436 <i>Models Under 10B Parameters</i></b>						
437 OmniSQL (Li et al., 2025b)	438 Qwen2.5-Coder-7B	439 76.1	440 69.7	441 76.2	442 34.9	443 50.2
444 SQL-R1 (Ma et al., 2025)	445 Qwen2.5-Coder-7B	446 78.1	447 76.7	448 83.3	449 -	450 -
451 Arctic-Text2SQL-R1Yao et al. (2025)	452 OmniSQL-7B	453 81.5	454 -	455 -	456 36.7	457 51.8
458 SQL-o1 (Lyu et al., 2025)	459 Llama3-8B	460 78.7	461 72.6	462 82.7	463 -	464 -
<b>466 <i>Models Under 30B Parameters</i></b>						
467 OmniSQL (Li et al., 2025b)	468 Qwen2.5-Coder-14B	469 72.9	470 69.0	471 76.4	472 39.9	473 56.9
474 SQL-R1 (Ma et al., 2025)	475 OmniSQL-14B	476 79.3	477 78.5	478 86.2	479 -	480 -
481 Arctic-Text2SQL-R1Yao et al. (2025)	482 OmniSQL-14B	483 79.4	484 -	485 -	486 40.7	487 58.2
<b>488 <i>MTIR-SQL (Ours)</i></b>						
489 <b>MTIR-SQL (Ours)</b>	<b>Qwen3-4B</b>	<b>71.2</b>	<b>78.6</b>	<b>78.7</b>	<b>31.4</b>	<b>56.0</b>
490 <b>MTIR-SQL (Ours)</b>	<b>Qwen3-8B</b>	<b>72.9</b>	<b>77.2</b>	<b>77.4</b>	<b>34.4</b>	<b>57.0</b>
491 <b>MTIR-SQL (Ours)</b>	<b>Qwen3-14B</b>	<b>76.3</b>	<b>81.0</b>	<b>81.1</b>	<b>36.0</b>	<b>60.0</b>

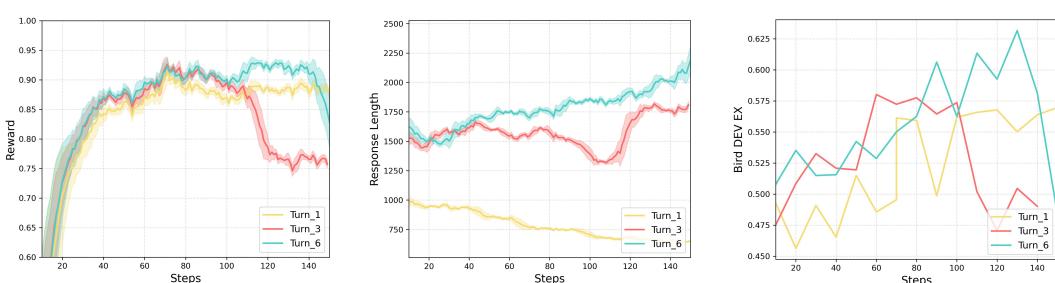
432  
 433 **Performance on Cross-domain Benchmark.** We evaluate the robustness and generalization ca-  
 434 *ability of our proposed MTIR-SQL across five challenging benchmarks, spanning perturbation-  
 435 *based datasets (Spider-DK, Spider-Syn, Spider-Realistic) and domain-specific tasks (EHRSQL,  
 436 *ScienceBenchmark), as summarized in Table 3. MTIR-SQL exhibits superior resilience to lin-  
 437 *guistic variations and domain shifts. Notably, on **Spider-Syn**, our approach achieves state-of-  
 438 *the-art performance, with the Qwen3-14B backbone reaching 81.0%, surpassing the competitive  
 439 **SQL-R1** (OmniSQL-14B) score of 78.5%. Furthermore, in the highly specialized **ScienceBench-  
 440 *mark***, MTIR-SQL establishes a new benchmark high of 60.0%, outperforming Arctic-Text2SQL-  
 441 *R1* (58.2%) and OmniSQL (56.9%). Even at smaller scales, our method demonstrates remarkable  
 442 *data efficiency; for instance, MTIR-SQL (Qwen3-4B) achieves 78.6% on Spider-Syn, significantly  
 443 *outperforming the similarly sized SQL-R1 (Qwen2.5-Coder-3B) by a margin of 12.2 points. These  
 444 *results validate that MTIR-SQL effectively mitigates performance degradation caused by synonym  
 445 *perturbation and cross-domain transfer.*********

### 445 4.3 ABLATION STUDY

446 **Ablation Study of RL Methods.** To assess the effectiveness of MTIR-SQL, we conducted com-  
 447 *parisons against PPO, GRPO, and our improved GRPO-Filter using the Qwen3-4B model. As il-  
 448 *lustrated in Figure 3 and summarized in Table 2, GRPO converges more rapidly than PPO due to  
 449 *the absence of a critic warm-up phase, but it often suffers from reward collapse in later training  
 450 *stages. PPO, in contrast, provides greater stability but at the cost of slower convergence. Crucially,  
 451 *GRPO-Filter addresses these limitations by selectively filtering low-quality rollouts and removing  
 452 *the KL constraint, thereby stabilizing multi-turn training while achieving substantial performance  
 453 *gains. This demonstrates that our modifications are not merely incremental but essential for enabling  
 454 *robust reinforcement learning in execution-aware Text-to-SQL tasks.********



466 Figure 3: Comparing the impact of different RL Methods on training and performance.



478 Figure 4: Comparing the impact of different max turns on training and performance.

479 **Ablation Study on Max Turns.** We further analyzed the impact of maximum tool calling turns by  
 480 *conducting experiments with settings of 1, 3, and 6. The training curves are shown in Figure 4, and  
 481 *the quantitative results are summarized in Table 2. The main findings are as follows: (1) Larger Max  
 482 *Turns generally lead to higher final rewards and stronger overall performance. More turns provide  
 483 *the model with additional opportunities to explore, optimize responses, and execute complex tasks;  
 484 *(2) However, excessive turns (such as 6) may also result in training instability, occasionally causing  
 485 *reward saturation or collapse phenomena; (3) Although Max Turns = 1 demonstrates faster conver-******

gence, due to severely limited interaction flexibility, there exists a gap between the final performance and optimal values.

Table 4: Ablation of Reward Components for MTIR-SQL on BIRD Dev.

Reward Components	BIRD Dev (EX %)
MTIR-SQL	63.1
w/o $R_{\text{format}}$	62.3 ↓ (0.8)
w/o $R_{\text{exec}}$	59.4 ↓ (3.9)
w/o $R_{\text{result}}$	58.8 ↓ (4.3)

**Ablation Study on Reward Design.** This ablation study evaluates the impact of removing each reward component on the model’s performance using the BIRD Development set:

- $R_f$  (**Format Reward**): We analyze the sensitivity of the model to the format reward weight, as visualized in Figure 5. Setting the format reward coefficient to a moderate value (0.1) improves performance from 62.3% to 63.1%, representing a 0.8% increase compared to the baseline (0). However, further increasing the weight to 0.2 leads to a significant performance drop to 60.5% (a 4.1% decline), indicating that excessive format rewards can negatively impact the model by over-constraining generation. A slight recovery is observed at 0.3 (61.2%), reinforcing the conclusion that a balanced format reward is beneficial but should not be overemphasized.
- $R_e$  (**Execution Reward**): Removing the execution reward results in the largest performance drop, from 63.1% to 59.4% (a 3.9% decrease), highlighting its crucial role in the natural language-to-SQL conversion process. Without execution-based feedback, the model struggles to make accurate predictions.
- $R_r$  (**Result Reward**): Excluding the result reward leads to a smaller decline in performance, from 63.1% to 58.8% (a 4.3% drop), underlining its importance in ensuring the functional correctness of the model’s SQL queries.

In conclusion, removing any reward—particularly  $R_f$ —significantly hampers the model’s performance. This underscores the necessity of a balanced reward system that integrates execution feedback, exploration, and result accuracy for optimal performance.

## 5 CONCLUSION

We propose **MTIR-SQL**, a novel reinforcement learning framework for complex multi-turn SQL generation tasks. MTIR-SQL’s central innovation resides in its feedback-driven reasoning approach, where execution results inform subsequent reasoning iterations, creating a self-correcting mechanism that substantially improves generation stability and query accuracy. We extend GRPO with **trajectory filtering** to mitigate distribution drift and remove KL divergence constraints to enhance learning efficiency. Experimental results demonstrate MTIR-SQL’s effectiveness: achieving **64.4%** accuracy on BIRD-SQL and **84.6%** execution accuracy on SPIDER with a 4B-parameter model, significantly outperforming baseline methods and advancing state-of-the-art in Text-to-SQL generation.

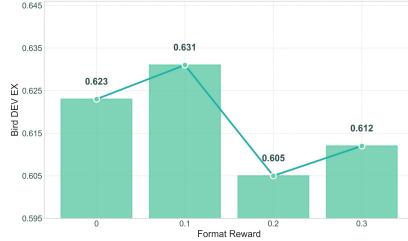


Figure 5: Ablation of Reward Components for MTIR-SQL on BIRD Dev Reward.

540 **6 ETHICS STATEMENT**  
541542 This study uses publicly available datasets (BIRD and SPIDER) and does not involve private or  
543 confidential data. No human participants are included, and we ensure fairness and transparency in  
544 our model's design and deployment.  
545546 **7 REPRODUCIBILITY STATEMENT**  
547548 The model code, datasets, and experimental setup are available upon request. Detailed instructions  
549 for reproducing our experiments are provided to ensure transparency and facilitate further research  
550 in the Text-to-SQL domain.  
551552 **REFERENCES**  
553554 Fei Bai, Yingqian Min, Beichen Zhang, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, Zheng Liu,  
555 Zhongyuan Wang, and Ji-Rong Wen. Towards effective code-integrated reasoning, 2025. URL  
556 <https://arxiv.org/abs/2505.24480>.  
557558 Zhenbiao Cao, Yuanlei Zheng, Zhihao Fan, Xiaojin Zhang, Wei Chen, and Xiang Bai. RSL-SQL:  
559 Robust Schema Linking in Text-to-SQL Generation, 2024. URL <https://arxiv.org/abs/2411.00073>.  
560561 Jiajun Chai, Guojun Yin, Zekun Xu, Chuhuai Yue, Yi Jia, Siyu Xia, Xiaohan Wang, Jiwen Jiang,  
562 Xiaoguang Li, Chengqi Dong, Hang He, and Wei Lin. Rlfactory: A plug-and-play reinforcement  
563 learning post-training framework for llm multi-turn tool-use, 2025. URL <https://arxiv.org/abs/2509.06980>.  
564565 Yaxun Dai, Wenxuan Xie, Xialie Zhuang, Tianyu Yang, Yiyi Yang, Haiqin Yang, Yuhang Zhao,  
566 Pingfu Chao, and Wenhao Jiang. Reex-sql: Reasoning with execution-aware reinforcement learning  
567 for text-to-sql, 2025. URL <https://arxiv.org/abs/2505.12768>.  
568569 DeepSeek-AI, Qihao Zhu, Daya Guo, Zhihong Shao, Dejian Yang, Peiyi Wang, Runxin Xu, Y. Wu,  
570 Yukun Li, and Huazuo Gao et al. Deepseek-coder-v2: Breaking the barrier of closed-source  
571 models in code intelligence, 2024. URL <https://arxiv.org/abs/2406.11931>.  
572573 DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, and  
574 Chenggang Zhao et al. Deepseek-v3 technical report, 2025. URL <https://arxiv.org/abs/2412.19437>.  
575576 Guanting Dong, Yifei Chen, Xiaoxi Li, Jiajie Jin, Hongjin Qian, Yutao Zhu, Hangyu Mao, Guorui  
577 Zhou, Zhicheng Dou, and Ji-Rong Wen. Tool-star: Empowering llm-brained multi-tool reasoner  
578 via reinforcement learning, 2025a. URL <https://arxiv.org/abs/2505.16410>.  
579580 Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia  
581 Chen, Jiazen Du, Huiyang Wang, Fuzheng Zhang, Guorui Zhou, Yutao Zhu, Ji-Rong Wen, and  
582 Zhicheng Dou. Agentic reinforced policy optimization, 2025b. URL <https://arxiv.org/abs/2507.19849>.  
583584 Kushal Gajjar, Harshit Sikchi, Arpit Singh Gautam, Marc Hammons, and Saurabh Jha. Cognisql-  
585 r1-zero: Lightweight reinforced reasoning for efficient sql generation, 2025. URL <https://arxiv.org/abs/2507.06013>.  
586587 Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao  
588 Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. Deepseek-coder: When the  
589 large language model meets programming – the rise of code intelligence, 2024. URL <https://arxiv.org/abs/2401.14196>.  
590591 Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao  
592 Huang. Next-generation database interfaces: A survey of llm-based text-to-sql, 2025. URL  
593 <https://arxiv.org/abs/2406.08426>.  
594

594 Siming Huang, Tianhao Cheng, J. K. Liu, Jiaran Hao, Liuyihan Song, Yang Xu, J. Yang, Jiaheng Liu,  
 595 Chenchen Zhang, Linzheng Chai, Ruijing Yuan, Zhaoxiang Zhang, Jie Fu, Qian Liu, Ge Zhang,  
 596 Zili Wang, Yuan Qi, Yinghui Xu, and Wei Chu. Opencoder: The open cookbook for top-tier code  
 597 large language models, 2025. URL <https://arxiv.org/abs/2411.04905>.

598 Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun  
 599 Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men,  
 600 Fei Huang, Bo Zheng, Yibo Miao, Shanghaioran Quan, Yunlong Feng, Xingzhang Ren, Xu-  
 601 ancheng Ren, Jingren Zhou, and Junyang Lin. Qwen2.5-coder technical report, 2024. URL  
 602 <https://arxiv.org/abs/2409.12186>.

603 Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and  
 604 Jiawei Han. Search-rl: Training llms to reason and leverage search engines with reinforcement  
 605 learning, 2025. URL <https://arxiv.org/abs/2503.09516>.

606 Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. The dawn of natural language  
 607 to sql: Are we fully ready? *Proceedings of the VLDB Endowment*, 17(11):3318–3331, July  
 608 2024a. ISSN 2150-8097. doi: 10.14778/3681954.3682003. URL <http://dx.doi.org/10.14778/3681954.3682003>.

609 Boyan Li, Jiayi Zhang, Ju Fan, Yanwei Xu, Chong Chen, Nan Tang, and Yuyu Luo. Alpha-sql:  
 610 Zero-shot text-to-sql using monte carlo tree search, 2025a. URL <https://arxiv.org/abs/2502.17248>.

611 Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan  
 612 Pan, Cuiping Li, and Hong Chen. Codes: Towards building open-source language models for  
 613 text-to-sql, 2024b. URL <https://arxiv.org/abs/2402.16347>.

614 Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang,  
 615 Tieying Zhang, Jianjun Chen, Rui Shi, Hong Chen, and Cuiping Li. Omnisql: Synthesizing high-  
 616 quality text-to-sql data at scale, 2025b. URL <https://arxiv.org/abs/2503.02240>.

617 Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin,  
 618 Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C.  
 619 Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can llm already serve as a database  
 620 interface? a big bench for large-scale database grounded text-to-sqls, 2023. URL <https://arxiv.org/abs/2305.03111>.

621 Xuefeng Li, Haoyang Zou, and Pengfei Liu. Torl: Scaling tool-integrated rl, 2025c. URL <https://arxiv.org/abs/2503.23383>.

622 Heng Lin and Zhongwen Xu. Understanding tool-integrated reasoning, 2025. URL <https://arxiv.org/abs/2508.19201>.

623 Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li,  
 624 Nan Tang, and Yuyu Luo. A survey of text-to-sql in the era of llms: Where are we, and where are  
 625 we going?, 2025. URL <https://arxiv.org/abs/2408.05109>.

626 Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane  
 627 Tazi, and Ao Tang et al. Starcoder 2 and the stack v2: The next generation, 2024. URL <https://arxiv.org/abs/2402.19173>.

628 Shuai Lyu, Haoran Luo, Ripeng Li, Zhonghong Ou, Jiangfeng Sun, Yang Qin, Xiaoran Shang,  
 629 Meina Song, and Yifan Zhu. Sql-01: A self-reward heuristic dynamic search method for text-to-  
 630 sql, 2025. URL <https://arxiv.org/abs/2502.11741>.

631 Peixian Ma, Xialie Zhuang, Chengjin Xu, Xuhui Jiang, Ran Chen, and Jian Guo. Sql-r1: Training  
 632 natural language to sql reasoning model by reinforcement learning, 2025. URL <https://arxiv.org/abs/2504.08600>.

633 Xinji Mai, Haotian Xu, Zhong-Zhi Li, Xing W, Weinong Wang, Jian Hu, Yingying Zhang, and Wen-  
 634 qiang Zhang. Agent rl scaling law: Agent rl with spontaneous code execution for mathematical  
 635 problem solving, 2025. URL <https://arxiv.org/abs/2505.07773>.

648 Mayank Mishra, Matt Stallone, Gaoyuan Zhang, Yikang Shen, Aditya Prasad, Adriana Meza Soria,  
 649 and Michele Merler et al. Granite code models: A family of open foundation models for code  
 650 intelligence, 2024. URL <https://arxiv.org/abs/2405.04324>.

651

652 Simone Papicchio, Simone Rossi, Luca Cagliero, and Paolo Papotti. Think2sql: Reinforce llm  
 653 reasoning capabilities for text2sql, 2025. URL <https://arxiv.org/abs/2504.15077>.

654

655 Mohammadreza Pourreza and Davood Rafiei. Din-sql: Decomposed in-context learning of text-to-  
 656 sql with self-correction, 2023. URL <https://arxiv.org/abs/2304.11015>.

657

658 Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tar-  
 659 lok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O. Arik. Chase-sql: Multi-path  
 660 reasoning and preference optimized candidate selection in text-to-sql, 2024. URL <https://arxiv.org/abs/2410.01943>.

661

662 Mohammadreza Pourreza, Shayan Talaei, Ruoxi Sun, Xingchen Wan, Hailong Li, Azalia Mirho-  
 663 seini, Amin Saberi, and Sercan "O. Arik. Reasoning-sql: Reinforcement learning with sql tailored  
 664 partial rewards for reasoning-enhanced text-to-sql, 2025. URL <https://arxiv.org/abs/2503.23157>.

665

666 Yang Qin, Chao Chen, Zhihang Fu, Ze Chen, Dezhong Peng, Peng Hu, and Jieping Ye. Route:  
 667 Robust multitask tuning and collaboration for text-to-sql, 2025. URL <https://arxiv.org/abs/2412.10138>.

668

669 Ning Shang, Yifei Liu, Yi Zhu, Li Lyyna Zhang, Weijiang Xu, Xinyu Guan, Buze Zhang, Bingcheng  
 670 Dong, Xudong Zhou, Bowen Zhang, Ying Xin, Ziming Miao, Scarlett Li, Fan Yang, and Mao  
 671 Yang. rstar2-agent: Agentic reasoning technical report, 2025. URL <https://arxiv.org/abs/2508.20722>.

672

673 Mistral AI team. Codestral — mistral ai, May 2024. URL <https://mistral.ai/news/codestratal>.

674

675 Renxi Wang, Rifo Ahmad Genadi, Bilal El Bouardi, Yongxin Wang, Fajri Koto, Zhengzhong Liu,  
 676 Timothy Baldwin, and Haonan Li. Agentfly: Extensible and scalable reinforcement learning for  
 677 lm agents, 2025. URL <https://arxiv.org/abs/2507.14897>.

678

679 Wenzuan Xie, Gaochen Wu, and Bowen Zhou. Mag-sql: Multi-agent generative approach with soft  
 680 schema linking and iterative sub-sql refinement for text-to-sql, 2024. URL <https://arxiv.org/abs/2408.07930>.

681

682 Xiangjin Xie, Guangwei Xu, Lingyan Zhao, and Ruijie Guo. Opensearch-sql: Enhancing text-to-sql  
 683 with dynamic few-shot and consistency alignment, 2025. URL <https://arxiv.org/abs/2502.14913>.

684

685 Zhenghai Xue, Longtao Zheng, Qian Liu, Yingru Li, Xiaosen Zheng, Zejun Ma, and Bo An. Sim-  
 686 pletir: End-to-end reinforcement learning for multi-turn tool-integrated reasoning, 2025. URL  
 687 <https://arxiv.org/abs/2509.02479>.

688

689 Jiaxi Yang, Binyuan Hui, Min Yang, Jian Yang, Junyang Lin, and Chang Zhou. Synthesizing text-to-  
 690 sql data from weak and strong llms, 2024. URL <https://arxiv.org/abs/2408.03256>.

691

692 Zhewei Yao, Guoheng Sun, Lukasz Borchmann, Zheyu Shen, Minghang Deng, Bohan Zhai, Hao  
 693 Zhang, Ang Li, and Yuxiong He. Arctic-text2sql-r1: Simple rewards, strong reasoning in text-to-  
 694 sql, 2025. URL <https://arxiv.org/abs/2505.20315>.

695

696 Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai,  
 697 Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guang-  
 698 ming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu,  
 699 Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao  
 700 Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingx-  
 701 uan Wang. Dapo: An open-source llm reinforcement learning system at scale, 2025. URL  
<https://arxiv.org/abs/2503.14476>.

702 Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li,  
703 Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-  
704 labeled dataset for complex and cross-domain semantic parsing and text-to-sql task, 2019. URL  
705 <https://arxiv.org/abs/1809.08887>.

706 Siliang Zeng, Quan Wei, William Brown, Oana Frunza, Yuriy Nevmyvaka, and Mingyi Hong.  
707 Reinforcing multi-turn reasoning in llm agents via turn-level credit assignment, 2025. URL  
708 <https://arxiv.org/abs/2505.11821>.

709 Bohan Zhai, Canwen Xu, Yuxiong He, and Zhewei Yao. Excot: Optimizing reasoning for text-to-sql  
710 with execution feedback, 2025. URL <https://arxiv.org/abs/2503.19988>.

711 Guibin Zhang, Hejia Geng, Xiaohang Yu, Zhenfei Yin, Zaibin Zhang, Zelin Tan, Heng Zhou,  
712 Zhongzhi Li, Xiangyuan Xue, Yijiang Li, Yifan Zhou, Yang Chen, Chen Zhang, Yutao Fan, Zihu  
713 Wang, Songtao Huang, Yue Liao, Hongru Wang, Mengyue Yang, Heng Ji, Michael Littman, Jun  
714 Wang, Shuic Cheng Yan, Philip Torr, and Lei Bai. The landscape of agentic reinforcement learning  
715 for llms: A survey, 2025. URL <https://arxiv.org/abs/2509.02547>.

716 Weikang Zhao, Xili Wang, Chengdi Ma, Lingbin Kong, Zhaohua Yang, Mingxiang Tuo, Xiaowei  
717 Shi, Yitao Zhai, and Xunliang Cai. Mua-rl: Multi-turn user-interacting agent reinforcement learn-  
718 ing for agentic tool use, 2025. URL <https://arxiv.org/abs/2508.18669>.

719

720

721

722

723

724

725

726

727

728

729

730

731

732

733

734

735

736

737

738

739

740

741

742

743

744

745

746

747

748

749

750

751

752

753

754

755

756 **A USE OF LLMs**  
757758 In this work, we use Large Language Models (LLMs) for text refinement and grammar checking.  
759 LLMs help improve the clarity, coherence, and grammatical accuracy of the generated content,  
760 ensuring the final text meets academic standards. Their use is limited to enhancing written content,  
761 without influencing the research methodology or decision-making process.  
762763 **B LLM RESPONSE ROLLOUT WITH MULTI-TURN SQL EXCURSIONS CALLS**  
764765 The algorithm describes the response generation process of a generative model (e.g., LLM) based  
766 on multi-turn interactions. The core idea of the algorithm is to progressively generate a response  
767 sequence based on the user’s input and previous responses. In each generation step, the model  
768 evaluates the current output and interacts with external tools for validation (e.g., executing SQL  
769 queries). The results returned by the tool are then integrated into the generated response. The entire  
770 process is conducted within a maximum action budget to ensure that the final output meets the  
771 problem’s requirements and is validated for accuracy. After each round, the model adjusts its output  
772 based on the results, continuing until a complete response is achieved or the budget limit is reached.  
773774 **Algorithm 1** LLM Response Rollout with Multi-Turn SQL Execution Tool Calls775 **Require:** Input query  $x$ , policy model  $\pi_\theta$ , SQL execution tool  $\mathcal{T}$ , maximum action budget  $B$ .  
776 **Ensure:** Final response  $y$ .

```

777 1: Initialize rollout sequence  $y \leftarrow \emptyset$ 
778 2: Initialize action count  $b \leftarrow 0$ 
779 3: while  $b < B$  do
780 4:   Initialize current action LLM rollout sequence  $y_b \leftarrow \emptyset$ 
781 5:   while True do
782 6:     Generate response token  $y_t \sim \pi_\theta(\cdot | x, y, y_b)$ 
783 7:     Append  $y_t$  to rollout sequence  $y_b \leftarrow y_b + y_t$ 
784 8:     if  $y_t \in \langle \text{tool call}, \langle \text{tool response} \rangle, \langle \text{eos} \rangle$  then
785 9:       break
786 10:      end if
787 11:    end while
788 12:     $y \leftarrow y + y_b$ 
789 13:    if  $\langle \text{tool call} \rangle$  detected in  $y_b$  then
790 14:      Extract SQL query  $q \leftarrow \text{Parse}(y_b, \langle \text{tool call} \rangle, \langle \text{tool call} \rangle)$ 
791 15:      Retrieve SQL query results  $d \leftarrow \mathcal{T}(q)$ 
792 16:      Insert  $d$  into rollout  $y \leftarrow y + \langle \text{tool response} \rangle d \langle \text{tool response} \rangle$ 
793 17:    else if  $\langle \text{tool response} \rangle$  detected in  $y_b$  then
794 18:      return final generated response  $y$ 
795 19:    else
796 20:      Ask for rethink  $y \leftarrow y + \langle \text{My action is not correct. Let me rethink.} \rangle$ 
797 21:    end if
798 22:    Increment action count  $b \leftarrow b + 1$ 
799 23: end while
800 24: return final generated response  $y$ 

```

801  
802  
803  
804  
805  
806  
807  
808  
809

## C ADDITIONAL RESULTS AND ANALYSIS

### C.1 ANALYSIS AND COMPARISON OF BASE LLMs

As presented in Table 5, we conducted a comprehensive evaluation of different base LLMs to investigate the effectiveness of our proposed training framework. We selected strong code-specialized models (Qwen2.5-Coder series) and NL2SQL-specific models (OmniSQL series) as baselines, comparing them against the Qwen3 (Thinking Mode) series.

A critical observation from the baseline results is the **initial performance disparity**. As shown in the middle section of Table 5, the vanilla Qwen3 models in Thinking Mode exhibited relatively modest performance, significantly lagging behind the Qwen2.5-Coder and OmniSQL counterparts. For instance, on the challenging BIRD (Dev) benchmark, the Qwen3-4B base model achieved an accuracy of only 48.1%, which is notably lower than the 50.5% of Qwen2.5-Coder-3B and substantially behind the domain-specific OmniSQL-7B (61.5%). This suggests that the inherent NL2SQL capability of the Qwen3 (Thinking Mode) backbone is initially weak and suboptimal for direct deployment in complex reasoning scenarios.

However, the integration of our **MTIR-SQL** framework yields a transformative improvement. Despite the weak initialization, the models fine-tuned with MTIR-SQL demonstrated remarkable performance gains across all metrics. Specifically:

- On the Spider (Dev) set, MTIR-SQL enabled the Qwen3-4B model to surge from 72.3% to **82.4%**, a substantial absolute improvement of **10.1%**.
- Similarly, on the BIRD (Dev) benchmark, the 14B variant improved from a baseline of 51.8% to **67.2%**, achieving a massive gain of **15.4%**.

These results highlight a pivotal insight: while the base Qwen3 models do not possess state-of-the-art capabilities out-of-the-box, they exhibit exceptional *plasticity* and potential when guided by our proposed method. The significant delta between the base and fine-tuned results confirms that MTIR-SQL successfully activates the model’s latent reasoning abilities, allowing a weaker backbone to achieve competitive performance comparable to, or even exceeding, larger models trained with standard supervised fine-tuning. This underscores the efficacy of our training strategy in bridging the gap between weak initialization and high-performance execution.

Table 5: Comparison of different base LLMs on Spider and BIRD benchmark with Greedy Search. Note that “Thinking Mode” refers to the vanilla Qwen3 behavior without our specific training.

Base Model	Spider (Dev)	Spider (Test)	BIRD (Dev)
Qwen2.5-Coder-3B	77.0	77.2	50.5
Qwen2.5-Coder-7B	73.4	82.2	50.9
Qwen2.5-Coder-14B	78.1	86.6	61.5
Qwen2.5-Coder-32B	77.7	87.5	64.5
OmniSQL-7B	81.2	87.9	61.5
OmniSQL-14B	81.4	88.3	64.2
OmniSQL-32B	80.9	87.6	64.5
Qwen3-4B (Thinking Mode)	72.3	72.8	48.1
Qwen3-8B (Thinking Mode)	73.5	76.1	50.8
Qwen3-14B (Thinking Mode)	75.9	76.2	51.8
<b>MTIR-SQL + Qwen3-4B</b>	<b>82.4</b>	<b>83.4</b>	<b>63.1</b>
<b>MTIR-SQL + Qwen3-8B</b>	<b>83.6</b>	<b>84.2</b>	<b>63.6</b>
<b>MTIR-SQL + Qwen3-14B</b>	<b>86.7</b>	<b>87.2</b>	<b>67.2</b>

864 C.2 ANALYSIS AND COMPARISON OF EFFICIENCY  
865866 To evaluate the practical deployability of our proposed framework, we conduct a comprehensive  
867 analysis of inference efficiency on the BIRD-dev dataset. We focus on three key metrics: *Latency per*  
868 *Question*, *Total Tokens per Question* (including input prompt and output completion), and *Execution*  
869 *Accuracy*. Additionally, we investigate the *Tool Call Frequency* to understand the reasoning behavior  
870 of our model. The results are summarized in Table 6.871 **Trade-off between Accuracy and Overhead.** As illustrated in Table 6, existing methods often  
872 struggle to balance performance with computational cost. While **CHESS** achieves a respectable  
873 accuracy of 61.5%, it incurs a prohibitive computational penalty, requiring an average of 251.3  
874 seconds and over 320K tokens per query. This suggests that its multi-turn reasoning or agent-  
875 based retrieval mechanisms, though effective, are inefficient for real-time applications. Conversely,  
876 **Qwen2.5-Coder-7B** offers the lowest latency (0.3s) but lags significantly in semantic correctness  
877 (58.2%), indicating a limitation in handling complex schema linking without sufficient reasoning  
878 depth.879 **Superiority of MTIR-SQL.** Our proposed **MTIR-SQL** series demonstrates a superior efficiency-  
880 performance frontier. Notably, **MTIR-SQL-14B** establishes an accuracy of **67.2%**, outperforming  
881 the strong baseline **SQL-R1-7B** by a margin of 3.5%. More importantly, this performance gain does  
882 not come at the cost of efficiency. **MTIR-SQL-14B** consumes only **1.7K tokens** on average—the  
883 lowest among all compared methods—while maintaining a latency of 0.5s. This counter-intuitive  
884 result, where a larger model uses fewer tokens, indicates that the **MTIR** strategy effectively aligns  
885 the model to generate concise, precise SQL queries directly, reducing the need for verbose chain-of-  
886 thought reasoning or iterative self-correction.887 **Strategic Tool Utilization.** A distinctive feature of our framework is the capability for autonomous  
888 tool interaction. As shown in the "Tool Call / Question" column of Table 6, the **MTIR-SQL** models  
889 exhibit a highly efficient usage pattern, averaging between **1.31** and **1.42 tool calls** per question.  
890 This low frequency is particularly revealing: it suggests that the Reinforcement Learning refinement  
891 has taught the model to invoke external tools (e.g., for schema state verification or preliminary ex-  
892 ecution) *selectively* rather than indiscriminately. Unlike redundant agent loops that inflate latency,  
893 our model executes tools only when essential for resolving ambiguity. This "precision-first" behav-  
894 ior explains how **MTIR-SQL** achieves high accuracy (67.2%) with minimal latency overhead (0.5s),  
895 effectively validating that intelligent tool use can enhance performance without compromising de-  
896 ployment efficiency.897 Furthermore, even our smaller variants, **MTIR-SQL-4B** and **8B**, exhibit competitive accuracy  
898 (63.1% and 63.6%) with minimal latency overhead, proving that our training methodology is model-  
899 agnostic and highly scalable. Overall, **MTIR-SQL** provides the most viable solution for production  
900 environments where both high precision and low latency are critical.901 Table 6: Efficiency comparison of different NL2SQL methods on BIRD-dev dataset.  
902

NL2SQL Method	Candidate Selection	Latency (s) / Question	Total Tokens (K) Question	Tool Call Question	Accuracy (%)
Qwen2.5-Coder-7B	Greedy Search	0.3	2.5	-	58.2
XiYan-SQL-7B	Greedy Search	0.5	4.1	-	62.1
CHESS	Greedy Search	251.3	320.8	-	61.5
SQL-R1-7B	Greedy Search	0.4	3.1	-	63.7
<b>MTIR-SQL-4B (Ours)</b>	<b>Greedy Search</b>	<b>0.5</b>	<b>2.9</b>	<b>1.34</b>	<b>63.1</b>
<b>MTIR-SQL-8B (Ours)</b>	<b>Greedy Search</b>	<b>0.4</b>	<b>2.0</b>	<b>1.31</b>	<b>63.6</b>
<b>MTIR-SQL-14B (Ours)</b>	<b>Greedy Search</b>	<b>0.5</b>	<b>1.7</b>	<b>1.42</b>	<b>67.2</b>

918 **D PROMPT AND CASE STUDY**  
919920 Prompts used while training and several cases are presented as follows.  
921922 **D.1 SYSTEM PROMPT**  
923

System Prompt of Tool	
924	<b>##Tools</b>
925	You may call one or more functions to assist with the user query.
926	You are provided with function signatures within <tools></tools> XML tags:
927	
928	
929	
930	
931	<tools>
932	{"name": "sql-execute-sql-query", "description": "
933	Execute SQL query and return partial results
934	containing column names (maximum 10 records).
935	Args:db_name (str): The name of the database.
936	sql (str): The SQL query to execute.
937	Returns:Dict[str, Union[List[Dict], Dict, None]]: A
938	dictionary containing 'columns' and 'data' of the
939	query (maximum of 10 records).
940	Raises: TimeoutError: If the query execution exceeds
941	the timeout.
942	sqlite3.Error: If an error occurs during the query
943	execution.
944	",
945	"parameters": {
946	"type": "object",
947	"properties": {
948	"db_name": {"title": "Db Name", "type": "string"},
949	"sql": {"title": "Sql", "type": "string"}
950	},
951	"required": ["db_name", "sql"]
952	}
953	</tools>
954	For each function call, return a JSON object with function name and arguments within
955	<tool_call></tool_call> XML tags:
956	
957	<tool_call>
958	{"name": <function-name>, "arguments": <args-json-object>}
959	</tool_call>
960	
961	
962	
963	
964	
965	
966	
967	
968	
969	
970	
971	

972 D.2 USER PROMPT  
973974 **Prompts used while training and evaluating**  
975976 You are a helpful SQL expert assistant. You should first think about how to write the SQL  
977 query by analyzing the question, database schema, and external knowledge, then validate  
978 your SQL with the tool until it is correct. Finally, you provide the final SQL query in  
979 `<answer> </answer>`.980 **Task Configuration**981 **Database Engine:** SQLite982 **Database:** {db\_id}983 **Database Schema:** {schema}984 **User Question:** {external\_knowledge} + {question}?985 **Requirements**986 1. **Precision:** Make sure you only output the information that is asked in the question.  
987 If the question asks for a specific column, make sure to only include that column in  
988 the SELECT clause, nothing more.  
989 2. **Completeness:** The generated query should return all of the information asked in  
990 the question without any missing or extra information.  
991 3. **Correctness:** Before generating the final SQL query, please think through the steps  
992 of how to write the query. Validate your SQL through tool testing.993 **Output Format:**994 **Important:** Use EITHER thinking + tool calls OR thinking + final answer. Do not mix the  
995 structures.996 **Option A (when validation needed):**997 `<think> Your analysis... </think>`  
998 [Tool calls for validation]999 **Option B (final answer):**1000 `<think> Your final analysis... </think>`  
1001 `<answer>`  
1002 `'''sql`  
1003 `YOUR_SQL_QUERY`  
1004 `</answer>`1005  
1006  
1007  
1008  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1020  
1021  
1022  
1023  
1024  
1025

1026 D.3 CASE STUDY

1027

1028

1029

1030

1031

1032

## Case1 Single Turn

## USER QUESTION

Exclusively virtual refers to Virtual = 'F'. How many schools with an average score in Math greater than 400 in the SAT test are exclusively virtual?

## MODEL RESPONSE

&lt;think&gt;

Okay, let's tackle this problem. The user is asking for the number of schools that are exclusively virtual (Virtual = 'F') and have an average math score greater than 400 in the SAT test.... So the query should be correct.

&lt;/think&gt;

&lt;tool\_call&gt;

1043

```
{
  "name": "sql-execute_sql_query",
  "arguments": {
    "db_name": "california_schools",
    "sql": "SELECT COUNT(*) FROM satscores JOIN schools ON satscores
           .cds = schools.CDSCode WHERE schools.Virtual = 'F' AND
           satscores.AvgScrMath > 400;"}
```

1049

}

1050

}

&lt;/tool\_call&gt;

&lt;tool\_response&gt;

The result is:{

```
  "columns": [
    "COUNT(*)"
  ],
  "data": [
    {
      "COUNT(*)": 4
    }
  ]
}
```

1061

}

1062

&lt;/tool\_response&gt;

&lt;think&gt;

Okay, let me check the user's question again. They want the number of schools that are exclusively virtual (Virtual = 'F') and have an average math score greater than 400 in the SAT....The result from the tool is 4, so that's the answer.

&lt;/think&gt;

&lt;answer&gt;

“sql

```
SELECT COUNT(*) FROM satscores JOIN schools ON satscores.cds =
schools.CDSCode WHERE schools.Virtual = 'F' AND satscores.
```

AvgScrMath &gt; 400;

1073

“

&lt;/answer&gt;

1076

1077

1078

1079

1080  
1081  
1082 **USER QUESTION**  
1083 calcium refers to element = 'ca'; label = '+' mean molecules are carcinogenic; label = '-'  
1084 means molecules are non-carcinogenic; MAX(label).Among the molecules with element  
1085 Calcium, are they mostly carcinogenic or non carcinogenic?  
1086  
1087 **MODEL RESPONSE**  
1088  
1089 **Turn 1**  
1090 <think>  
1091 Alright, let's solve this problem. The user wants to know whether molecules containing cal-  
1092 cium element (element 'Ca') are mostly carcinogenic or not. Label '+' means carcinogenic,  
1093 '-' means non-carcinogenic. So, we need to find the maximum label value for molecules  
1094 containing calcium... So the final SQL query will be to select the maximum label from  
1095 molecules that contain calcium atoms.  
1096 </think>  
1097 <tool\_call>  
1098 {  
1099 "name": "sql-execute\_sql\_query",  
1100 "arguments": {  
1101 "db\_name": "toxicology",  
1102 "sql": "SELECT MAX(m.label) AS max\_label FROM molecule m JOIN  
1103 atom a ON m.molecule\_id = a.molecule\_id WHERE a.element = '  
1104 Ca';"  
1105 }  
1106 }  
1107 </tool\_call>  
1108 <tool\_response>  
1109 The result is:: {  
1110 "columns": [  
1111 "max\_label"  
1112 ],  
1113 "data": [  
1114 {  
1115 "max\_label": null  
1116 }  
1117 ]  
1118 }  
1119 </tool\_response>  
1120 **Turn 2**  
1121 <think>  
1122 Hmm, the SQL query result returns max\_label as null. This is unexpected. Let me check the  
1123 query again. Maybe the label in the molecule table is stored as a string, but perhaps the query  
1124 didn't correctly select the label... Let me try the corrected query.  
1125 </think>  
1126 <tool\_call>  
1127 {  
1128 "name": "sql-execute\_sql\_query",  
1129 "arguments": {  
1130 "db\_name": "toxicology",  
1131 "sql": "SELECT MAX(m.label) AS max\_label FROM molecule m JOIN  
1132 atom a ON m.molecule\_id = a.molecule\_id WHERE a.element = '  
1133 CA';"  
1134 }  
1135 }  
1136 </tool\_call>  
1137 <tool\_response>

```

1134
1135     The result is: {
1136         "columns": [
1137             "max_label"
1138         ],
1139         "data": [
1140             {
1141                 "max_label": null
1142             }
1143         ]
1144     }
1145     </tool_response>
1146     Turn 3
1147     <think>
1148     Wait, the result is still null. Let me think again. Maybe the label is stored as a string in the
1149     molecule table, but the query didn't select it correctly. Let me try the corrected query.
1150     </think>
1151     <tool_call>
1152     {
1153         "name": "sql-execute-sql-query",
1154         "arguments": {
1155             "db_name": "toxicology",
1156             "sql": "SELECT MAX(m.label) AS max_label FROM molecule m JOIN
1157                 atom a ON m.molecule_id = a.molecule_id WHERE a.element = 'ca';"
1158         }
1159     }
1160     </tool_call>
1161     <tool_response>
1162     The result is: {
1163         "columns": [
1164             "max_label"
1165         ],
1166         "data": [
1167             {
1168                 "max_label": "-"
1169             }
1170         ]
1171     }
1172     </tool_response>
1173     Turn 4
1174     <think>
1175     Okay, so after adjusting the query to use 'ca' as the element, the result is that the max_label
1176     is '-'. That means among the molecules with calcium atoms, the maximum label is '-', which
1177     indicates they are non-carcinogenic. Therefore, the answer should be that they are mostly
1178     non-carcinogenic.
1179     </think>
1180     <answer>
1181     ``sql
1182     SELECT MAX(m.label) AS max_label FROM molecule m JOIN atom a ON m.
1183     molecule_id = a.molecule_id WHERE a.element = 'ca';
1184     ``
1185     </answer>
1186
1187

```

1188  
 1189

**Case3 Multi-Turn**

1190 **USER QUESTION**

1191 What is the number of SAT test takers of the schools with the highest FRPM count for K-12  
 1192 students?  
 1193

1194 **MODEL RESPONSE**

1195

1196 **Turn 1**

1197 <think>

1198 Alright, let me solve this problem. The user is asking about the number of SAT test takers  
 1199 at schools with the highest FRPM count for K-12 students. First, I need to determine which  
 1200 tables are involved.... Let me verify using the tool.  
 1201

1202 </think>  
 1203 <tool\_call>

1204 {  
 1205 "name": "sql-execute\_sql\_query",  
 1206 "arguments": {  
 1207 "db\_name": "california\_schools",  
 1208 "sql": "SELECT SUM(s.NumTstTakr) FROM satscores s JOIN frpm f ON  
 1209 s.cds = f.CDSCode WHERE f.\"FRPM Count (K-12)\" = (SELECT  
 1210 MAX(\"FRPM Count (K-12)\") FROM frpm);"  
 1211 }  
 1212 }

1213 </tool\_call>  
 1214 <tool\_response>

1215 {  
 1216 "columns": [  
 1217 "SUM(s.NumTstTakr)"  
 1218 ],  
 1219 "data": [  
 1220 {  
 1221 "SUM(s.NumTstTakr)": null  
 1222 }  
 1223 ]  
 1224 }

1225 </tool\_response>  
 1226 **Turn 2**

1227 <think>

1228 Hmm, the SQL query returned a sum result of null. This is not right. Let me check the query  
 1229 again. Maybe there's an error in the column names or table names... Let me test again.  
 1230

1231 </think>  
 1232 <tool\_call>

1233 {  
 1234 "name": "sql-execute\_sql\_query",  
 1235 "arguments": {  
 1236 "db\_name": "california\_schools",  
 1237 "sql": "SELECT SUM(s.NumTstTakr) FROM satscores s JOIN frpm f ON  
 1238 s.cds = f.CDSCode WHERE f.\"FRPM Count (K-12)\" = (SELECT  
 1239 MAX(\"FRPM Count (K-12)\") FROM frpm);"  
 1240 }  
 1241 }

1242 </tool\_call>  
 1243 <tool\_response>

1244 {  
 1245 "columns": [  
 1246 "SUM(s.NumTstTakr)"  
 1247 ]

```

1242
1243     ],
1244     "data": [
1245         {
1246             "SUM(s.NumTstTakr)": null
1247         }
1248     ]
1249 }
1250 </tool_response>
1251 Turn 3
1252 <think>
1253 Hmm, the result is still null. This is strange. Maybe there's an issue with the query structure.
1254 I need to check if the column name in the frpm table is really named "FRPM Count (K-12)"
1255 or if there's a spelling error. Let me try again.
1256 </think>
1257 <tool_call>
1258 {
1259     "name": "sql-execute_sql_query",
1260     "arguments": {
1261         "db_name": "california_schools",
1262         "sql": "SELECT SUM(s.NumTstTakr) FROM satscores s JOIN frpm f ON
1263             s.cds = f.CDSCode WHERE f.\"FRPM Count (K-12)\" = (SELECT
1264             MAX(\"FRPM Count (K-12)\") FROM frpm);"
1265     }
1266 }
1267 </tool_call>
1268 <tool_response>
1269 {
1270     "columns": [
1271         "SUM(s.NumTstTakr)"
1272     ],
1273     "data": [
1274         {
1275             "SUM(s.NumTstTakr)": null
1276         }
1277     ]
1278 }
1279 </tool_response>
1280 Turn 4
1281 <think>
1282 Maybe I should first check what the maximum value of FRPM Count (K-12) in the frpm
1283 table is. Let me write a query to get that value.
1284 </think>
1285 <tool_call>
1286 {
1287     "name": "sql-execute_sql_query",
1288     "arguments": {
1289         "db_name": "california_schools",
1290         "sql": "SELECT SUM(s.NumTstTakr) FROM satscores s JOIN frpm f ON
1291             s.cds = f.CDSCode WHERE f.\"FRPM Count (K-12)\" = (SELECT
1292             MAX(\"FRPM Count (K-12)\") FROM frpm);"
1293     }
1294 }
1295 </tool_call>
1296 <tool_response>
1297 {
1298     "columns": [
1299         "SUM(s.NumTstTakr)"
1300     ]
1301 }

```

```

1296
1297     ],
1298     "data": [
1299         {
1300             "SUM(s.NumTstTakr)": null
1301         }
1302     ]
1303 }
1304
1305 </tool_response>
1306 <think>
1307 Maybe there's an issue with the column names or the tables not joining correctly. Let me
1308 think again. Maybe I should check if the satscores table has data for schools in the frpm
1309 table.... I think this should work. Let me use this as the final query.
1310 </think>
1311 <tool_call>
1312 {
1313     "name": "sql-execute_sql_query",
1314     "arguments": {
1315         "db_name": "california_schools",
1316         "sql": "SELECT SUM(s.NumTstTakr) FROM satscores s JOIN (SELECT
1317             CDSCode FROM frpm WHERE \"FRPM Count (K-12)\" = (SELECT MAX
1318             (\"FRPM Count (K-12)\") FROM frpm)) AS top_frp AS top_frp ON s.cds =
1319             top_frp.CDSCode;"
1320     }
1321 }
1322 </tool_call>
1323 <tool_response>
1324 {
1325     "columns": [
1326         "SUM(s.NumTstTakr)"
1327     ],
1328     "data": [
1329         {
1330             "SUM(s.NumTstTakr)": 217547
1331         }
1332     ]
1333 }
1334 </tool_response>
1335 Turn 5
1336 <think>
1337 The tool response shows a result of 217547, which seems to be a valid number.
1338 </think>
1339 <answer>
1340     “sql
1341
1342     SELECT SUM(s.NumTstTakr) FROM satscores s JOIN (SELECT CDSCode FROM
1343     frpm WHERE "FRPM_Count_(K-12)" = (SELECT MAX("FRPM_Count_(K-12)"
1344     ) FROM frpm)) AS top_frp AS top_frp ON s.cds = top_frp.CDSCode;
1345
1346     “
1347
1348 </answer>
1349

```