# SkyRL-SQL: Multi-turn SQL Data Agents via RL

**Shu Liu***
UC Berkeley
lshu@berkeley.edu

**Alan Zhu***
UC Berkeley
aczhu@berkeley.edu

**Sumanth Hegde**
Anyscale

**Shiyi Cao**
UC Berkeley

**Shuo Yuan**
UC Berkeley

**Samion Suwito**
UC Berkeley

**Tyler Griggs**
UC Berkeley

**Matei Zaharia**
UC Berkeley

**Joseph E. Gonzalez**
UC Berkeley

**Ion Stoica**
UC Berkeley

## Abstract

Recent advances in reinforcement learning (RL) have enabled large language models (LLMs) to act as interactive agents across domains like coding and web search. However, database interactions, particularly in the Text-to-SQL domain, remain underexplored. Existing work largely treats Text-to-SQL as a one-shot generation task, which struggles in real-world scenarios where natural language queries are often ambiguous and databases are messy. We introduce a simple and data-efficient multi-turn RL approach that enables LLMs to probe databases, refine queries, and verify results across successive interactions, mirroring human exploratory analysis. Using only about 600 training examples, our 7B-parameter model improves execution accuracy by up to 9.2% on Spider benchmarks, outperforming stronger baselines including o4-mini, GPT-4o, and open-source SFT models trained on orders of magnitude more data. Our results highlight the promise of multi-turn RL as a scalable, practical approach for building robust Text-to-SQL agents.

## 1 Introduction

Recent advances in RL have turned LLMs into **interactive agents** capable of reasoning, exploring, and executing actions over multiple interaction turns. This has led to impressive progress in domains like coding [Qwen Team, 2025], web search [Zheng et al., 2025], and kernel code generation [Baronio et al., 2025]. However, an important and impactful application that has been under-explored by LLM agents research is that of database interactions.

Databases are ubiquitous in industry and research for storing and managing data. To examine and analyze the data, users' natural language questions need to be converted into precise SQL queries, naturally leading to exciting opportunities for involving LLMs in both turning user questions into SQL and creating natural language responses from the query results [Biswal et al., 2024]. In this work, we focus on the former task, referred to as **Text2SQL**.

In real-world applications, natural language questions are often vague or incomplete [Papenmeier et al., 2020]. For example, users might say "latest review" without referencing an exact column or expect joins across tables without stating them explicitly. Real-world databases also tend to be "messy" [Chu et al., 2016], with missing values, typos in data entry, inconsistent formatting within

*Equal contribution.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Multi-Turn Interactions in Large Language Models.

columns, and under-specified descriptions. Human analysts typically resolve such ambiguity by **exploring the database step by step** — a practice known as exploratory data analysis (EDA).

In contrast, existing Text2SQL work focuses on one-shot SQL generation: the model is given the natural language question and database schema and asked to generate SQL in a single step [Zhong et al., 2017, Xu et al., 2017, Guo et al., 2019, Liu et al., 2023, Gao et al., 2023, Li et al., 2024a, Pourreza et al., 2025, Li et al., 2025]. As the model cannot engage in exploratory data analysis, such generations can often be error-prone in real-world settings. On the other hand, multi-turn Text2SQL interactions overcome the limitations of the one-shot settings, allowing the model to **probe the database, refine, and verify the SQL query** through database interactions just as human experts would (Figure 1). Query generation then becomes a multi-turn decision process, where models incrementally explore and self-correct through trial and error.

However, we find that current models are not optimized for the multi-turn Text2SQL task, and that targeted fine-tuning can lead to impressive improvements over the untrained baseline: our model gains 7.2% improvement compared to the untrained model. Further, we study different fine-tuning methods, finding that RL training with a simple reward function leads to faster and more data-efficient learning than SFT: achieving our improvements with only 0.03% of the training data used for SFT. Finally, we analyze the inference-time traces of our fine-tuned model for learned behaviors and common failure modes, providing insights for further research into multi-turn Text2SQL agents.

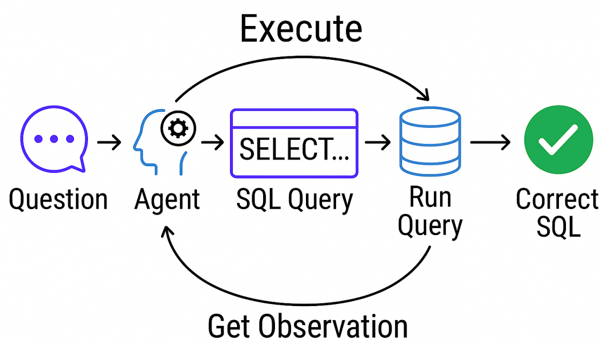


Figure 1: Workflow on Text2SQL tasks with database interactions

## 2 Related Works

**Text2SQL** Complex LLM-based pipelines are common for the Text2SQL task (e.g., demo/schema retrieval [Gao et al., 2023], task decomposition [Pourreza and Rafiei, 2023], multi-agent self-correction [Askari et al., 2024], etc.). More recently, the improved baseline performance of large foundation models has enabled the direct generation of SQL queries from natural language [Liu et al., 2023]. Further fine-tuning LLMs specifically for the Text2SQL task leads to even greater improvements [Li et al., 2024b, Zhai et al., 2025, Cohere Team, 2025]. However, many methods rely on supervised fine-tuning (SFT) for such fine-tuning. For example, OmniSQL [Li et al., 2025] performs SFT with a large million-scale synthetic dataset. In contrast, our work focuses on using reinforcement learning for training, efficiently (in terms of compute and data) producing a strong Text2SQL model.

**Reinforcement Learning** The current paradigm in RL for LLMs is RL with verifiable rewards (RLVR) [Lambert et al., 2024], where rewards are assigned via functions scoring correctness with clear-cut logic like string matching. Such reward signals are perfectly accurate compared to learned reward models, and have been shown to be effective in a variety of domains such as math, coding, web search, etc. [DeepSeek-AI, 2025]. The standard RLVR training algorithm used is GRPO [Shao et al., 2024], popular for its compute efficiency compared to the original PPO algorithm [Schulman et al., 2017]. Where PPO trains a critic model to estimate the difficulty of a prompt to calculate advantage, GRPO instead uses the average reward of multiple rollouts to estimate the difficulty, removing the cost of training a critic model. In this work, we make use of GRPO for these cost savings.

Recent work has extended RLVR to multi-turn settings, leveraging and enhancing the long-horizon reasoning capabilities of LLMs to discover, collect, and utilize large amounts of information [Zheng et al., 2025, Baronio et al., 2025]. RLVR has also been applied to the Text2SQL task [Ali et al., 2025, Yao et al., 2025, Pourreza et al., 2025, Papicchio et al., 2025], but these works focused on the one-shot generation task. In contrast, our work considers RL for Text2SQL in the multi-turn generation setting.

## 3 Multi-Turn SQL Interactions

When Text2SQL systems operate in the single-turn setting, they need to understand the schema, disambiguate the request, plan joins/aggregations, and write a flawless query in one shot. Such systems are brittle on real databases as they are unable to adapt to unexpected errors, either in the database itself or in its own generations.

In the real world, a human data engineer will interact with a database over multiple turns, planning their actions, probing the database to verify their assumptions, and revising their queries accordingly. This flexibility allows human data engineers to overcome errors in their preconceptions about the database. As the data engineer spends more time interacting with the database, their query becomes more refined and accurate.

Intuitively, a Text2SQL agent would similarly benefit from the ability to interact with the database over multiple turns. It can explore the schema (e.g., list tables/columns, preview rows), run partial queries to test hypotheses, use execution feedback and errors to repair queries, and ask/resolve ambiguities before committing to a final answer. Such an iterative loop mirrors how humans work with unfamiliar databases and provides richer, verifiable signals (syntax checks, execution success, result comparisons), which improves robustness and typically yields higher accuracy on complex, compositional questions.

To verify our hypothesis, we conducted a simple experiment by applying out-of-the-box LLMs to the difficult BIRD [Li et al., 2023] Text2SQL benchmarks, but varying the number of interaction turns allowed to the agents. To simulate the behaviors of human analysts exploring unfamiliar databases, the model is allowed to:

- Issue free-form exploratory SQL (`<sql>...</sql>`)
- Observe partial result (`<observation>...</observation>`)
- Reflect (`<think>...</think>`) based on the history to improve its query
- Commit to a final answer (`<solution>...</solution>`)

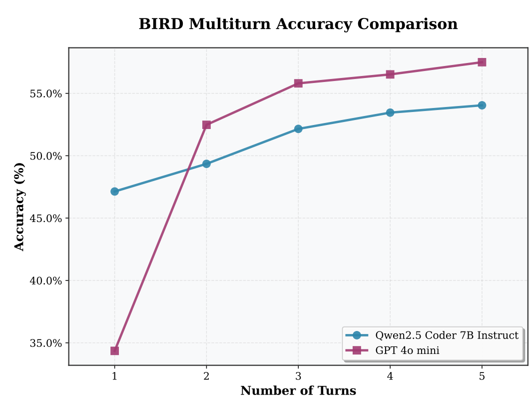


Figure 2: Performance improves as the number of turns scales.

In Figure 2, we observed a general trend of improving performance with the number of turns, indicating that Text2SQL pipelines can indeed benefit from more interactions with the database by correcting errors. In Figure 3, we present an example of such a correction. Here, the task is to find the number of undergraduate students at a specific university. Initially, the model used the wrong

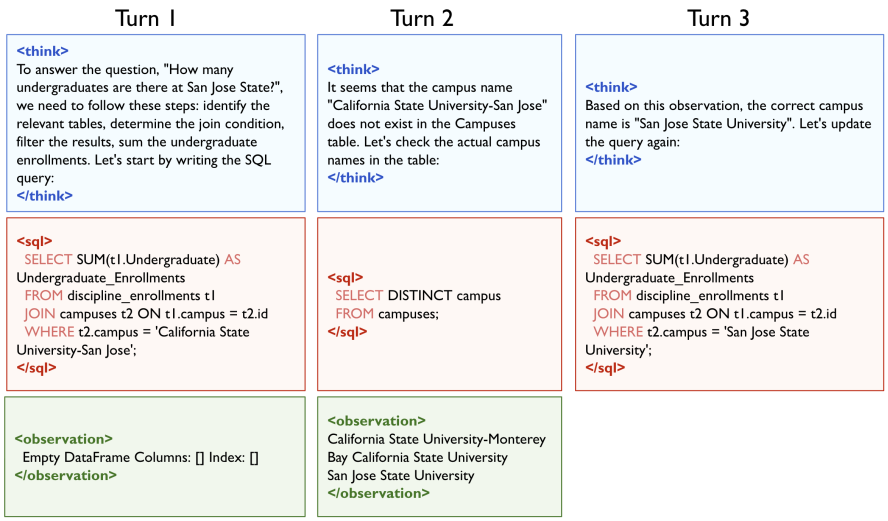


Figure 3: Multi-turn interactions fixing a wrong campus name by checking valid values.

format for the campus name (Turn 1), then queried for valid campus names (Turn 2), and corrected the invalid campus name in its initial attempt (Turn 3).

A natural follow-up question is **whether Text2SQL agents can be trained to take advantage of the multi-turn interactions**. We thus explore the training of LLMs to effectively explore and probe databases to build accurate SQL queries.

## 4 Training Pipeline

We construct an RL training pipeline to teach models to interact with a database. Our code extends the SkyRL framework [Griggs et al., 2025] with the Search-R1 [Jin et al., 2025] agent loop. Figure 4 shows the architecture of our system at a high level. We implemented a parallel SQL execution and evaluation framework with SQLite, allowing multiple rollouts to be executed at once.

Within each rollout, the agent is told the total number of turns it has before it needs to commit to a final solution. Our prompt also contains instructions with the database schema, question, and guidance for interactive SQL generation (i.e., informing the agent of the number of turns available, how to indicate intermediate SQL vs. final answer).

After the agent conducts reasoning ([Think]) and generates an intermediate SQL query ([SQL]), the query is run on the database, and the query results injected back into the trace as an observation (DB Feedback), completing a turn. This continues until the turn limit is reached or the model commits to a final solution ([Solution]), which can then be compared against a ground truth solution via, in our case, execution accuracy. We found that empirically, most correct completions happen within the first 3-4 turns. After 5 turns, models often enter repetitive behaviors. Thus, to save costs we chose to limit the number of interaction turns to 4 in our training runs.

### 4.1 Reward Design

The reward is simple and can be decomposed into (i) a **format reward**, requiring <think> blocks indicating reasoning and <sql> blocks indicating an intermediate SQL query in each turn and a <solution> block indicating commitment to a final query in each rollout, and (ii) an **execution reward**, comparing the executed SQL results of the final committed query with the ground-truth results.

$$\text{Reward} = \begin{cases} -1, & \text{if format is invalid,} \\ 0, & \text{if format is valid but execution is incorrect,} \\ 1, & \text{if format is valid and execution is correct.} \end{cases}$$

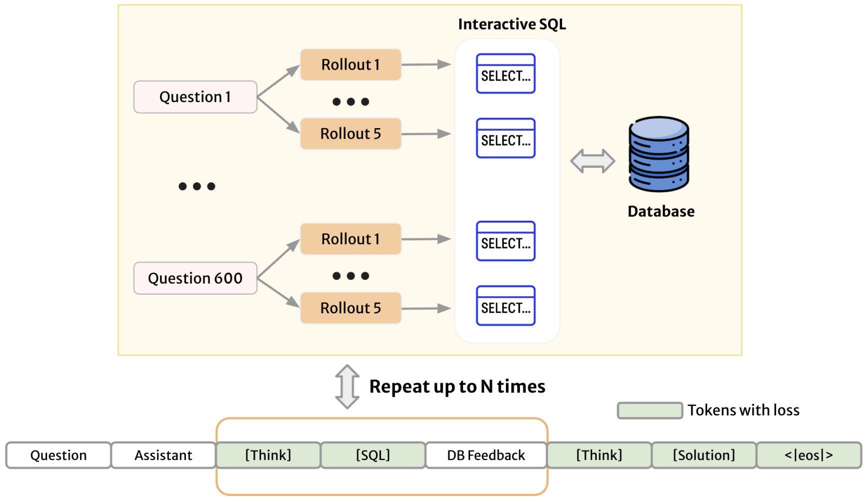


Figure 4: The trajectory from a batch of questions processed by the interaction pipeline, each with multiple independent rollouts. The bottom illustrates the iterative "think", execute SQL, receive feedback, and generate solution loop. Figure adapted from Zheng et al. [2025].

## 5 Experiments

To test our hypothesis that Text2SQL agents can be trained to effectively leverage multi-turn interactions and validate the efficacy of our training pipeline, we trained Qwen2.5-Coder-7B-Instruct using the GRPO [Shao et al., 2024] reinforcement learning algorithm.

**Training Data** We use a total of 653 training samples, with 540 moderately-to-highly complex samples from SynSQL-2.5M [Li et al., 2025] and 113 samples from the Spider train set [Yu et al., 2018]. These selections provide for a mix of difficult synthetic examples from SynSQL-2.5M and high-quality real-world examples from Spider.

**Evaluation Setup** We perform our evaluations on widely-used Text-to-SQL benchmarks, namely Spider [Yu et al., 2018] and its variants Spider-DK [Gan et al., 2021b] (requires domain knowledge), Spider-Syn [Gan et al., 2021a] (synthetic augmentation), and Spider-Realistic [Deng et al., 2021] (mimics real-world ambiguity in column names). We report execution accuracy (EX) based on exact row match. All models are evaluated in multi-turn mode with up to 5 interaction turns, with greedy decoding (temp = 0), except for o4-mini whose API does not allow temperature specification. Instead, for o4-mini we average the performance across 5 evaluation runs.

**Experiment Setup** We train Qwen2.5-Coder-7B-Instruct for multi-turn Text2SQL, allowing each rollout to go for up to 5 turns. We compare the performance of the resulting trained model against existing baseline models in the multi-turn setting.

To investigate the training efficiency of the multi-turn setting, we also trained Qwen2.5-Coder-7B-Instruct as a single-turn Text2SQL agent. We compare the training curves of the single-turn training run against the multi-turn training run. To allow for fair comparisons between the models, which were trained with different number of turns, we evaluate both models in both the single-turn and multi-turn setting.

For all our experiments, we trained at a learning rate of 1e-6 using a batch size of 256 for 35 steps (approximately 14 epochs). 5 rollouts per prompt were sampled at a temperature of 0.6 and top-p of 0.95. The exact prompt we used is in Appendix A.

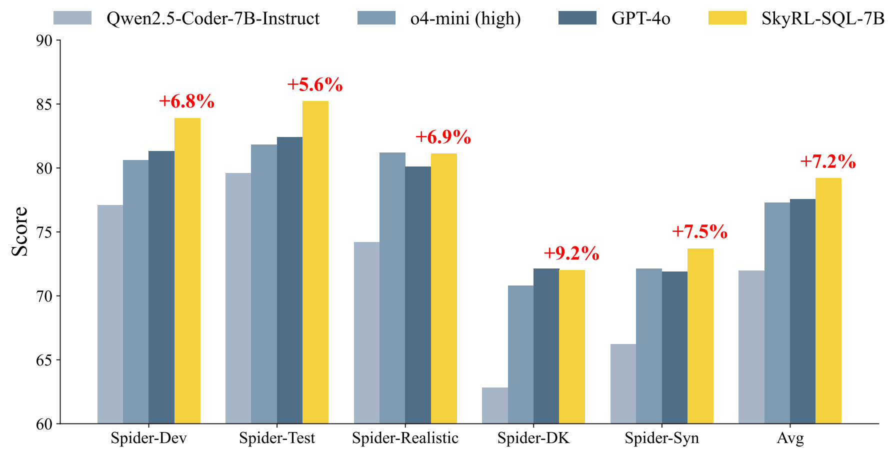


Figure 5: Performance comparison of SKYRL-SQL-7B vs. base and commercial models.

# 6 Results and Discussion

Figure 5 shows the performance of our trained model, which we call SKYRL-SQL-7B, against the performance of the initial base model (Qwen2.5-Coder-7B-Instruct) and the powerful closed-source models GPT-4o and o4-mini with high thinking. SKYRL-SQL-7B achieves the highest overall average (79.2%) across 5 different benchmarks, outperforming GPT-4o by 1.6% and o4-mini by 1.8%. Compared to its base model, Qwen2.5-Coder-7B-Instruct, SKYRL-SQL-7B shows consistent improvements of 5.6% to 9.2%, with an average gain of +7.2% across benchmarks. Exact values are in Table 3 in Appendix B.

These results indicate that training a model to take advantage of multi-turn interactions can lead to dramatic improvements in that setting. In our case, it allowed Qwen2.5-Coder-7B-Instruct, which is worse at multi-turn Text2SQL than state-of-the-art closed-source models, to exceed the performance of those closed-source models. In addition, we also found several key observations during the training process, which we present below.

## 6.1 Key Observations

**Observation 1: Multi-Turn RL Learns Faster and Generalizes Better**
Multi-turn reinforcement learning (RL) not only **learns faster in training steps** but also **generalizes better**, even when **explicit feedback is not available at test time**.

Figure 6 shows the training curves of our model trained in the multi-turn setting and of our model trained in the single-turn setting. Both use identical rewards (format and execution outcomes) and training settings. Relative to a single-turn baseline, multi-turn RL converges 2.8× faster (fewer training steps to a fixed reward threshold) and achieves 16% higher reward after 35 training steps.

These gains arise from the **exploratory nature** of multi-turn RL: the model receives denser feedback by observing how intermediate queries succeed or fail, learns to adjust its reasoning mid-process, and internalizes correct patterns that generalize beyond the training data. By engaging in a closed-loop interaction with the database, the model learns to navigate a richer action space, adjusting its reasoning dynamically — something the single-turn training process cannot do.

Additionally, Table 1 presents results of evaluating the two models on both the single-turn and multi-turn settings. Even without writing interactive queries at test time, the model trained solely in the multi-turn setting outperforms the single-turn counterparts by up to +3.5%, suggesting stronger internalized reasoning. The performance gap widens when evaluating in the multi-turn setting. Here, the multi-turn trained model adapts and refines answers effectively, gaining up to +6.4%, while the single-turn trained model fails to leverage feedback and often produces degraded results.

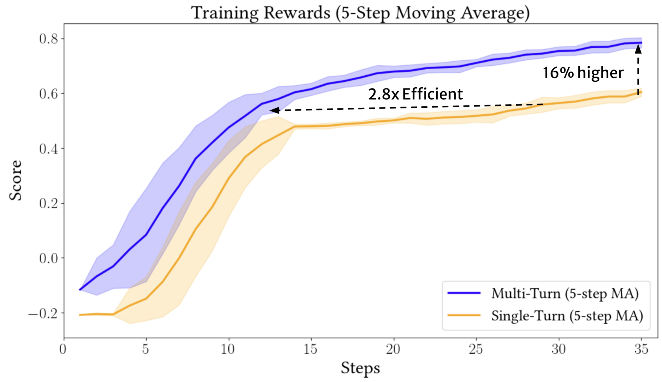


Figure 6: Training curves of multi-turn RL v.s. single-turn RL on the same training data. Multi-turn RL reaches 60% reward 2.8× faster in training steps and achieves +16% higher reward after 35 training steps.

Table 1: Comparison of single-turn vs. multi-turn RL. The model trained in the multi-turn setting outperforms the model trained in the single-turn setting, when evaluated on both single-turn and multi-turn Text2SQL.

| EVAL SETTING | TRAIN SETTING | SPIDER DEV | SPIDER TEST | SPIDER REALISTIC | SPIDER DK | SPIDER SYN | AVG |
|---|---|---|---|---|---|---|---|
| SINGLE-TURN | SINGLE-TURN | 81.2 | 83.8 | 76.8 | 67.9 | 70.1 | 76.0 |
| | MULTI-TURN | 82.4 | 83.7 | 80.3 | 70.5 | 71.2 | **77.6** |
| MULTI-TURN | SINGLE-TURN | 79.5 | 82.2 | 77.6 | 65.6 | 68.4 | 74.7 |
| | MULTI-TURN | 83.9 | 85.2 | 81.1 | 72.0 | 73.7 | **79.2** |

These results suggest that multi-turn training not only allows a model to leverage environmental feedback, but also **improves learning and reasoning in general**.

**Observation 2: Small Data, Simple Reward is Enough for Multi-Turn**
With just **653 samples** and **simple reward**, SKYRL-SQL-7B beats large-scale SFT and single-turn RL with complex reward functions.

Table 2 shows the performance of SKYRL-SQL-7B against two SFT-trained models, Reasoning-SQL-7B [Pourreza et al., 2025], and OmniSQL-7B [Li et al., 2025]. Reasoning-SQL uses complex partial rewards (e.g., execution accuracy, syntax, schema linking) and 8k examples to train a model in the single-turn setting. Evaluation results for this model are taken from Pourreza et al. [2025]; the model is not currently publicly available. OmniSQL-7B is an SFT model trained on the full 2.5M SynSQL dataset. We evaluated this model in single-turn as it does not follow instructions to use multi-turn tool calls.

Despite having 12× less data, SKYRL-SQL-7B outperforms Reasoning-SQL-7B by 5.7%, using multi-turn training with only a basic reward structure (i.e., correct format + execution accuracy) without partial scoring. Compared with OmniSQL-7B, our model performs better (with an average gain of +1%) despite using only **0.03%** of the training data.

**Observation 3: Multi-Turn Interactions Enable Effective Error Corrections.**
The model **learns to reflect, verify, and revise SQL** through interaction with the database — mirroring human-like problem solving.

We found several success patterns throughout the evaluation indicative of what the model learned through database interactions. Here we briefly discuss some of the common patterns analogous to strategies used by human data engineers. First, the model employs **step-by-step verification** by decomposing questions into sub-steps and verifying intermediate results before committing to the

Table 2: Comparison against SFT-trained Text2SQL models. SkyRL-SQL-7B achieves the best average performance while also requiring the smallest train set size by an order of magnitude. Spider-Test and Spider-Realistic results are not available for Reasoning-SQL-7B as the model is not publicly available.

| | Train Set Size | Spider Dev | Spider Test | Spider Realistic | Spider DK | Spider Syn | **Avg** |
|---|---|---|---|---|---|---|---|
| **Reasoning-SQL-7B** | 8K | 78.7 | / | / | 73.3 | 69.3 | 73.8 |
| **OmniSQL-7B** | 2.5M | 81.2 | 87.9 | 76.2 | 76.1 | 69.7 | 78.2 |
| **SkyRL-SQL-7B** | 653 | 83.9 | 85.2 | 81.1 | 72.0 | 73.7 | **79.2** |

final solution. An example trace is presented in Appendix C.1. Second, the model uses feedback from the database to **correct syntax errors** in its queries via reflection or introspection queries (e.g., `PRAGMA` calls); example in Appendix C.2. Third, the model can **correct logical mistakes** when given unexpected results from the database; example in Appendix C.3.

**Observation 4: Multi-Turn Interactions Do Not Fix All Errors.**
The multi-turn RL trained model can fail due to **overconfidence, insufficient exploration, or repetitive loops.**

Though multi-turn RL training can lead to improved performance, there are still some common reasons for failure. First, when the model is **overconfident** in its assumptions, it will not use intermediate turns and immediately commit to an unverified final answer. An example is in Appendix C.4. Second, we find that the model sometimes gets stuck in a loop, attempting the same or very similar intermediate queries in response to unexpected feedback from the database. An example is in Appendix C.5.

### 6.2 Ablations: Models and Prompt Designs

In addition to quantitative and qualitative analysis of SkyRL-SQL-7B, we ran ablations to gain insights into future directions. First, we examined the impact of prompt design with respect to the information included. Specifically, we tested the efficacy of models after removing schema information from the prompt. In this setting, models would be *required* to explore and probe the database to produce answers. Though it would be relatively easy for human data engineers to obtain this information through just a few `PRAGMA` calls, LLM agents struggled, performing up to 42.7% worse compared to providing full information in the prompt. Nevertheless, our model mitigates this effect, exhibiting a smaller degradation and retaining stronger performance under the no-schema setting by actively exploring the database. Details are in Appendix D.1.

Additionally, we tested our training pipeline with the newer Qwen3-8B model. Unlike Qwen2.5, Qwen3 is trained to use <think> blocks out of the box (whereas we had to prompt this behavior into Qwen2.5-7B-Instruct). It was unclear whether this improved reasoning capability would mean multi-turn RL would not be necessary or unable to further improve performance. However, the resulting trained model, which we call SkyRL-SQL-8B, achieved a 3% higher average performance across our evaluation benchmarks, indicating multi-turn RL continues to be effective for thinking models. Further details are in Appendix D.2.

## 7 Conclusion

In this work, we demonstrated the efficacy of multi-turn training for the Text2SQL task. Models trained in such a manner outperform baselines under both single-turn and multi-turn evaluation. Moreover, the success patterns exhibited are analogous to strategies employed by human data engineers. This represents a major step towards the development of autonomous data engineer agents.

However, there are still many exciting opportunities for future work in developing such agents. Though our results demonstrate impressive improvements over baselines, there are still common failure modes that could be addressed. Moreover, our agents are not fully autonomous as they are provided with the database schema and column descriptions as part of the prompt. In the real world, descriptions may be ambiguous, or the schema may be too large to place in the context.

Finally, there are major systems considerations for multi-turn agents. For example, exploratory queries can be costly. During training, the model often issues exploratory queries like `PRAGMA` or `SELECT *` to inspect tables or preview data. Such queries can sometimes fetch hundreds of rows, leading to bloated observations and wasted context. While we truncate large outputs during training, better solutions may guide models via reward shaping or prompt tuning, encouraging the model to use `LIMIT` clauses for more efficient exploration.

## Acknowledgments and Disclosure of Funding

Sky Computing Lab is supported by gifts from Accenture, AMD, Anyscale, Cisco, Google, IBM, Intel, Intesa Sanpaolo, Lambda, Microsoft, NVIDIA, Samsung SDS, SAP, and VMware.

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE 2146752. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

## References

Alnur Ali, Ashutosh Baheti, Jonathan Chang, Ta-Chung Chi, Brandon Cui, Andrew Drozdov, Jonathan Frankle, Abhay Gupta, Pallavi Koppol, Sean Kulinski, and et al. The power of RLVR: Training a leading SQL reasoning model on databricks, Jul 2025. URL https://www.databricks.com/blog/power-rlvr-training-leading-sql-reasoning-model-databricks.

Arian Askari, Christian Poelitz, and Xinye Tang. MAGIC: generating self-correction guideline for in-context text-to-SQL. *arXiv preprint arXiv:2406.12692*, 2024.

Carlo Baronio, Pietro Marsella, Ben Pan, and Silas Alberti. Multi-turn RL training for CUDA kernel generation, 2025. URL https://cognition.ai/blog/kevin-32b.

Asim Biswal, Liana Patel, Siddarth Jha, Amog Kamsetty, Shu Liu, Joseph E Gonzalez, Carlos Guestrin, and Matei Zaharia. Text2SQL is not enough: Unifying AI and databases with TAG. *arXiv preprint arXiv:2408.14717*, 2024.

Xu Chu, Ihab F. Ilyas, Sanjay Krishnan, and Jiannan Wang. Data cleaning: Overview and emerging challenges. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD/PODS'16, page 2201–2206. ACM, June 2016. doi: 10.1145/2882903.2912574. URL http://dx.doi.org/10.1145/2882903.2912574.

Cohere Team. Command A: An enterprise-ready large language model. *arXiv preprint arXiv:2504.00698*, 2025.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. Structure-grounded pretraining for text-to-SQL. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou, editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1337–1350, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.105. URL https://aclanthology.org/2021.naacl-main.105/.

Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R. Woodward, Jinxia Xie, and Pengsheng Huang. Towards robustness of text-to-SQL models against synonym substitution. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long*

*Papers)*, pages 2505–2515, Online, August 2021a. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.195. URL https://aclanthology.org/2021.acl-long.195.

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

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-SQL empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363*, 2023.

Tyler Griggs, Sumanth Hegde, Eric Tang, Shu Liu, Shiyi Cao, Dacheng Li, Charlie Ruan, Philipp Moritz, Kourosh Hakhamaneshi, Richard Liaw, Akshay Malik, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Evolving SkyRL into a highly-modular RL framework, 2025. Notion Blog.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. Towards complex text-to-SQL in cross-domain database with intermediate representation. *arXiv preprint arXiv:1905.08205*, 2019.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. The dawn of natural language to SQL: Are we fully ready? *arXiv preprint arXiv:2406.01265*, 2024a. doi: 10.14778/3681954.3682003.

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

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

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can LLM already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *arXiv preprint arXiv:2305.03111*, 2023.

Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S. Yu. A comprehensive evaluation of ChatGPT's zero-shot text-to-SQL capability. *arXiv preprint arXiv:2303.13547*, 2023.

Andrea Papenmeier, Alfred Sliwa, Dagmar Kern, Daniel Hienert, Ahmet Aker, and Norbert Fuhr. 'A modern up-to-date laptop'–Vagueness in natural language queries for product search. In *Proceedings of the 2020 ACM Designing Interactive Systems Conference*, pages 2077–2089, 2020.

Simone Papicchio, Simone Rossi, Luca Cagliero, and Paolo Papotti. Think2SQL: Reinforce LLM reasoning capabilities for Text2SQL. *arXiv preprint arXiv:2504.15077*, 2025.

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

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

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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

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

Zhewei Yao, Yuxiong He, and Snowflake AI Research. Smaller models, smarter SQL: Arctic-text2sql-R1 tops BIRD and wins broadly, May 2025. URL https://www.snowflake.com/en/engineering-blog/arctic-text2sql-r1-sql-generation-benchmark/.

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

Bohan Zhai, Canwen Xu, Yuxiong He, and Zhewei Yao. ExCoT: Optimizing reasoning for text-to-SQL with execution feedback. *arXiv preprint arXiv:2503.19988*, 2025.

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. DeepResearcher: Scaling deep research via reinforcement learning in real-world environments, 2025. URL https://arxiv.org/abs/2504.03160.

Victor Zhong, Caiming Xiong, and Richard Socher. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.

# A Prompt

**Prompt for Multi-Turn Text2SQL Task**

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

Database Engine:
SQLite

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

External Knowledge:
{external_knowledge}

Question:
{question}

Instructions:
- Make sure you only output the information that is asked in
the question.  If the question asks for a specific column,
make sure to only include that column in the SELECT clause,
nothing more.
- The generated query should return all of the information
asked in the question without any missing or extra
information.
- Before generating the final SQL query, please think through
the steps of how to write the query.  It should include
detailed considerations such as analisying questions,
summarizing relevant findings, brainstorming new ideas,
verifying the accuracy of the current steps, refining any
errors, thinking of how to call SQL tools, and revisiting
previous steps.

Format:
- Conduct thinking inside <think>...</think> blocks every
time you get new observation or information.
- 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.
- The returned dataframe will be truncated in 50 rows if
observation is too long.
- If you find no further exploration is needed or reaches max
turns, you MUST directly provide the final SQL query solution
inside <solution>...</solution>.

---------------- START OF EXAMPLE ----------------
Question:  how many pigs are in the farm?
```

```
Database Schema:
Table: animals
- id (INTEGER, PRIMARY KEY)
- species (TEXT)
- age (INTEGER)
- name (TEXT)

<think>I am querying how many pigs are in the farm. I will
begin by checking if the 'animals' table exists and contains
entries with species = 'pig'.</think>
<sql>SELECT COUNT(*) FROM animals WHERE species =
'pig';</sql>
<observation>
+-------+
| COUNT(*) |
+-------+
| 12 |
+-------+
</observation>
<think>The result indicates that there are 12 pigs in the
farm. Since the question asks for how many pigs, I can now
output the final SQL as the solution.</think>
<solution>SELECT COUNT(*) FROM animals WHERE species =
'pig';</solution>
---------------- END OF EXAMPLE ----------------
```

# B Detailed Results

Here we present the exact benchmark scores for Figure 5.

Table 3: Exact values for Figure 5.

| | Spider-Dev | Spider-Test | Spider-Realistic | Spider-DK | Spider-Synthetic | Average |
|---|---|---|---|---|---|---|
| Qwen-2.5-Coder-7B-Instruct | 77.1 | 79.6 | 74.2 | 62.8 | 66.2 | 71.98 |
| o4-mini (high thinking) | 80.6 | 81.8 | 81.2 | 70.8 | 72.1 | 77.3 |
| GPT-4o | 81.3 | 82.4 | 80.1 | 72.1 | 71.9 | 77.56 |
| SkyRL-SQL-7B | 83.9 | 85.2 | 81.1 | 72.0 | 73.7 | 79.18 |

# C Examples of Success Patterns and Failure Modes

## C.1 Success Patterns 1: Step-by-step verification

The following trace from a task asking for students who don't have cat as their pets serves as an example demonstrating how multi-turn RL enables the model to think step by step: querying students who have pets, listing students who have cats, and eliminating the second list from the first.

```
<think>I need to find the student IDs of students who do not have a cat
    pet. This involves checking the Has_Pet table for students who have
    pets and then excluding those who have cats. I will start by
    identifying students who have pets and then filter out those who have
     cats.</think>
<sql>SELECT StuID FROM Has_Pet;</sql>
<observation>...</observation>
<think>I have the list of students who have pets. Next, I need to find
    which of these students have cats. I will join the Has_Pet table with
     the Pets table to get the pet type for each student.</think>
<sql>SELECT hp.StuID FROM Has_Pet hp
    JOIN Pets p ON hp.PetID = p.PetID
```

```
    WHERE p.PetType = 'cat';</sql>
<observation>...</observation>
<think>I now have the list of students who have cats. To find students
    who do not have cats, I will use the EXCEPT clause to exclude these
    students from the list of students who have pets.</think>
<sql>SELECT StuID
    FROM Has_Pet EXCEPT
        SELECT hp.StuID
        FROM Has_Pet hp JOIN Pets p ON hp.PetID = p.PetID
        WHERE p.PetType = 'cat';</sql>
<observation>...</observation>
<think>The query has been refined and should now provide the correct
    result. I will now provide the final SQL query solution.</think>
<solution>...</solution>
```

### C.2 Success Pattern 2: Correction of syntax errors

The example below illustrates how our trained model looks for the correct column name with a PRAGMA call when encountering an "no such column" error.

```
...
<sql>SELECT Collection_Subset_Details
    FROM Collection_Subsets
    WHERE Collection_Subset_Name = 'Top collection';</sql>
<observation>OperationalError: no such column: Collection_Subset_Details
    ...</observation>
<think>It appears that the column name `Collection_Subset_Details` does
    not exist in the `Collection_Subsets` table. I will double-check the
    schema and ensure I am querying the correct table and column.</think>
<sql>PRAGMA table_info(Collection_Subsets);</sql>
...
```

### C.3 Success Pattern 3: Correction of logical mistakes

In the following example, the task explicitly asks for the semester(s) with the maximum enrollment. The initial SQL used a tautological HAVING clause (count compared to itself), so it never enforced maximality. The model detected this mismatch from execution feedback and repaired the logic by comparing each group's count to the global maximum, thus implementing the intended argmax and correctly returning the top semester(s), tie-aware.

```
...
<sql>SELECT s.semester_name, s.semester_id
    FROM Semesters s
    JOIN Student_Enrolment se ON s.semester_id = se.semester_id
    GROUP BY s.semester_id
    HAVING COUNT(se.student_id) = (
        SELECT COUNT(se2.student_id)
        FROM Student_Enrolment se2 WHERE se2.semester_id = s.semester_id
    )
    ORDER BY s.semester_id;</sql>
<observation>...</observation>
<think>The query now returns the semester with the most students
    registered for each semester. However, I notice that the results are
    not as expected. I need to ensure that for each semester, only the
    semester with the most students registered is returned. I will modify
     the query to use a subquery to filter the results correctly.</think>
<sql>SELECT s.semester_name, s.semester_id
    FROM Semesters s JOIN Student_Enrolment se ON s.semester_id = se.
        semester_id
    GROUP BY s.semester_id
    HAVING COUNT(se.student_id) = (SELECT MAX(student_count)
    FROM (SELECT COUNT(se2.student_id) AS student_count
    FROM Student_Enrolment se2 GROUP BY se2.semester_id))
```

```
    ORDER BY s.semester_id;</sql>
...
```

### C.4 Failure Mode 1: Model overconfidence

The following example trace is from the task "list the names of all distinct paintings ordered by length." The model incorrectly assumes that the length of the paintings should be height times width, whereas it is really just the height of the painting.

```
<think>I need to list the names of all distinct paintings and order them
    by their length. The length of a painting can be calculated using the
     height and width in millimeters. I will use the `DISTINCT` keyword
    to ensure that each painting title is listed only once, and I will
    order the results by the calculated length.</think>
<sql>SELECT DISTINCT title
    FROM Paintings
    ORDER BY height_mm * width_mm;</sql>
...
```

### C.5 Failure Mode 2: Stuck in a loop

The trace below shows how the model gets stuck in a loop, keeping running into the same “no such column” error although it is informed with the observation from the database.

```
<think>...</think>
<sql>SELECT AVG(T1.room_count)
    FROM Rooms AS T1 JOIN Properties AS T2 ON T1.property_id = T2.
        property_id
    JOIN Property_Features AS T3 ON T2.property_id = T3.property_id
    WHERE T3.feature_name = 'garden';</sql>
<observation>OperationalError: no such column: T1.room_count...</
    observation>
<think>...</think>
<sql>SELECT AVG(T1.room_count)
    FROM Rooms AS T1 JOIN Property_Features AS T2 ON T1.property_id = T2.
        property_id
    WHERE T2.feature_name = 'garden';</sql>
<observation>OperationalError: no such column: T1.room_count...</
    observation>
<think>...</think>
<sql>...</sql>
<observation>OperationalError: no such column: T1.room_count...</
    observation>
...
```

## D Ablation Details

### D.1 Removing Schema

In practice, access to the full database schema isn’t always available—human data engineers may only see table names or partial metadata. Therefore, we want to test the models’ robustness and the ability to explore the databases under such incomplete information. Our experiments always included the full database schema in the prompt, in line with standard practice for the Spider datasets. This schema included detailed metadata such as table and column descriptions and example entries. We now evaluate a table-names-only setting (no column definitions/relations/examples) and conduct the evaluations across the Spider benchmarks. The relevant modified portions of the prompt are presented below; all other parts of the prompt remain the same.

**Changes to the Prompt for Multi-Turn Text2SQL Task without Schema**

```
Task Overview:
You are a data science expert. Below, you are provided with
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 Schema: You are not provided with the database
schema and the table names. You need to figure out the
database structure by yourself with the table names.
{table_names}

...

---------------- START OF EXAMPLE ----------------
Question: how many pigs are in the farm?

<think>I am querying how many pigs are in the farm. I will
begin by checking if the 'animals' table exists and contains
entries with species = 'pig'.</think>

...
```

As Table 4 shows, removing the database schema from the prompt causes a large degradation across every Spider split. The baseline Qwen2.5-Coder-7B-Instruct falls from an average of 72.0% to 33.8% (-38.2%). By contrast, SkyRL-SQL-7B drops from 79.2% to 46.8% (-32.4%), consistently retaining more accuracy under schema-free evaluation. This smaller decline indicates our model can better infer and navigate the schema on its own, rather than relying solely on the explicit schema context.

Table 4: Removing the schema from the prompt leads to significant performance drops across all Spider benchmarks.

| | Spider Dev | Spider Test | Spider Realistic | Spider DK | Spider Syn | Avg |
|---|---|---|---|---|---|---|
| *Evaluation with Schema* | | | | | | |
| **Qwen2.5-Coder-7B-Instruct** | 77.1 | 79.6 | 74.2 | 62.8 | 66.2 | 72.0 |
| **SkyRL-SQL-7B** | 83.9 | 85.2 | 81.1 | 72.0 | 73.7 | **79.2** |
| *Evaluation w/o Schema* | | | | | | |
| **Qwen2.5-Coder-7B-Instruct** | 39.8 | 36.9 | 33.3 | 27.7 | 31.2 | 33.8 |
| **SkyRL-SQL-7B** | 55.8 | 50.0 | 43.7 | 42.1 | 42.3 | **46.8** |

### D.2 Thinking Models

Qwen2.5-7B-Instruct is not a native thinking model; it was not trained to emit `<think>` blocks without specific prompting. More recent models have adopted the thinking paradigm, where models are trained to, by default, emit `<think>` blocks.

To test whether our multi-turn training recipe still works for newer thinking models, we ran the same RL recipe with Qwen3-8B as the base model. We make a minor prompt change for the thinking model training and evaluation, dropping the thinking-specific instructions since these models already possess that capability.

**Changes to the Prompt for Multi-Turn Text2SQL Task for Thinking Models**

```
...

Format:
- 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.
- The returned dataframe will be truncated in 50 rows if
observation is too long.
- If you find no further exploration is needed or reaches max
turns, you MUST directly provide the final SQL query solution
inside <solution>...</solution>.

...

---------------- START OF EXAMPLE ----------------
Question:  how many pigs are in the farm?
Database Schema:
Table:  animals
- id (INTEGER, PRIMARY KEY)
- species (TEXT)
- age (INTEGER)
- name (TEXT)

<sql>SELECT COUNT(*) FROM animals WHERE species =
'pig';</sql>
<observation>
+-------+
| COUNT(*) |
+-------+
| 12 |
+-------+
</observation>
<solution>SELECT COUNT(*) FROM animals WHERE species =
'pig';</solution>
---------------- END OF EXAMPLE ----------------
```

As shown in Table 5, the method led to consistent improvements across all Spider benchmarks, indicating that the benefits extend to thinking models. We expect that increasing the training data will yield more substantial performance gains.

Table 5: Comparison between Qwen3-8B and the trained Qwen3-8B (which we refer to as SkyRL-SQL-8B) with the same set of 653 training data. Both models are evaluated for 5 turns with thinking mode on and a temperature of 0.6, aligning the best practice for Qwen3-8B.

| | Spider Dev | Spider Test | Spider Realistic | Spider DK | Spider Syn | **Avg** |
|---|---|---|---|---|---|---|
| **Qwen3-8B** | 79.3 | 81.5 | 76.0 | 69.5 | 69.7 | 75.2 |
| **SkyRL-SQL-8B** | 81.4 | 84.5 | 79.9 | 71.6 | 73.4 | **78.2** |