CSC-SQL: Corrective Self-Consistency in Text-to-SQL via Reinforcement Learning

Lei Sheng*
Wuhan University
of Technology, China
`xuanfeng1992@whut.edu.cn`

Shuai-Shuai Xu
University of Science and
Technology of China, China
`sa517432@mail.ustc.edu.cn`

Abstract

Large language models (LLMs) have demonstrated strong capabilities in translating natural language questions about relational databases into SQL queries. In particular, test-time scaling techniques such as Self-Consistency and Self-Correction can enhance SQL generation accuracy by increasing computational effort during inference. However, these methods have notable limitations: Self-Consistency may select suboptimal outputs despite majority votes, while Self-Correction typically addresses only syntactic errors. To leverage the strengths of both approaches, we propose CSC-SQL, a novel method that integrates Self-Consistency and Self-Correction. CSC-SQL selects the two most frequently occurring outputs from parallel sampling and feeds them into a merge revision model for correction. Additionally, we employ the Group Relative Policy Optimization (GRPO) algorithm to fine-tune both the SQL generation and revision models via reinforcement learning, significantly enhancing output quality. Experimental results confirm the effectiveness and generalizability of CSC-SQL. On the BIRD private test set, our 7B model achieves 71.72% execution accuracy, while the 32B model achieves 73.67%, outperforming other known methods using open source models. The code has been open sourced at `https://github.com/CycloneBoy/csc_sql`.

1 Introduction

The Text-to-SQL task involves translating natural language questions into SQL queries for database access (Katsogiannis-Meimarakis and Koutrika, 2023; Shi et al., 2024). Current mainstream approaches rely on large language models (LLMs) to generate SQL (Liu et al., 2024). Recently, test-time scaling (TTS) (Zhang et al., 2025a) techniques have gained popularity, the most prevalent TTS strategies are Self-Consistency (SC) (Wang et al., 2023; Dong et al., 2023; Gao et al., 2023; Xie et al., 2025) and Self-Correction (Cao et al., 2024; Gao et al., 2024).

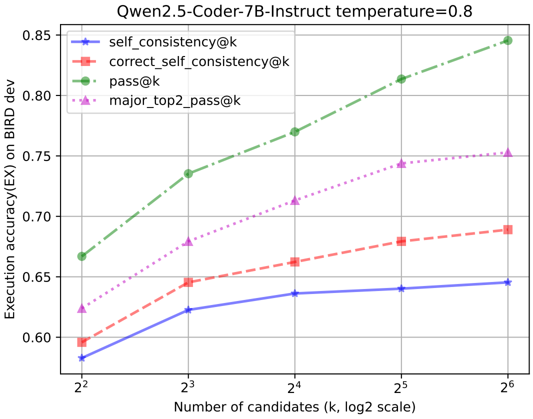

Figure 1: On the BIRD development set, the trend chart of different metrics under different sampling numbers using the Qwen2.5-Coder-7B-Instruct model at a temperature of 0.8. Among them, major_top2_pass@k denotes selecting the two groups with the most votes to calculate the pass@k metric, and self_consistency@k and correct_self_consistency@k represent the results of using the SC method and our CSC method respectively.

A primary limitation of SC is that the most frequently selected output is not always the correct final result. As illustrated in Figure 1, the gap between self_consistency@k and pass@k widens as the number of candidates increases. To address this issue, some methods (Pourreza and Rafiei, 2023; Pourreza et al., 2024, 2025) combine SC with a second-stage corrector to refine SC outputs, while others (Gao et al., 2024; Gorti et al., 2025) utilize a fine-tuned selection model to determine the final output. However, the former approach only revises the top one voting group SQL identified by SC, relying heavily on the correction capability of the base LLM, whereas the latter necessitates comparison among multiple similar SQL candi-

---

[1] Corresponding author: `xuanfeng1992@whut.edu.cn`

Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, pages 1473–1496
December 20-24, 2025 ©2025 Association for Computational Linguistics

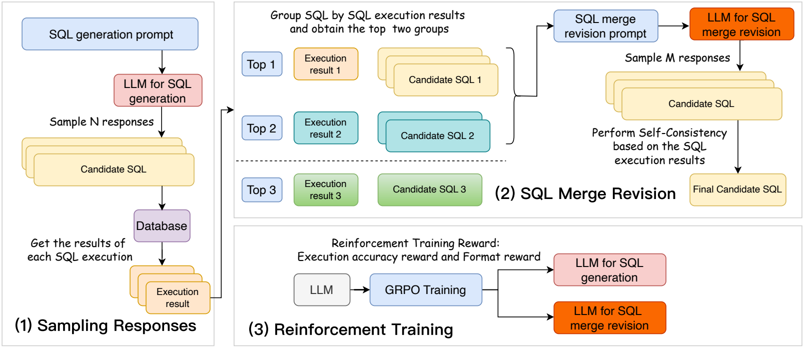

Figure 2: Overview of the proposed CSC-SQL framework

dates (typically more than three), which requires high-quality datasets to be effective. To further investigate the behavior of SC, we propose a new metric, major_top2_pass@k, which calculates pass@k based only on the top two voting groups, determined by SQL execution results. As shown in Figure 1, major_top2_pass@k mitigates the gap between self_consistency@k and pass@k by leveraging only the top two candidate groups.

Based on these analyses, we propose the Corrective Self-Consistency (CSC) framework CSC-SQL. First, the SQL generation model produces N candidate SQLs via parallel sampling. The execution results of these candidates are then used to identify the top two groups based on majority voting. These top two SQL queries, along with the merged schema and their execution outcomes, are used to construct a Merge-Revision template (Sheng et al., 2025). This template is then passed to a revision model, which generates M new candidate SQL queries. Finally, the SC method is applied to select the final output. During the post-training of LLMs (Tie et al., 2025), reinforcement learning (RL) has been shown to significantly enhance reasoning capabilities (Besta et al., 2025). Following the approach of (Pourreza et al., 2025; Ma et al., 2025), we post-train both the SQL generation model and the revision model using the Group Relative Policy Optimization (GRPO) (Shao et al., 2024) algorithm.

We conducted experiments on the BIRD (Li et al., 2024c) development set using models of varying sizes. The results demonstrate that our proposed CSC method consistently improves EX by 0.72%-5.54% compared to the Self-Consistency baseline. Specifically, the 7B model achieved 69.19% EX and the 32B model achieved 71.33% EX. On the BIRD private test set, our 7B model achieves 71.72% EX, and the 32B model achieves 73.67%, surpassing other known methods using open-source models. We also tested the model trained on the BIRD dataset directly on the Spider (Yu et al., 2019) dataset. The experimental results show that our CSC-SQL method has strong generalization ability and also brings certain performance improvements.

2 Methodology

Our CSC-SQL framework, illustrated in Figure 2, consists of three main components: Sampling Responses, SQL Merge Revision and Reinforcement Training.

CSC-SQL framework: First, the SQL generation model produces N candidate SQLs through parallel sampling. These candidates are then grouped and ranked based on their execution results, and the top two groups are selected. If the candidate SQL queries are consistent, they are directly used as the final output. Otherwise, the top two selected SQL queries, along with their execution results, are used to construct a template following the method of (Sheng et al., 2025). This template is input into the merge-revision model to generate M revised candidate SQL queries. Finally, the SC method is applied to these M candidates to select the final SQL output.

Reinforcement Training: We apply the GRPO (Shao et al., 2024) algorithm to perform reinforcement training on both the SQL generation model and the merge-revision model, aiming to enhance their generation capabilities. The reward function comprises two components: execution accuracy reward (R_{EX}) and format reward (R_{Format}). For detailed reward design, please refer to Appendix B.

SQL Merge Revision Dataset: We utilized the Qwen2.5-Coder-7B-Instruct model and its GRPO-trained variant to generate eight candidate SQL queries in parallel on the BIRD training set. These candidates were grouped and voted upon based on their execution results. The two groups receiving the highest number of votes were selected to construct the merged revision dataset.

3 Experiments

3.1 Experiments Setting

We use the BIRD and Spider dataset as the evaluation datasets. Execution accuracy (EX) is adopted as the evaluation metric. Results are reported as the average performance over three runs for each experimental setting. Additional implementation details are provided in Appendix C.

3.2 Main Results

Method	Model	Size	Dev EX(%)	Test EX(%)
CHASE-SQL (Pourreza et al., 2024)	Gemini-1.5-pro	UNK	73.01	73.0
XiYan-SQL (Gao et al., 2024)	GPT-4o	UNK	73.34	75.63
RSL-SQL (Cao et al., 2024)	GPT-4o	UNK	67.21	68.70
CHASE-SQL (Pourreza et al., 2024)	Mistral Large	123B	70.33	-
Reasoning-SQL (Pourreza et al., 2025)	Qwen2.5-Coder-14B-Instruct	14B	72.29	72.78
Alpha-SQL (Li et al., 2025a)	Qwen2.5-Coder-32B-Instruct	32B	69.70	70.26
BASE-SQL (Sheng et al., 2025)	Qwen2.5-Coder-32B-Instruct	32B	67.47	-
XiYan-SQL (Gao et al., 2024)	XiYanSQL-QwenCoder -32B-2412	32B	67.01	69.03
Reward-SQL (Zhang et al., 2025b)	Qwen2.5-Coder-7B-Instruct	7B	68.90	-
Reasoning-SQL (Pourreza et al., 2025)	Qwen2.5-Coder-7B-Instruct	7B	68.05	-
Alpha-SQL (Li et al., 2025a)	Qwen2.5-Coder-7B-Instruct	7B	66.80	-
SQL-R1 (Ma et al., 2025)	Qwen2.5-Coder-7B-Instruct	7B	66.60	-
OMNI-SQL (Li et al., 2025b)	Qwen2.5-Coder-7B-Instruct	7B	66.10	67.97
CSC-SQL	Meta-Llama-3.1-70B	70B	71.69	-
CSC-SQL	XiYanSQL-QwenCoder -32B-2412	32B	71.33	73.67
CSC-SQL	Qwen2.5-Coder-32B-Instruct	32B	70.69	-
CSC-SQL	Qwen2.5-Coder-14B-Instruct	14B	69.60	-
CSC-SQL	OmniSQL-7B	7B	69.62	-
CSC-SQL	XiYanSQL-QwenCoder -7B-2502	7B	69.21	-
CSC-SQL	Qwen2.5-Coder-7B-Instruct	7B	69.19	71.72
CSC-SQL	XiYanSQL-QwenCoder -3B-2502	3B	65.28	-
CSC-SQL	Qwen2.5-Coder-3B-Instruct	3B	63.34	-

Table 1: Performance Comparison of different Text-to-SQL methods on BIRD dev and private test dataset.

The result on the BIRD dataset are shown in Table 1. They demonstrate that our CSC-SQL approach performs competitively compared to previous Text-to-SQL methods. On the private test set, our 32B model achieved an EX of 73.67%, surpassing all other approaches based on open-source models and improving by 4.64 points over the original base model XiYanSQL-QwenCoder-32B-2412. It even outperforms CHASE-SQL, which employs a complex pipeline using a closed-source LLM and fine-tuned open-source models (e.g., combining self-consistency with a second-stage corrector and training SQL selectors to filter final results). Our 7B model reached 69.19% EX on the development set and 71.72% on the test set, outperforming all methods that use the same base model (e.g., Reward-SQL, Reasoning-SQL, and SQL-R1, all trained with the GRPO algorithm) and demonstrating strong competitiveness. Notably, the CSC-SQL method can also be integrated into other frameworks as a replacement for the self-consistency component.

We evaluate the generality of the proposed CSC-SQL by evaluating it on the Spider dataset, directly using the model trained on the BIRD dataset without retraining a new model. Table 2 presents the experimental results of the 3B model, demonstrating that the CSC method consistently outperforms the SC method. This highlights the strong generalization ability of the merge revision model.

3.3 Ablation Study

Table 3 presents the results of our ablation study. Removing any module results in a notable performance decline. Specifically, eliminating the merge revision model leads to a 2.72-point drop in the performance of CSC-SQL, and a 3.51-point drop when the merge revision model is not post-trained with GRPO. These findings indicate that the merge revision model demonstrates strong error correction capabilities only after GRPO training; otherwise, it may introduce negative effects.

Furthermore, when the merge revision model does not incorporate self-consistency (SC) and generates only a single output, performance decreases by just 0.7 points, suggesting that SC plays a relatively minor role during the merge revision generation stage. In another variant, where the merge revision model selects only the top-1 group (similar to the combined SC and second-stage error correction strategy used in CHASE-SQL and DIN-SQL) performance drops by 2.92 points, which is even lower than when the merge revision model is entirely omitted. This result suggests that such an approach is ineffective for small LLMs.

Dataset	SQL Generateion Model	Train Method	Post Method	EX(%)				
				n=4	n=8	n=16	n=32	n=64
Spider Dev	Qwen2.5-Coder-3B-Instruct	-	SC	64.41	68.09	70.21	70.70	70.79
			CSC	64.60(+0.19)	68.76(+0.67)	70.70(+0.49)	71.28(+0.58)	72.15(+1.36)
		GRPO	SC	71.18	73.31	73.11	75.44	76.02
			CSC	71.37(+0.19)	73.98(+0.67)	73.50(+0.39)	75.92(+0.48)	76.69(+0.67)
	XiYanSQL-QwenCoder-3B-2502	-	SC	69.14	72.92	75.44	76.98	77.85
			CSC	71.37(+2.23)	74.18(+1.26)	76.79(+1.35)	77.95(+0.97)	78.72(+0.87)
		GRPO	SC	74.37	76.01	75.24	75.92	76.21
			CSC	74.47(+0.1)	76.21(+0.2)	75.44(+0.2)	76.40 (+0.48)	76.11(-0.1)
Spider Test	Qwen2.5-Coder-3B-Instruct	-	SC	69.12	72.57	73.82	74.80	75.59
			CSC	69.77(+0.65)	73.50(+0.93)	75.55(+1.73)	76.39(+1.59)	76.99(+1.40)
		GRPO	SC	74.20	75.78	76.99	76.48	77.50
			CSC	74.57(+0.37)	76.53(+0.75)	77.60(+0.61)	77.50(+1.02)	78.16(+0.66)
	XiYanSQL-QwenCoder-3B-2502	-	SC	72.29	76.15	76.67	77.27	78.06
			CSC	73.22(+0.93)	76.94(+0.79)	78.06(+1.39)	78.76(+1.49)	79.37(+1.31)
		GRPO	SC	75.45	76.43	76.99	77.04	77.69
			CSC	76.06(+0.61)	77.08(+0.65)	77.92(+0.93)	78.20(+1.16)	78.57(+0.88)

Table 2: The table shows the EX comparison results of different SQL generation models with different post-processing methods of SC and CSC under different sampling numbers on the Spider development and test set. SC represents the direct use of Self-Consistency, and CSC represents the Corrective Self-Consistency method we proposed. The N represents the number of candidate results generated by the SQL generation model. The number of candidate results of the SQL merge revision model is fixed at 8. The 3B SQL generation model uses the 3B SQL merge revision model.

Method	Dev EX (%)	ΔEX (%)
CSC-SQL	68.30	-
with SQL generation model samples is 8	66.96	-1.34
with merge revision model samples is 1	67.60	-0.70
with merge top k group size is 1	65.38	-2.92
w/o merge revision model	65.58	-2.72
w/o GRPO training for merge revision model	64.79	-3.51
w/o GRPO training for SQL generate model	66.41	-1.89

Table 3: Ablation study of CSC-SQL on the BIRD development set. Its basic model is Qwen2.5-Coder7B-Instruct, and the number of SQL generated is 16.

SQL Generation Model	Train Method	G Size (n)	SC EX (%)	R Size (m)	Merge top k group size	CSC EX(%)	
						With GRPO	
						3B	7B
Qwen2.5-Coder-7B-Instruct	-	8	61.86	8	1	62.32	62.58
					2	**63.86**	**63.93**
					3	63.16	63.03
		16	63.65	8	1	62.77	62.84
					2	**66.47**	**66.78**
					3	63.42	63.95
	GRPO	8	65.02	8	1	64.73	64.73
					2	**66.88**	**66.91**
					3	65.12	65.05
		16	65.17	8	1	65.44	65.38
					2	**67.88**	**67.93**
					3	65.90	65.97

Table 4: The table shows EX of the SQL merge revision model under different merge revision top K group sizes using the 3B and 7B Qwen2.5-Coder-Instruct models on the BIRD development set. G size(n) and R size(m) represent the number of parallel samples of the SQL generation model and the merge revision model, respectively.

3.4 Analysis

3.4.1 Different merge revision top k group sizes

To analyze the impact of different top k group sizes on merge revision model, we evaluated 7B model on the BIRD development set. We fixed the number of merge revision candidates to 8 and tested group sizes of 1, 2, and 3. The experimental results are presented in Table 4. Among the configurations, a group size of 2 yields the best performance and is used as our default setting. A group size of 1 corresponds to standard revision mode, where each example is corrected individually; however, this results in limited improvement and, in some cases, performance degradation. When the group size is increased to 3, performance improves slightly but not significantly.

3.4.2 Different merge revision model sizes

To evaluate the impact of CSC across different merge revision model sizes and parallel sampling size, we conducted a detailed experimental, presented in Table 5. A comparison between the merge revision model with and without GRPO training reveals that the model without GRPO often results in degraded performance, indicating its limited error correction capability. In contrast, GRPO training enhances the model's ability to correct errors. When the SQL generation model and the number of its samples remain fixed, and the merge revision model samples only once (i.e., no second-round

SQL Generation		G Size (n)	SC EX (%)	R Size (m)	CSC EX(%)			
					Without GRPO		With GRPO	
Model	Training				3B	7B	3B	7B
Qwen2.5-Coder-7B-Instruct	-	8	61.86	1	56.45	59.65	63.04	63.62
				4	58.86	61.53	63.91	64.04
				8	59.71	61.73	63.86	63.93
				16	60.23	62.45	63.84	64.04
		16	63.65	1	53.91	58.86	64.66	65.64
				4	57.82	62.25	66.30	66.67
				8	59.77	62.25	66.47	66.78
				16	60.30	62.77	66.51	66.73
	GRPO	8	65.02	1	58.47	62.90	66.16	66.68
				4	62.84	63.95	66.84	67.01
				8	61.79	64.14	66.88	66.91
				16	63.16	63.62	67.08	67.06
		16	65.17	1	58.01	63.10	66.94	67.60
				4	61.47	64.86	67.95	67.86
				8	62.90	64.79	67.88	67.93
				16	62.71	65.25	67.88	67.95

Table 5: The table shows the corresponding EX of the SQL merge revision model under different configuration parameters using the 3B and 7B Qwen2.5-Coder-Instruct models on the BIRD development set without GRPO training and after GRPO training. G size(n) and R size(m) represent the number of parallel samples of the SQL generation model and the merge revision model, respectively.

SC voting is performed), the GRPO-trained merge revision model still demonstrates strong error correction capabilities.

Further performance gains can be achieved by increasing the number of samples in the merge revision model and applying a second SC voting step. However, once the number of merge revision samples increases to 4-16, the size of the merge revision model and the number of its parallel samples have minimal impact on the final execution accuracy (EX). In contrast, increasing the number of samples generated by the SQL generation model has a more substantial effect on EX. This suggests that the primary factor influencing CSC performance is the number of samples generated by the SQL generation model. A higher number of samples results in a larger value of major_top2_pass@k (as shown in Figure 1), thereby increasing the likelihood that the top two SQL candidates selected for merge revision are accurate. Consequently, this leads to improved performance of the SQL merge revision model. These findings also support the trend observed in Figure 1: for most models, as the number of SQL generation samples n increases, the performance gains of CSC over SC also increase.

For more other detailed analysis, please refer to Appendix D.

4 Conclusion

In this paper, we address the limitations of the SC method in the Text-to-SQL task, specifically that the result with the most votes is not always the optimal one. We propose a novel RL-based framework, CSC-SQL, which improves upon SC by feeding the top two voted SQL candidates into a merge revision model for SQL regeneration. This framework is further enhanced through post-training using the GRPO algorithm, resulting in significant performance gains over the standard SC approach. We conduct extensive experiments on the BIRD and Spider datasets, demonstrating the effectiveness and generalizability of CSC-SQL. Notably, small models trained with CSC-SQL can outperform larger models of the same architecture using SC. In future work, we plan to integrate strategies such as Adaptive Self-Consistency and Soft Self-Consistency to improve the efficiency of parallel sampling.

5 Limitations

The CSC-SQL method requires more computational resources than the standard Self-Consistency approach. After parallel sampling, it performs result merging and correction, which increases computational overhead. To mitigate this, methods such as Adaptive Self-Consistency (Aggarwal et al., 2023) and Soft Self-Consistency (Wang et al., 2024) can be incorporated to enhance sampling efficiency. This enhancement is already part of our future work plan. Additionally, CSC-SQL relies on sufficient diversity among sampled results; if the model generates mostly consistent outputs, the CSC-SQL method may not yield significant performance improvements.

6 Acknowledgements

We appreciate all the constructive comments from the anonymous reviewers, which greatly helped improve the quality of the manuscript.

References

Pranjal Aggarwal, Aman Madaan, Yiming Yang, and Mausam. 2023. Let‘s sample step by step: Adaptive-consistency for efficient reasoning and coding with LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 12375–12396, Singapore. Association for Computational Linguistics.

Maciej Besta, Julia Barth, Eric Schreiber, Ales Kubicek, Afonso Catarino, Robert Gerstenberger, Piotr Nyczyk, Patrick Iff, Yueling Li, Sam Houliston, Tomasz Sternal, Marcin Copik, Grzegorz Kwaśniewski, Jürgen Müller, Łukasz Flis, Hannes Eberhard, Hubert Niewiadomski, and Torsten Hoefler. 2025. Reasoning language models: A blueprint. *Preprint*, arXiv:2501.11223.

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

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 181 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, lu Chen, Jinshu Lin, and Dongfang Lou. 2023. C3: Zero-shot Text-to-SQL with ChatGPT. *arXiv preprint*. ArXiv:2307.07306 [cs] version: 1.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *arXiv preprint*. ArXiv:2308.15363 [cs] version: 4.

Yingqi Gao, Yifu Liu, Xiaoxia Li, Xiaorong Shi, Yin Zhu, Yiming Wang, Shiqi Li, Wei Li, Yuntao Hong, Zhiling Luo, Jinyang Gao, Liyu Mou, and Yu Li. 2024. Xiyan-sql: A multi-generator ensemble framework for text-to-sql. *arXiv preprint arXiv:2411.08599*.

Satya Krishna Gorti, Ilan Gofman, Zhaoyan Liu, Jiapeng Wu, Noël Vouitsis, Guangwei Yu, Jesse C. Cresswell, and Rasa Hosseinzadeh. 2025. Msc-sql: Multi-sample critiquing small language models for text-to-sql translation.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-SQL in cross-domain database with intermediate representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4524–4535, Florence, Italy. Association for Computational Linguistics.

Mingqian He, Yongliang Shen, Wenqi Zhang, Qiuying Peng, Jun Wang, and Weiming Lu. 2025. Star-sql: Self-taught reasoner for text-to-sql. *Preprint*, arXiv:2502.13550.

George Katsogiannis-Meimarakis and Georgia Koutrika. 2023. A survey on deep learning approaches for text-to-SQL. *The VLDB Journal*, 32(4):905–936.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. *Preprint*, arXiv:2309.06180.

Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. 2024. MCS-SQL: Leveraging Multiple Prompts and Multiple-Choice Selection For Text-to-SQL Generation. *arXiv preprint*. ArXiv:2405.07467 [cs] version: 1.

Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024a. The Dawn of Natural Language to SQL: Are We Fully Ready? *arXiv preprint*. ArXiv:2406.01265 [cs] version: 1.

Boyan Li, Jiayi Zhang, Ju Fan, Yanwei Xu, Chong Chen, Nan Tang, and Yuyu Luo. 2025a. Alpha-sql: Zero-shot text-to-sql using monte carlo tree search. *Preprint*, arXiv:2502.17248.

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. 2025b. Omnisql: Synthesizing high-quality text-to-sql data at scale. *Preprint*, arXiv:2503.02240.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024b. CodeS: Towards Building Open-source Language Models for Text-to-SQL. *arXiv preprint*. ArXiv:2402.16347 [cs] version: 1.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, and 1 others. 2024c. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36.

Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuyu Luo, Yuxin Zhang, Ju Fan, Guoliang Li, and Nan Tang. 2024. A Survey of NL2SQL with Large Language Models: Where are we, and where are we going? *arXiv preprint*. ArXiv:2408.05109 [cs].

Shuai Lyu, Haoran Luo, Zhonghong Ou, Yifan Zhu, Xiaoran Shang, Yang Qin, and Meina Song. 2025. Sql-o1: A self-reward heuristic dynamic search method for text-to-sql. *Preprint*, arXiv:2502.11741.

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

OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, and 244 others. 2024. Openai o1 system card. *Preprint*, arXiv:2412.16720.

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

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

Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. *arXiv preprint*. ArXiv:2304.11015 [cs] version: 3.

Mohammadreza Pourreza and Davood Rafiei. 2024. DTS-SQL: Decomposed text-to-SQL with small large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 8212–8220, Miami, Florida, USA. Association for Computational Linguistics.

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

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

Lei Sheng, Shuai-Shuai Xu, and Wei Xie. 2025. Base-sql: A powerful open source text-to-sql baseline approach. *Preprint*, arXiv:2502.10739.

Liang Shi, Zhengju Tang, and Zhi Yang. 2024. A Survey on Employing Large Language Models for Text-to-SQL Tasks. *arXiv preprint*. ArXiv:2407.15186 [cs] version: 1.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. CHESS: Contextual Harnessing for Efficient SQL Synthesis. *arXiv preprint*. ArXiv:2405.16755 [cs] version: 1.

Guiyao Tie, Zeli Zhao, Dingjie Song, Fuyang Wei, Rong Zhou, Yurou Dai, Wen Yin, Zhejian Yang, Jiangyue Yan, Yao Su, Zhenhan Dai, Yifeng Xie, Yihan Cao, Lichao Sun, Pan Zhou, Lifang He, Hechang Chen, Yu Zhang, Qingsong Wen, and 7 others. 2025. A survey on post-training of large language models. *Preprint*, arXiv:2503.06072.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. 2020. Trl: Transformer reinforcement learning. `https://github.com/huggingface/trl`.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578, Online. Association for Computational Linguistics.

Han Wang, Archiki Prasad, Elias Stengel-Eskin, and Mohit Bansal. 2024. Soft self-consistency improves language model agents. *Preprint*, arXiv:2402.13212.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. *Preprint*, arXiv:2203.11171.

Xiangjin Xie, Guangwei Xu, Lingyan Zhao, and Ruijie Guo. 2025. Opensearch-sql: Enhancing text-to-sql with dynamic few-shot and consistency alignment. *Preprint*, arXiv:2502.14913.

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. 2019. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. *arXiv preprint*. ArXiv:1809.08887 [cs] version: 5.

Shuozhi Yuan, Liming Chen, Miaomiao Yuan, Jin Zhao, Haoran Peng, and Wenming Guo. 2025. Mcts-sql: An effective framework for text-to-sql with monte carlo tree search. *Preprint*, arXiv:2501.16607.

Qiyuan Zhang, Fuyuan Lyu, Zexu Sun, Lei Wang, Weixu Zhang, Wenyue Hua, Haolun Wu, Zhihan Guo, Yufei Wang, Niklas Muennighoff, Irwin King, Xue Liu, and Chen Ma. 2025a. A survey on test-time scaling in large language models: What, how, where, and how well? *Preprint*, arXiv:2503.24235.

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

A Related Work

Early Text-to-SQL method was mostly based on rules and fine-tuned small language models (Wang et al., 2020; Guo et al., 2019). With the revolutionary progress of LLMs, the mainstream method now uses the powerful in-context learning capabilities of LLM to generate SQL (Dong et al., 2023; Pourreza and Rafiei, 2023; Gao et al., 2023; Lee et al., 2024). The pipeline method is adopted to integrate multiple sub-steps such as schema linking, SQL generation, and Self-Correction to improve the accuracy of generation (Talaei et al., 2024; Pourreza et al., 2024; Gao et al., 2024; Xie et al., 2025). (Pourreza and Rafiei, 2024; Li et al., 2024b; Gorti et al., 2025; Sheng et al., 2025) considering issues such as data privacy and execution efficiency, the method of fine-tuning open source models is adopted. With the rise of test time compute (Zhang et al., 2025a) methods, it can significantly improve LLMs performance by adding additional computation during inference. (Gao et al., 2023; Li et al., 2024a; Xie et al., 2025) directly use Self-Consistency to select the final SQL from many candidate SQLs, while (Pourreza et al., 2024; Gao et al., 2024) train a selection model to chose the final SQL, thereby improving the consistency of model generation. (Yuan et al., 2025; Lyu et al., 2025; Li et al., 2025a) use Monte Carlo Tree Search (MCTS) to dynamically expand the reasoning steps to generate SQL. (He et al., 2025; Li et al., 2025b) uses high-quality rationales to perform supervised fine-tune (SFT) on the model, so that the model can first generate an effective reasoning process and then generate the final SQL. Recent reasoning language models (Besta et al., 2025; OpenAI et al., 2024; DeepSeek-AI et al., 2025) use RL-based post-training methods to promote the reasoning and reflection process inside the model, so that the model shows strong performance in complex reasoning tasks. (Pourreza et al., 2025; Ma et al., 2025; Papicchio et al., 2025) uses the GRPO (Shao et al., 2024) algorithm to post-train the model to improve the model's ability to generate SQL.

B Reward Function Design

The execution accuracy reward (R_{EX}) evaluates the correctness of the generated SQL queries by comparing their execution results with those of the gold-standard SQL on the target database. The format reward (R_{Format}) encourages the model to structure its output by including the reasoning process within <think>...</think> tags and the final answer within <answer>...</answer> tags.

$$R_{EX} = \begin{cases} 1, & \text{if execution results is correct.} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

$$R_{Format} = \begin{cases} 1, & \text{if output format is match.} \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

The final reward is a weighted sum of the execution accuracy reward and the format reward:

$$R = R_{EX} + 0.1 * R_{Format} \tag{3}$$

C Experiments Setting

To evaluate the effectiveness of our CSC-SQL framework and the CSC method, we conducted experiments on 11 open-source LLMs. These models encompass diverse architectures and training strategies, enabling a comprehensive assessment of the adaptability of our approach. We employed the GRPO algorithm to fine-tune four different SQL generation models and two merge revision models of varying sizes (Qwen2.5-Coder-3B-Instruct and Qwen2.5-Coder-7B-Instruct, respectively).

We use the TRL (von Werra et al., 2020) library for GRPO training, employing a cosine learning rate scheduler with a warmup ratio of 0.1 and a small learning rate of 3e-6. Each model is trained for one epoch with an effective batch size of 12. For each input prompt, we generate six completions to satisfy the group size requirement for GRPO training. The maximum input template length is set to 8192 tokens, and the maximum generated output length is 1024 tokens. The execution accuracy reward function is assigned a weight of 1.0, while the format reward function is weighted at 0.1. All experiments are conducted on a machine equipped with four NVIDIA GPUs (each with 80 GB VRAM), and the vLLM (Kwon et al., 2023) framework is used to accelerate inference during testing. Unless otherwise specified, all experiments use default sampling parameters, with a temperature of 0.8 and a merged revision sample size of 8.

The method used to represent database table schemas follows a similar approach to previous work (Li et al., 2024b; Talaei et al., 2024). Specifically, we adopt the representation based on CREATE TABLE statements as proposed by OmniSQL

SQL Generation		Post Method	Dev EX(%)				
Model	Training		n=4	n=8	n=16	n=32	n=64
Qwen2.5-Coder-3B-Instruct	-	SC	44.70	49.87	53.19	55.58	56.15
		CSC	45.96(+1.26)	51.43(+1.56)	55.91(+2.72)	58.84(+3.26)	60.39(+4.24)
	GRPO	SC	53.72	56.95	59.06	59.93	60.60
		CSC	54.65(+0.93)	58.41(+1.46)	61.36(+2.30)	62.13(+2.20)	63.34(+2.74)
XiYanSQL-QwenCoder-3B-2502	-	SC	49.22	54.56	56.82	58.19	58.87
		CSC	51.06(+1.85)	57.74(+3.17)	61.10(+4.28)	63.23(+5.04)	64.41(+5.54)
	GRPO	SC	56.76	59.95	60.78	61.23	61.30
		CSC	58.15(+1.39)	62.17(+2.22)	63.49(+2.71)	64.91(+3.68)	65.28(+3.98)
Qwen2.5-Coder-7B-Instruct	-	SC	58.34	62.18	63.35	63.80	64.45
		CSC	59.63(+1.28)	64.28(+2.10)	66.41(+3.06)	67.54(+3.73)	68.70(+4.25)
	GRPO	SC	63.73	64.95	65.58	65.88	66.15
		CSC	64.91(+1.17)	66.96(+2.01)	68.30(+2.72)	68.61(+2.73)	69.19(+3.04)
XiYanSQL-QwenCoder-7B-2502	-	SC	59.26	62.58	63.67	64.84	66.19
		CSC	61.13(+1.87)	65.34(+2.76)	66.93(+3.26)	68.08(+3.24)	69.21(+3.02)
	GRPO	SC	61.41	63.71	64.65	65.69	65.95
		CSC	63.04(+1.63)	66.08(+2.37)	67.71(+3.06)	68.86(+3.17)	69.04(+3.09)
OmniSQL-7B	-	SC	65.86	67.26	67.23	67.42	67.54
		CSC	66.82(+0.96)	68.32(+1.06)	69.21(+1.99)	69.23(+1.81)	69.62(+2.08)
Meta-Llama-3.1-8B-Instruct	-	SC	47.74	52.35	55.74	56.89	58.11
		CSC	49.15(+1.41)	54.65(+2.30)	58.80(+3.06)	61.45(+4.56)	62.84(+4.74)
gemma-3-12b-it	-	SC	57.69	59.34	60.50	61.15	60.89
		CSC	58.89(+1.20)	61.26(+1.91)	63.10(+2.61)	63.73(+2.59)	63.95(+3.06)
Qwen2.5-Coder-14B-Instruct	-	SC	65.49	66.36	67.24	67.17	67.19
		CSC	66.51(+1.02)	68.02(+1.66)	69.28(+2.04)	69.49(+2.33)	69.60(+2.41)
Qwen2.5-Coder-32B-Instruct	-	SC	66.97	67.56	68.54	68.58	68.67
		CSC	68.02(+1.06)	69.12(+1.56)	69.95(+1.41)	70.32(+1.74)	70.69(+2.02)
XiYanSQL-QwenCoder-32B-2412	-	SC	66.11	67.50	68.45	68.75	68.95
		CSC	67.26(+1.15)	69.37(+1.88)	70.57(+2.12)	70.87(+2.13)	71.33(+2.39)
Meta-Llama-3.1-70B	-	SC	65.58	67.43	69.23	69.56	69.82
		CSC	66.30(+0.72)	68.86(+1.43)	70.75(+1.52)	71.45(+1.89)	71.69(+1.87)

Table 6: The table shows the EX comparison results of different SQL generation models with different post-processing methods of SC and CSC under different sampling numbers on the BIRD development set. The 3B SQL generation model uses the 3B SQL merge revision model, and the models of other sizes all use the 7B SQL merge revision model.

(Li et al., 2025b), which incorporates column descriptions, representative values, and question-relevant values in the comment field.

D Additional Analysis

D.1 Analysis of different model with CSC method

To analyze the performance of the SC and CSC methods across models of varying sizes and different numbers of generated SQL queries, we conducted extensive experiments on the BIRD development set. The results, presented in Table 6, demonstrate that:

For the same model and samples size n, the CSC method consistently outperforms SC method. Specifically, CSC achieves a steady improvement in EX ranging from 0.72% to 5.54% compared to SC. For instance, when the sample size is 64, the EX of XiYanSQL-QwenCoder-3B-2502 without GRPO post-training increases from 58.87% to 64.41%, while Meta-Llama-3.1-8B-Instruct improves from 58.11% to 62.84%. In most models, as the number of samples n increases, the performance gain of CSC over SC also increases.

For the same model, GRPO training generally improves EX under both SC and CSC methods. For example, with a sample size of 8, Qwen2.5-Coder-3B-Instruct shows an EX improvement of 7.08% under SC and 6.98% under CSC, while Qwen2.5-Coder-7B-Instruct improves by 2.77% under SC and 2.68% under CSC. However, as the sample size n increases, the benefits of GRPO training diminish. At a sample size of 64, Qwen2.5-Coder-7B-Instruct achieves only a 1.70% improvement under SC and 0.49% under CSC.

For a fixed sampling number n, models within the same family consistently achieve higher EX under the CSC method compared to the SC method, especially as model size increases. For instance, when using the Qwen2.5-Coder model family with a sampling number of 32, EX improves from 58.84% for the 3B model to 70.32% for the 32B model. Moreover, CSC enables smaller language models to outperform larger counterparts within the same family. For example, with a sampling number of 64, Qwen2.5-Coder-7B-Instruct using CSC surpasses both Qwen2.5-Coder-14B-Instruct and Qwen2.5-Coder-32B-Instruct using

Model	Train Method	Post Method	Dev EX(%)			
			Simple	Moderate	Challenge	All
Qwen2.5-Coder-3B-Instruct	-	SC	64.11	45.91	38.62	56.19
		CSC	67.89	50.43	42.76	60.23
	GRPO	SC	68.65	49.14	45.52	60.56
		CSC	71.46	52.80	46.90	63.49
Qwen2.5-Coder-7B-Instruct	-	SC	71.14	56.47	48.28	64.54
		CSC	74.38	63.36	51.72	68.90
	GRPO	SC	71.03	61.21	52.41	66.30
		CSC	73.95	64.01	55.86	69.23

Table 7: The table shows the performance of different models at different difficulty levels on the BIRD development set when the sampling temperature is 0.8 and the SQL generation model samples 64 candidate results in parallel.

SC. These results highlight the efficiency benefits of CSC, as it allows smaller models to substitute for larger ones while maintaining or improving performance.

D.2 Difficulty Analysis

Table 7 presents the performance of the SC and our proposed CSC methods across different difficulty levels on the BIRD development set. The results indicate that the CSC method consistently outperforms the SC method at all difficulty levels.

D.3 Detailed Cost Analysis

When using LLM-based algorithms, computational time and cost are critical considerations. The proposed CSC-SQL method is built on open-source models that require GPU servers for deployment in real-world applications. As an example of inference cost, we consider an H800 GPU server equipped with a single NVIDIA 80GB GPU (hardware configuration: 20-core Xeon Platinum 8458P CPU, 100GB RAM, 500GB SSD, with a rental price of approximately \$2.0 per hour). We evaluated the inference time and cost of 7B and 32B models on the BIRD development set (containing 1,534 questions), and the results are presented in Table 8. Since the merge revision model is 7B and generates 8 correction candidates, the total inference time increases gradually with the number of candidates generated by the SC method, while the proportion of time spent on the merge revision step decreases. For example, when the SQL generation model is 32B and generates 16 candidates, the additional inference time from the merge revision model increases by only about 7%, resulting in an average cost of \$0.0014 per question. This demonstrates an effective balance between performance and cost. For performance-focused applications, the number of SQL generation candidates can be increased to 64. Overall, CSC-SQL offers flexible trade-offs between computational efficiency and model performance.

D.4 Analysis at different sampling temperatures

The self-consistency method determines the final result by voting among multiple sampled candidates. However, when the diversity of these candidates is limited, the effectiveness of voting diminishes. Similar to the self-consistency approach, our proposed CSC method also relies on the diversity of sampled results. To investigate this, we tested four sampling temperatures ranging from 0.2 to 1.0. As shown in Figure 3, regardless of the sampling temperature, all evaluation metrics improve as the number of samples increases. At lower temperatures, limited diversity leads to lower metric scores. When the temperature is set to 0.8 or 1.0, the performance difference is minimal. Therefore, we adopt a temperature of 0.8 in subsequent experiments.

E Output Example Analysis

In this section, we present example output results from the SQL generation model using the Qwen2.5-Coder-7B model trained with GRPO, along with the 7B merge revision model, to illustrate the model's step-by-step reasoning process on BIRD development set.

E.1 Some successful merge revision cases

The examples include outputs from the top-1 group (where the SQL generation is incorrect) and the top-2 group (where the SQL generation is correct) after SC voting, as well as the final top-1 output produced by CSC, which demonstrates that CSC can effectively correct errors made by the SC method. The red part of the CSC output shows its correction process.

Model	Model Size	Train Method	SC Size	SC Use time (Hour)	CSC Use Time (Hour)	The proportion of time that CSC increases (%)	Total Use Time (Hour)	Total Cost ($)	Average Cost Per Question ($)	Bird Dev EX (%)
Qwen2.5-Coder-7B-Instruct	7B	GRPO	4	0.1	0.06	58	0.16	0.32	0.00022	64.91
			8	0.14	0.07	52	0.21	0.42	0.00028	66.96
			16	0.25	0.08	33	0.33	0.66	0.00044	68.30
			32	0.47	0.08	17	0.55	1.1	0.00072	68.61
			64	0.97	0.11	11	1.08	2.16	0.0014	69.19
XiYanSQL-QwenCoder-32B-2412	32B	-	4	0.2	0.07	33	0.27	0.54	0.00035	67.26
			8	0.63	0.07	11	0.7	1.4	0.00091	69.37
			16	1.0	0.07	7	1.07	2.14	0.0014	70.57
			32	1.74	0.1	6	1.84	3.68	0.0024	70.87
			64	3.18	0.1	3	3.28	6.56	0.0043	71.33

Table 8: The table shows the cost and performance of SQL generation on the BIRD development set using our CSC-SQL method with 7B and 32B sizes. The number of candidate SQLs generated by the merge revision model is fixed to 8. The inference cost analysis uses an NVIDIA H800 with 80GB of memory (rented at $2.0 per hour) as an example.

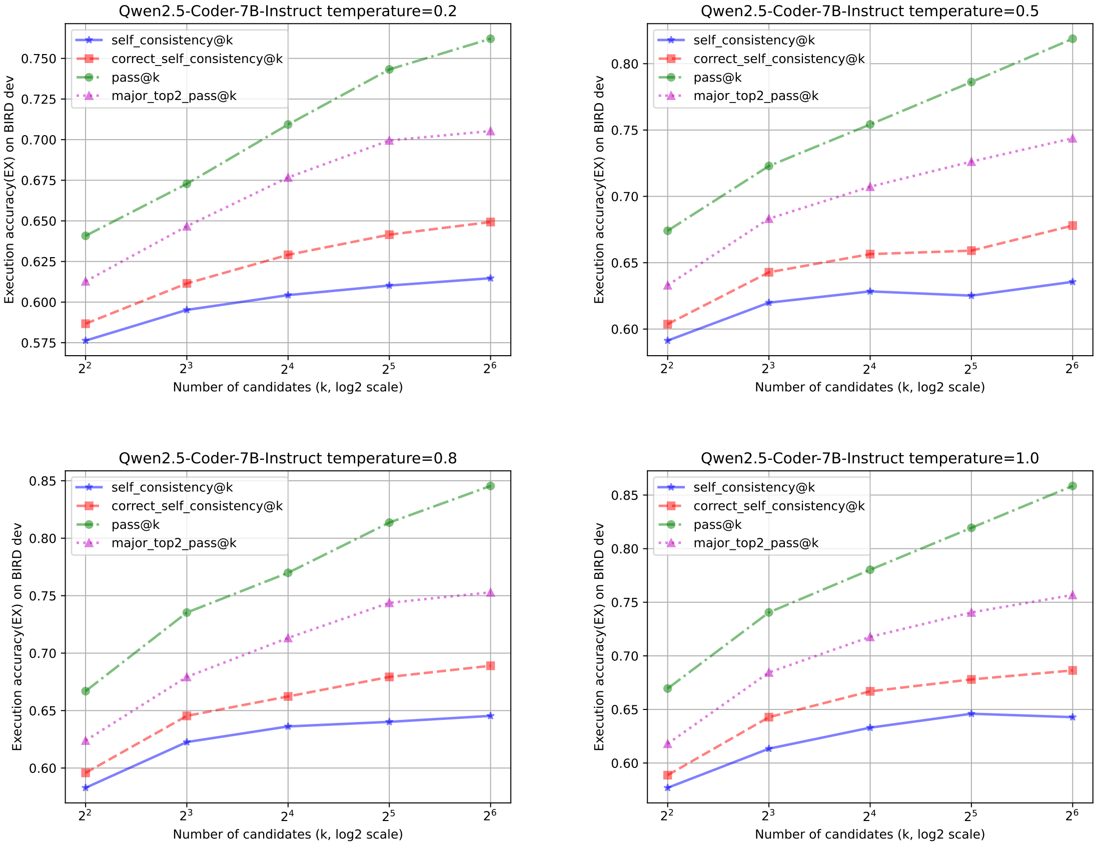

Figure 3: On the BIRD development set, using the Qwen2.5-Coder-7B-Instruct model, when the sampling temperature increases from 0.2 to 1.0, the trend charts of different metric under different sampling numbers.

E.1.1 Successful Case 82

The gold sql for question 82

question_id: 82
db_id: california_schools
question: What is the grade span offered in the school with the highest longitude?
evidence: the highest longitude refers to the school with the maximum absolute longitude value.
difficulty: simple
gold sql: SELECT GSoffered FROM schools ORDER BY ABS(longitude) DESC LIMIT 1

The top 1 group results after SC voting for question 82

<think>
To answer this question, we need to find the school with the highest longitude value and retrieve its grade span offered. Here are the steps to derive the SQL query:

1. **Identify the relevant tables**: We need to access the ‘schools‘ table to get the longitude values and the grade span offered. The ‘Longitude‘ column is in the ‘schools‘ table, and the ‘GSoffered‘ column is also in the same table.

2. **Order the schools by longitude in descending order**: To find the school with the highest longitude, we need to order the schools by the ‘Longitude‘ column in descending order. This will put the school with the highest longitude at the top.

3. **Limit the result to the school with the highest longitude**: Since we are only interested in the school with the highest longitude, we can limit the result to just 1 row.

4. **Select the grade span offered**: Finally, we only need to select the ‘GSoffered‘ column from the ‘schools‘ table.

Putting it all together, the SQL query will select the ‘GSoffered‘ column from the ‘schools‘ table, order the results by ‘Longitude‘ in descending order, and limit the result to 1 row.
</think>
<answer>
SELECT GSoffered FROM schools ORDER BY Longitude DESC LIMIT 1
</answer>

The top 2 group results after SC voting for question 82

<think>
1. **Identify the relevant tables and columns**: The question asks for the grade span offered in the school with the highest longitude. The school information, including its longitude, is stored in the ‘schools‘ table. The grade span offered is stored in the ‘GSoffered‘ column of the same table.

2. **Determine the order and limit**: To find the school with the highest longitude, we need to order the ‘schools‘ table by the ‘Longitude‘ column in descending order and limit the result to the first row using the ‘LIMIT 1‘ clause.

3. **Construct the SQL query**: The SQL query will select the ‘GSoffered‘ column from the ‘schools‘ table, order the results by the ‘Longitude‘ column in descending order, and limit the output to the first row.
</think>
<answer>
SELECT GSoffered FROM schools ORDER BY ABS(Longitude) DESC LIMIT 1
</answer>

The top 1 group results after CSC voting for question 82

<think>
To answer the question "What is the grade span offered in the school with the highest longitude?", we need to follow these steps:

1. **Understand the Data**: The table ‘schools‘ contains information about schools, including their grade span (‘GSoffered‘) and longitude (‘Longitude‘).

2. **Identify the Requirement**: We need to find the school with the highest longitude value and then retrieve its grade span.

3. **Determine the SQL Operation**:
- **ORDER BY**: To sort the schools by longitude in descending order.
- **LIMIT**: To get only the top result, which corresponds to the school with the highest longitude.
- **SELECT**: To specify that we want to retrieve the ‘GSoffered‘ column from the top result.

4. ** distinguish between Absolute Longitude**: The question asks for the "highest" longitude, which could mean either the maximum or minimum longitude value. In this context, it seems to be asking for the maximum absolute value of longitude, which could be either positive or negative. The first draft SQL query uses ‘ABS(Longitude)‘ to achieve this, sorting by the absolute value of longitude in descending order. However, if the intention is to find the school with the most eastern longitude (positive value), we should use ‘Longitude‘ directly.

5. **Construct the SQL Query**:
- **SELECT**: We need to select the ‘GSoffered‘ column.
- **FROM**: We need to specify the ‘schools‘ table.
- **ORDER BY**: We sort by ‘Longitude DESC‘ to get the school with the highest longitude at the top.
- **LIMIT**: We limit the results to 1 to get only the top result.

The draft SQL query provided in the instruction already correctly follows these steps. It uses ‘ABS(Longitude)‘ to find the school with the highest absolute longitude value, which is the intended meaning of "highest longitude" in this context.

Therefore, the correct SQL query is:
</think>
<answer>
SELECT GSoffered FROM schools ORDER BY ABS(Longitude) DESC LIMIT 1
</answer>

E.1.2 Successful Case 232

The gold sql for question 232

question_id: 232
db_id: toxicology
question: Please list top three molecules that have single bonds between two atoms and are not carcinogenic in alphabetical order.
evidence: label = ’-’ means molecules are not carcinogenic; single type bond refers to bond_type = ’-’; list top three molecules refers to return molecule_id and order by molecule_id;
difficulty: moderate
gold sql: SELECT DISTINCT T2.molecule_id FROM bond AS T1 INNER JOIN molecule AS T2 ON T1.molecule_id = T2.molecule_id WHERE T1.bond_type = ’-’ AND T2.label = ’-’ ORDER BY T2.molecule_id LIMIT 3

The top 1 group results after SC voting for question 232

<think>
1. **Identify the key requirements**: The question asks for the top three molecules that have single bonds between two atoms and are not carcinogenic. We need to find the molecules based on these conditions and order them alphabetically by ‘molecule_id‘.

2. **Determine the relevant tables**:
- ‘molecule‘ table contains the ‘ molecule_id‘ and ‘label‘ of molecules.
- ‘bond‘ table contains the ‘bond_type‘ of bonds and is linked to ‘molecule_id‘.

3. **Filter conditions**:
- We need to filter for molecules where ‘label = ’-’‘ (not carcinogenic).
- We need to filter for bonds where ‘bond_type = ’-’‘ (single bond).

4. **Join the tables**:
- Join ‘molecule‘ and ‘bond‘ tables on ‘molecule_id‘.

5. **Order and limit**:
- Order the results by ‘molecule_id‘ and limit the output to the top three.

6. **Formulate the SQL query**:
- Use an INNER JOIN to combine ‘molecule‘ and ‘bond‘ tables.
- Apply WHERE clauses to filter for not carcinogenic and single bonds.
- Use ORDER BY to sort by ‘molecule_id‘.
- Use LIMIT to restrict the output to 3 rows.

</think>
<answer>
SELECT T1.molecule_id FROM molecule AS T1 INNER JOIN bond AS T2 ON T1.molecule_id = T2.molecule_id WHERE T1.label = ’-’ AND T2.bond_type = ’-’ ORDER BY T1.molecule_id LIMIT 3
</answer>

The top 2 group results after SC voting for question 232

<think>
To answer this question, we need to find the top three molecules that have single bonds between two atoms and are not carcinogenic. Here’s a step-by-step approach to derive the SQL query:

1. **Identify the relevant tables**: We need to access information from the ‘molecule‘, ‘bond‘, and ‘connected‘ tables. The ‘molecule‘ table contains the ‘label‘ information, the ‘bond‘ table contains the ‘bond_type‘ information, and the ‘connected‘ table links atoms to bonds and molecules.

2. **Filter molecules not carcinogenic**: We will use the ‘WHERE‘ clause to filter molecules with ‘label = ’-’‘ (not carcinogenic).

3. **Filter single type bonds**: We will also use a ‘WHERE‘ clause to filter bonds with ‘bond_type = ’-’‘ (single bonds).

4. **Group by molecule_id**: Since we want to list the top three molecules, we will use the ‘GROUP BY‘ clause to group the results by ‘molecule_id‘.

5. **Order by molecule_id**: Finally, we will use the ‘ORDER BY‘ clause to sort the results in alphabetical order.

Here’s how we can construct the SQL query:

- Start with the ‘SELECT‘ statement to specify the required columns (‘molecule_id‘).
- Use ‘FROM‘ and ‘INNER JOIN‘ to join the ‘molecule‘, ‘bond‘, and ‘connected‘ tables based on the ‘molecule_id‘ and ‘bond_id‘.
- Add the ‘WHERE‘ clauses to filter molecules not carcinogenic and single type bonds.
- Add the ‘GROUP BY‘ clause to group the results by ‘molecule_id‘.
- Add the ‘ORDER BY‘ clause to sort the results in alphabetical order.
- Use the ‘LIMIT 3‘ clause to return only the top three molecules.
</think>
<answer>
SELECT T1.molecule_id FROM molecule AS T1 INNER JOIN bond AS T2 ON T1.molecule_id = T2.molecule_id WHERE T1.label = ’-’ AND T2.bond_type = ’-’ GROUP BY T1.molecule_id ORDER BY T1.molecule_id ASC LIMIT 3
</answer>

The top 1 group results after CSC voting for question 232

<think>
The task requires us to find the top three molecules that have single bonds and are not carcinogenic, ordered alphabetically by molecule_id. The correct SQL query should:
1. Join the ‘molecule‘ and ‘bond‘ tables based on the ‘molecule_id‘ column.
2. Filter the joined data to include only those rows where the ‘label‘ is ’-’ (indicating the molecule is not carcinogenic) and the ‘bond_type‘ is ’-’ (indicating a single bond).
3. Group the results by ‘molecule_id‘ to exclude duplicates in the result set.
4. Order the results by ‘molecule_id‘ in ascending order.
5. Limit the results to the top 3 rows.

The draft SQL query 1 seems to have an issue; it’s not correctly filtering out duplicates and isn’t grouping the results, which would explain why it returns duplicate rows. The draft SQL query 2 avoids duplicates by grouping by ‘molecule_id‘ but still isn’t excluding duplicate rows, which could be due to a syntax error or oversight.

To correct the query, we need to ensure that the ‘GROUP BY‘ clause correctly groups the results, but since we’re only interested in the ‘molecule_id‘, we should only select that column in the ‘SELECT‘ clause. We also need to ensure that the ‘LIMIT‘ clause correctly restricts the output to the top 3 rows.
</think>
<answer>
SELECT T1.molecule_id FROM molecule AS T1 INNER JOIN bond AS T2 ON T1.molecule_id = T2.molecule_id WHERE T1.label = ’-’ AND T2.bond_type = ’-’ GROUP BY T1.molecule_id ORDER BY T1.molecule_id ASC LIMIT 3
</answer>

E.2 Some failed merge revision cases

We also found some failure cases where the merged revision model did not output the final result correctly, which shows that the merge revision model also has certain limitations and does not work in some situations. In the future, we will further analyze these failure cases to improve the merge revision model.

E.2.1 Failed Case 15

The gold sql for question 15

question_id: 15
db_id: california_schools
question: Which active district has the highest average score in Reading?
evidence: difficulty: simple
gold sql: SELECT T1.District FROM schools AS T1 INNER JOIN satscores AS T2 ON T1.CDSCode = T2.cds WHERE T1.StatusType = ’Active’ ORDER BY T2.AvgScrRead DESC LIMIT 1

The top 1 group results after SC voting for question 15

<think>
To answer the question, we need to follow these steps:

1. **Identify the relevant tables**: The question involves finding districts with high SAT scores and determining which one is active. Therefore, we need to look at the ‘satscores‘ table for SAT scores and the ‘schools‘ table for district information.

2. **Join the tables**: Since the ‘satscores‘ table doesn’t directly contain district details, we need to join it with the ‘schools‘ table using the ‘CDSCode‘ column, which is the primary key in ‘schools‘ and acts as a foreign key in ‘satscores‘.

3. **Filter for active districts**: We need to filter the results to include only active districts, which can be identified by the ‘StatusType‘ column in the ‘schools‘ table.

4. **Calculate the average reading score**: We need to calculate the average reading score for each active district.

5. **Order by average score**: We need to order the results by the average reading score in descending order to find the district with the highest average score.

6. **Limit the results**: Since we are only interested in the district with the highest average score, we can limit the results to 1.

7. **Select the district name**: Finally, we only need to select the district name, which is the ‘District‘ column in the ‘schools‘ table.

Given these steps, the SQL query can be constructed as follows:
</think>
<answer>
SELECT T2.District FROM satscores AS T1 INNER JOIN schools AS T2 ON T1.cds = T2.CDSCode WHERE T2.StatusType = ’Active’ GROUP BY T2.District ORDER BY AVG(T1.AvgScrRead) DESC LIMIT 1
</answer>

The top 2 group results after SC voting for question 15

<think>
To answer this question, we need to follow these steps:

1. **Identify the relevant tables**: The question involves finding information about districts and their satscores. Therefore, the relevant tables are ‘schools‘ and ‘satscores‘.

2. **Understand the relationships**: The ‘schools‘ table has a foreign key ‘CDSCode‘ that references the ‘CDSCode‘ in the ‘satscores‘ table. This means we can join these two tables on the ‘CDSCode‘ column.

3. **Filter active districts**: The question specifies that we should only consider active districts. Therefore, we need to add a filter in the ‘WHERE‘ clause to include only ‘StatusType = ’Active’‘ districts.

4. **Sort by average score in Reading**: To find the district with the highest average score in Reading, we need to sort the results in descending order based on the ‘AvgScrRead‘ column from the ‘satscores‘ table.

5. **Limit the output**: Since we are interested in the district with the highest average score, we can limit the output to the first row.

6. **Select the required columns**: The question asks for the district name, so we need to select the ‘dname‘ column from the ‘satscores‘ table.

Based on these steps, the SQL query can be written as follows:
</think>
<answer>
SELECT T2.dname FROM schools AS T1 INNER JOIN satscores AS T2 ON T1.CDSCode = T2.cds WHERE T1.StatusType = ’Active’ ORDER BY T2.AvgScrRead DESC LIMIT 1
</answer>

The top 1 group results after CSC voting for question 15

<think>
To answer the question "Which active district has the highest average score in Reading?", we need to follow these steps:

1. **Identify the relevant tables and columns**:
- ‘satscores‘ table: contains the average reading scores.
- ‘schools‘ table: contains the districts and their status.

2. **Join the tables**:
- We need to join the ‘satscores‘ table with the ‘schools‘ table on the ‘cds‘ column to link each district with its average reading score.

3. **Filter for active districts**:
- We need to filter the results to include only active districts. This can be done using a WHERE clause.

4. **Group by district**:
- We need to group the results by district to calculate the average score for each district.

5. **Order by average score in descending order**:
- We need to sort the results by the average score in descending order to get the district with the highest score at the top.

6. **Limit the results**:
- Since we only want the district with the highest average score, we can limit the results to 1.

According to the draft SQL provided, the steps seem to be followed correctly, but the table aliases in the query are not consistent. We will correct them to ensure the query works.

The correct SQL query should look like this:
</think>
<answer>
SELECT T2.District FROM satscores AS T1 INNER JOIN schools AS T2 ON T1.cds = T2.CDSCode WHERE T2.StatusType = ’Active’ GROUP BY T2.District ORDER BY AVG(T1.AvgScrRead) DESC LIMIT 1
</answer>

E.2.2 Failed Case 590

The gold sql for question 590

question_id: 590
db_id: codebase_community
question: Which user ID has the lowest view?
evidence: lowest views refers to Min(Views)
difficulty: simple
gold sql: SELECT Id FROM users WHERE Views = (SELECT MIN(Views) FROM users)

The top 1 group results after SC voting for question 590

<think>
To answer the question "Which user ID has the lowest view?", we need to follow these steps:

1. **Identify the relevant table**: The question is about users, their views, and their IDs. The most relevant table for this information is the ‘users‘ table.

2. **Determine the relevant columns**: The columns of interest are ‘Views‘ and ‘Id‘ (user ID).

3. **Order the results by views**: To find the user with the lowest views, we need to order the results in ascending order based on the ‘Views‘ column.

4. **Limit the results**: Since we are only interested in the user with the lowest views, we can limit the results to the first row of the ordered list.

5. **Select the correct column**: The question asks for the user ID, so we need to select the ‘Id‘ column.

Putting these steps together, we can formulate the SQL query.
</think>
<answer>
SELECT Id FROM users ORDER BY Views ASC LIMIT 1
</answer>

The top 1 group results after CSC voting for question 590

<think> To answer the question "Which user ID has the lowest view?", we need to find the user with the minimum number of views. Here are the steps to derive the SQL query:

1. Identify the relevant table: The ‘users‘ table contains information about users, including their views.
2. Determine the column of interest: The ‘Views‘ column in the ‘users‘ table contains the number of views each user has.
3. Find the minimum value of views: To find the user with the lowest view, we need to identify the user with the minimum value in the ‘Views‘ column.
4. Select the user ID: Once we have the minimum value, we select the user ID from the ‘users‘ table that corresponds to this minimum value.

The SQL query will use the ‘MIN‘ function to find the minimum number of views and then use a subquery to find the user ID with this minimum value.
</think>
<answer>
SELECT Id FROM users WHERE Views = (SELECT MIN(Views) FROM users);
</answer>

The top 1 group results after CSC voting for question 590

<think>
Let’s analyze the question and the database schema. The question asks for the user ID with the lowest view count. The database schema has a table named ‘users‘ with two columns: ‘Id‘ and ‘Views‘.

1. **Identify the required operation**: The task requires finding the user ID with the minimum view count.
2. **Determine the appropriate SQL function**: The SQL function ‘MIN‘ can be used to find the minimum value in the ‘Views‘ column.
3. **Formulate the SQL query**:
- We need to select the ‘Id‘ from the ‘users‘ table.
- We use ‘ORDER BY Views ASC‘ to sort the rows in ascending order based on the view count.
- We use ‘LIMIT 1‘ to get only the top row, which corresponds to the user with the lowest view count.
4. **Validate the query**:
- The first draft SQL provided is: ‘SELECT Id FROM users ORDER BY Views ASC LIMIT 1‘
- The execution result confirms that this query correctly returns the user ID with the lowest view count.

Based on these steps, the final SQL query is: ‘SELECT Id FROM users ORDER BY Views ASC LIMIT 1‘
</think>
<answer>
SELECT Id FROM users ORDER BY Views ASC LIMIT 1
</answer>

F Prompt Templates

F.1 Prompt for SQL Generation

SQL Generation Prompt

You first thinks about the reasoning process in the mind and then provides the user with the answer.

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.

Database Engine:
SQLite

Database Schema:
{DATABASE SCHEMA}

This schema describes the database's structure, including tables, columns, primary keys, foreign keys, and any relevant relationships or constraints.

Question:
{EVIDENCE}
{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.

Output Format:
Show your work in <think> </think> tags. And return the final SQLite SQL query that starts with keyword ‘SELECT‘ in <answer> </answer> tags, for example <answer>SELECT AVG(rating_score) FROM movies</answer>.

Let me solve this step by step.

F.2 Prompt for SQL Merge Revision

SQL Merge Revision Prompt

You first thinks about the reasoning process in the mind and then provides the user with the answer.

Task Overview:
You are a data science expert. Below, you are provided with a database schema, a natural language question, some draft SQL and its corresponding execution result. Your task is to understand the schema and generate a valid SQL query to answer the question.

Database Engine:
SQLite

Database Schema:
{DATABASE SCHEMA}

This schema describes the database's structure, including tables, columns, primary keys, foreign keys, and any relevant relationships or constraints.

Question:
{EVIDENCE}
{QUESTION}

Here are some corresponding draft SQL and execute result:
1. {PREDICT_SQL1}
Execution result
{EXECUTE_RESULT1}

2. {PREDICT_SQL2}
Execution result
{EXECUTE_RESULT2}

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.

Output Format:
Show your work in <think> </think> tags. And return the final SQLite SQL query that starts with keyword 'SELECT' in <answer> </answer> tags, for example <answer>SELECT AVG(rating_score) FROM movies</answer>.

Let me solve this step by step.