SLM-SQL: An Exploration of Small Language Models for Text-to-SQL

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 performance in translating natural language questions into SQL queries (Text-to-SQL). In contrast, small language models (SLMs) ranging from 0.5B to 1.5B parameters currently underperform on Text-to-SQL tasks due to their limited logical reasoning capabilities. However, SLMs offer inherent advantages in inference speed and suitability for edge deployment. To explore their potential in Text-to-SQL applications, we leverage recent advancements in post-training techniques. Specifically, we used the open-source SynSQL-2.5M dataset to construct two derived datasets: SynSQL-Think-916K for SQL generation and SynSQL-Merge-Think-310K for SQL merge revision. We then applied supervised fine-tuning and reinforcement learning-based post-training to the SLM, followed by inference using a corrective self-consistency approach. Experimental results validate the effectiveness our SLM-SQL method. On the BIRD development set, the five evaluated models achieved an average improvement of 31.4 points. Notably, the 0.5B model reached 56.87% execution accuracy (EX), while the 1.5B model achieved 67.08% EX. On the BIRD private test set, our 0.5B model achieves 61.82% EX, while the 1.5B model achieves 70.49%. We will release our dataset, model, and code to github: https://github.com/CycloneBoy/slm_sql.

1 Introduction

Converting natural language into SQL for database querying (Text-to-SQL) holds significant application potential and has garnered increasing attention in recent years (Katsogiannis-Meimarakis and Koutrika, 2023; Shi et al., 2024; Liu et al., 2024). Large language models (LLMs) exhibit strong capabilities in language understanding and logical reasoning (Zhao et al., 2025).

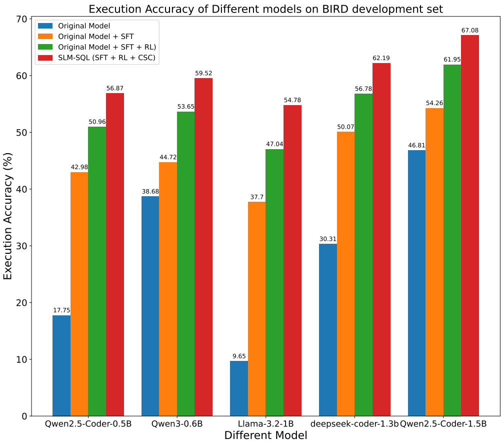

Figure 1: The execution accuracy of different models on the BIRD development set. The model represented by deepseek-coder-1.3b is deepseek-coder-1.3b-instruct.

As a result, most state-of-the-art Text-to-SQL approaches are based on LLMs.

Currently, most open-source LLM-based Text-to-SQL methods utilize models ranging from 3B to 32B parameters, which can be effectively post-trained to enhance their understanding and reasoning capabilities (Tie et al., 2025). Some approaches adopt multi-step pipeline frameworks (Pourreza and Rafiei, 2024; Gorti et al., 2025; Sheng et al., 2025) , with models primarily fine-tuned using the LoRA method (Hu et al., 2021). Others leverage pre-training on carefully curated SQL-centric corpora to improve SQL generation capabilities (Li et al., 2024a, 2025b). Recently, following the success of models such as OpenAI's o1 (OpenAI et al., 2024) and DeepSeek-R1 (DeepSeek-AI et al., 2025), an increasing number of methods (Pourreza et al., 2025; Ma et al., 2025; Papicchio et al., 2025; Sheng and Xu, 2025; Yao et al., 2025) have adopted reinforcement learning

[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 1497–1512
December 20-24, 2025 ©2025 Association for Computational Linguistics

(RL) techniques to further enhance the reasoning abilities of language models.

Recent studies have demonstrated that small language models (SLMs) (Nguyen et al., 2024; Lu et al., 2025) can also achieve competitive reasoning performance (Srivastava et al., 2025). However, the application of SLMs to Text-to-SQL tasks remains underexplored. This paper investigates the capabilities of SLMs with 0.5B to 1.5B parameters in Text-to-SQL scenarios. The two-stage generation approach proposed by CSC-SQL (Sheng and Xu, 2025) is relatively simple yet effective; its 3B model achieved 65.28% execution accuracy (EX) on the BIRD (Li et al., 2024b) development set. Building on this, we propose SLM-SQL, an improved method tailored for smaller models. First, the open-source SynSQL-2.5M dataset (Li et al., 2025b) was processed using heuristic rules to create the SynSQL-Think-916K dataset. Then, inspired by the Corrective Self-Consistency (CSC) technique introduced in CSC-SQL, SynSQL-Think-916K was used to synthesize the SynSQL-Merge-Think-310K dataset. Subsequently, the SQL generation model and SQL merge revision model were trained using supervised fine-tuning (SFT) and reinforcement learning (RL), respectively. Finally, the two-stage inference procedure from CSC-SQL was employed to generate the final SQL outputs.

To evaluate the effectiveness of the SLM-SQL method, we selected five open-source models ranging from 0.5B to 1.5B parameters and conducted experiments on the BIRD and Spider datasets (Yu et al., 2019). The experimental results are presented in Figure 1. On the BIRD development set, the five models achieved an average improvement of 31.4 points, with the 0.5B model reaching 56.87% execution accuracy (EX) and the 1.5B model achieving 67.08% EX, demonstrating the effectiveness of SLM-SQL. We then directly tested the models on the Spider dataset, where all five models also exhibited significant performance gains. The 0.5B model achieved 73.50% EX, while the 1.5B model reached 79.06% EX.

Our contributions are as follows:

(1) Based on the SynSQL-2.5M dataset, we constructed two datasets: SynSQL-Think-916K for SQL generation and SynSQL-Merge-Think-310K for SQL merge revision.

(2) We applied supervised fine-tuning and reinforcement learning to enhance the SQL generation capabilities of SLMs and trained five different models to evaluate the effectiveness and generalization of the proposed SLM-SQL method.

(3) Our method achieved 56.87% EX for the 0.5B model and 67.08% EX for the 1.5B model on the BIRD development set, while the 1.5B model achieves 70.49% EX on the BIRD private test set, demonstrating competitive performance.

2 Related Work

Text-to-SQL methods have evolved from early rule-based approaches and fine-tuned pre-trained language models (Wang et al., 2020; Guo et al., 2019) to large language model (LLM)-based approaches (Liu et al., 2024; Hong et al., 2024). These approaches can be categorized into three groups: (1) in-context learning (ICL)-based methods (Dong et al., 2023; Pourreza and Rafiei, 2023; Gao et al., 2023; Lee et al., 2024), (2) methods based on fine-tuning open-source LLMs (Pourreza and Rafiei, 2024; Li et al., 2024a; Sheng et al., 2025; Li et al., 2025b), and (3) hybrid approaches combining ICL and supervised fine-tuning (SFT) (Talaei et al., 2024; Pourreza et al., 2024; Gao et al., 2024). With the emergence of Test-Time Scaling techniques (Zhang et al., 2025), several works have incorporated strategies such as self-consistency (SC) (Gao et al., 2023; Xie et al., 2025; Sheng and Xu, 2025), self-correction (Pourreza et al., 2024; Gao et al., 2024), and Monte Carlo Tree Search (MCTS) (Yuan et al., 2025; Lyu et al., 2025; Li et al., 2025a) to enhance generation performance. Recently, reinforcement learning (RL) for post-training has proven effective in improving the reasoning capabilities of LLMs (OpenAI et al., 2024; DeepSeek-AI et al., 2025; Yang et al., 2025), and an increasing number of methods (Pourreza et al., 2025; Sheng and Xu, 2025) have adopted this technique. Notably, (Ma et al., 2025; Papicchio et al., 2025; Yao et al., 2025) first leverage synthetic datasets for SFT, followed by RL-based post-training, which significantly enhances SQL generation performance.

Small language models (SLMs) have gained increasing attention due to their efficiency and strong performance (Nguyen et al., 2024; Kim et al., 2025; Allal et al., 2025; Wang et al., 2025). (Srivastava et al., 2025; Subramanian et al., 2025) conducted a comprehensive analysis of the reasoning capabilities of various SLMs, while (Anjum, 2025) evaluated the performance of the distilled reasoning model DeepSeek-R1-1.5B on the Text-to-SQL task. In addition, CSC-SQL (Sheng and Xu, 2025) introduced a merge revision module and ultimately employed a 3B model to achieve an execution accuracy (EX) of 65.28% on the BIRD development set.

3 Methodology

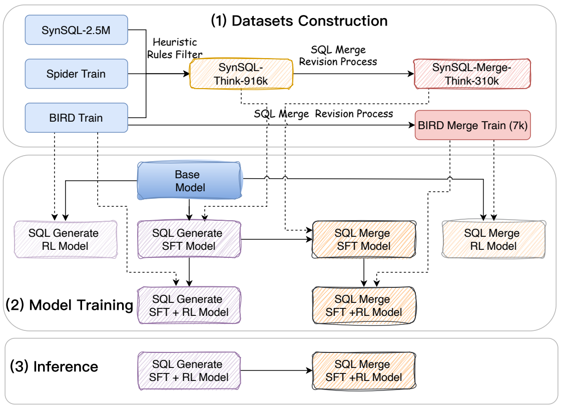

Figure 2: Overview of the proposed SLM-SQL framework.

Our SLM-SQL framework is illustrated in Figure 2 and comprises three main components: dataset construction, model training, and inference.

Dataset Construction:: We use the SynSQL-2.5M dataset (Li et al., 2025b) as the primary data source. This large-scale, high-quality, and diverse synthetic dataset was specifically developed for the Text-to-SQL task. It also includes the Spider and BIRD training sets, for which the OmniSQL method (Li et al., 2025b) has generated Chain-of-Thought (CoT) annotations. To reduce the learning difficulty and dataset size, we apply heuristic preprocessing rules. First, we filter out samples that (i) do not contain the SELECT keyword, (ii) the SQL statement appeared multiple times in the CoT, or (iii) include the '–' comment in the SQL statement. Next, we clean the CoT by removing content after the SQL statement (typically explanations or reflections) and place the preceding portion between <think> and </think> tags. We also remove markdown-specific SQL tags and enclose the SQL statements between <answer> and </answer> tags. Finally, we discard samples with input prompt lengths is higher than 7,000 tokens, resulting in the SynSQL-Think-916K dataset.

We employed the CSC-SQL method to construct a new merge revision dataset, SynSQL-Merge-Think-310K. First, we used the Qwen2.5-Coder-7B-Instruct model to generate eight candidate outputs in parallel on the SynSQL-Think-916K dataset. We then performed group voting based on SQL execution results and selected the two groups with the highest vote counts to create the merge revision dataset.

Model Training: In the CSC-SQL method, both the SQL generation and SQL merge revision models are directly post-trained using RL. In contrast, our SLM-SQL method draws on SQL-R1 (Ma et al., 2025) and Think2SQL (Papicchio et al., 2025), applying SFT on synthetic data before RL-based post-training to enhance the SQL generation capabilities of SLMs.

First, we select a SLM with 0.5B to 1.5B parameters and perform SFT on the SQL generation dataset SynSQL-Think-916K. The training objective is to minimize the cross-entropy loss between the predicted CoT and the synthesized CoT, enabling the model to first generate the reasoning process and then generate SQL. Next, we perform RL-based post-training on the BIRD training set using the Group Relative Policy Optimization (GRPO) (Shao et al., 2024) algorithm, further improving the SQL generation performance. The reward function consists of two components: execution accuracy reward (R_{EX}) and format reward (R_{Format}). See Appendix B for details.

For the merge revision model, we first apply SFT on the SynSQL-Merge-Think-310K dataset using the SFT-trained SQL generation model. We then conduct GRPO-based post-training on the BIRD-Merge-Train dataset, which is synthesized using the CSC-SQL method.

Inference: The inference procedure of our SLM-SQL method follows the same approach as the CSC-SQL method. First, the SQL generation model performs parallel sampling. Then, the generated SQL statements are grouped based on their execution results through a voting mechanism. If the voting results are inconsistent, the SQL is regenerated using the SQL merge revision model; otherwise, the SQL with consistent votes is selected as the final output.

4 Experiments

4.1 Experiments Setting

We conduct experiments on the BIRD and Spider datasets, using the widely adopted execution accuracy (EX) metric to evaluate the performance

Dataset	Train Model	Train Method	Size
SynSQL-2.5M (Li et al., 2025b)	SQL Generate Model	SFT	2,190,988
SynSQL-Think-916k		SFT	916,156
BIRD Train (Li et al., 2024c)		RL	9428
SynsQL-Merge-Think-310k	SQL Merge Revision Model	SFT	310,764
BIRD Merge Train (Sheng and Xu, 2025)		RL	7159

Table 1: Statistics of different datasets.

of our framework. The statistics of all experimental datasets are presented in Table 1. Five SLMs ranging in size from 0.5B to 1.5B were selected as base models for the experiments. Additional implementation details can be found in Appendix A.

Method	Model	Size	Dev EX(%)	Test EX(%)
AskData (Shkapenyuk et al., 2025)	GPT-4o	UNK	75.36	77.14
CHASE-SQL (Pourreza et al., 2024)	Gemini-1.5-pro	UNK	73.01	73.0
RSL-SQL (Cao et al., 2024)	GPT-4o	UNK	67.21	68.70
MCS-SQL (Lee et al., 2024)	GPT-4	UNK	63.36	65.45
Reasoning-SQL (Pourreza et al., 2025)	Qwen2.5-Coder-14B-Instruct	14B	72.29	72.78
OMNI-SQL (Li et al., 2025b)	Qwen2.5-Coder-32B-Instruct	32B	69.23	72.05
Alpha-SQL (Li et al., 2025a)	Qwen2.5-Coder-32B-Instruct	32B	69.70	70.26
XiYan-SQL (Gao et al., 2024)	XiYanSQL-QwenCoder -32B-2412	32B	67.01	69.03
Arctic-Text2SQL-R1 (Yao et al., 2025)	OmniSQL-7B	7B	68.90	68.47
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 (Sheng and Xu, 2025)	XiYanSQL-QwenCoder -3B-2502	3B	65.28	-
CodeS (Li et al., 2024a)	StarCoder	15B	58.47	60.37
DTS-SQL (Pourreza and Rafiei, 2024)	DeepSeek 7B	7B	55.80	60.31
Prem-1B-SQL (Anindyadeep, 2024)	deepseek-coder-1.3b-instruct	1.3B	46.0	51.54
Qwen2.5-Coder (Hui et al., 2024)	Qwen2.5-Coder-1.5B-Instruct	1.5B	28.40	-
SLM-SQL (Our)	Qwen2.5-Coder-0.5B-Instruct	0.5B	56.87	61.82
	Qwen3-0.6B	0.6B	59.52	-
	Llama-3.2-1B-Instruct	1B	54.78	-
	deepseek-coder-1.3b-instruct	1.3B	62.19	-
	Qwen2.5-Coder-1.5B-Instruct	1.5B	67.08	70.49

Table 2: Performance Comparison of different Text-to-SQL methods on BIRD dev and private test dataset.

Model	Train Method	Dev EX(%)		Test EX(%)	
		SC	CSC	SC	CSC
Qwen2.5-Coder -0.5B-Instruct	-	42.13	44.07	42.23	44.31
	SFT	65.31	68.28	67.26	70.72
	SFT + RL	70.60	72.08	70.42	73.50
Qwen3-0.6B	-	63.19	65.15	64.20	66.03
	SFT	68.02	70.05	71.48	73.90
	SFT + RL	72.05	72.99	73.89	75.81
Llama-3.2 -1B-Instruct	-	30.79	32.72	32.71	34.73
	SFT	63.77	66.18	69.06	70.86
	SFT + RL	67.63	69.12	73.05	74.29
deepseek-coder -1.3b-instruct	-	48.23	49.94	49.09	51.19
	SFT	72.86	74.31	75.13	77.19
	SFT + RL	75.47	76.11	77.05	78.08
Qwen2.5-Coder -1.5B-Instruct	-	63.54	65.99	67.61	69.34
	SFT	74.53	76.66	77.35	79.13
	SFT + RL	75.15	76.72	78.42	79.06

Table 3: The table shows the EX comparison results of different models on the Spider dataset. SC stands for direct use of Self-Consistency, and CSC stands for Corrective Self-Consistency. The number of SQL generation model samples is 16, and the number of SQL merge revision model samples is 8. The merge revision model of the CSC method uniformly uses the Qwen2.5-Coder-0.5B-Instruct model trained after SFT and RL.

4.2 Main Results

BIRD Results: Table 2 presents the evaluation results of SLM-SQL and baseline methods on the BIRD dataset. On the BIRD development set, SLM-SQL achieved competitive performance across model sizes ranging from 0.5B to 1.5B. Notably, the 0.6B model attained 59.52% EX, outperforming the 7B DTS-SQL (Pourreza and Rafiei, 2024) and the 15B CodeS (Li et al., 2024a) models. The 1.5B variant of SLM-SQL achieved 67.08% EX, surpassing several significantly larger models, including Qwen2.5-Coder-7B-Instruct and XiYanSQL-QwenCoder-32B. It also outperformed closed-source LLM-based methods such as RSL-SQL (Cao et al., 2024) and MCS-SQL (Lee et al., 2024), which rely on GPT-4 and GPT-4o, respectively. Compared to CSC-SQL (3B), SLM-SQL-1.5B improves performance by 1.8 points. Additionally, compared to Prem-1B-SQL (Anindyadeep, 2024) which is based on the same DeepSeek-Coder-1.3B-Instruct model SLM-SQL-1.3B shows an improvement of 16.19 points, further demonstrating the effectiveness of our approach. On the BIRD private test set, our SLM-SQL also performed exceptionally well, achieving 70.49% EX with a 1.5B model, surpassing many methods that use much larger models (e.g., Arctic-Text2SQL-R1, Alpha-SQL, RSL-SQL, etc.).

Spider Results: We also tested it on the Spider dataset, and the experimental results are presented in Table 3. Five SLM-SQL models of varying sizes achieved competitive performance on the Spider dataset, including an average improvement of 23.83 percentage points on the development set and 24.98 percentage points on the test set. Specifically, SLM-SQL-0.5B achieved 72.08% EX on the development set and 73.5% EX on the test set, while SLM-SQL-1.5B achieved 76.72% EX and 79.06% EX, respectively. Notably, despite not undergoing RL post-training on the Spider dataset, SLM-SQL still achieved strong performance.

4.3 Ablation Study

Method	Dev EX(%)			
	0.5B	ΔEX	1.5B	ΔEX
SLM-SQL	56.87	-	67.08	-
with SQL generation model samples is 16	53.22	-3.64	64.84	-2.24
w/o Corrective Self-Consistency	50.96	-5.91	61.95	-5.13
w/o RL	51.52	-5.35	62.91	-4.17
w/o SFT	34.94	-21.93	58.19	-8.89
w/o SFT and RL	22.14	-34.73	52.26	-14.82

Table 4: Ablation study of SLM-SQL on BIRD development set. 0.5B and 1.5B represent the basic models used, namely Qwen2.5-Coder-0.5B-Instruct and Qwen2.5-Coder-1.5B-Instruct.

Table 4 presents the results of our ablation study. Removing any module results in a significant performance degradation. In particular, excluding the SFT module causes the performance of SLM-SQL-0.5B to drop by 21.93 points and that of SLM-SQL-1.5B by 8.89 points, highlighting the importance of synthetic data for SFT. Removing the Corrective Self-Consistency module leads to an approximate 5-point drop in performance, suggesting that the merge revision model trained on SLMs remains effective in error correction. Additionally, reducing the number of SQL samples from 64 to 16 results in a performance drop of 3.64 points for SLM-SQL-0.5B and 2.24 points for SLM-SQL-1.5B, demonstrating the benefit of increased computational budgets during inference.

5 Conclusion

In this study, we investigate the performance of several 0.5B-1.5B SLMs on Text-to-SQL tasks. To address the limited understanding and reasoning capabilities of small language models, we reorganize the SynSQL-Think-916k dataset for SQL generation and the SynSQL-Merge-Think-310k dataset for SQL merge revision. We then enhance the reasoning ability of SLM-SQL through a combination of supervised fine-tuning and reinforcement learning. Experimental results demonstrate that SLMs in the 0.5B-1.5B range hold significant promise for solving Text-to-SQL tasks.

6 Limitations

In this paper, we focus exclusively on evaluating the performance of SLMs in the Text-to-SQL task using Self-Consistency and Corrective Self-Consistency methods, without exploring other more advanced approaches such as schema linking, agent-based frameworks, or pipeline-based methods. We plan to investigate these areas in future work. The model's generalization performance still needs further testing and verification. In the future, we plan to test more other Text-to-SQL datasets. Moreover, our current study is limited to the Text-to-SQL domain, and extending this research to broader code generation tasks represents an important future direction.

7 Ethical considerations

All datasets and models used in this study are publicly available. We will release the processed datasets, trained models, and source code after the review process to promote transparency and reproducibility. Additionally, our work focuses solely on Text-to-SQL generation, which does not involve the production of harmful or biased content. The synthetic dataset was generated using a large language model without human annotation, thereby ensuring that no forced labor was involved.

8 Acknowledgements

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

References

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, and 1 others. 2025. Smollm2: When smol goes big–data-centric training of a small language model. *arXiv preprint arXiv:2502.02737*.

Anindyadeep. 2024. Premsql: End-to-end local-first text-to-sql pipelines. `https://github.com/premAI-io/premsql`. Accessed: 2024-12-10.

Md Fahim Anjum. 2025. When reasoning beats scale: A 1.5b reasoning model outranks 13b llms as discriminator. *Preprint*, arXiv:2505.03786.

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.

Dejian Yang Zhenda Xie Kai Dong Wentao Zhang Guanting Chen Xiao Bi Y. Wu Y.K. Li Fuli Luo Yingfei Xiong Wenfeng Liang Daya Guo, Qihao Zhu. 2024. Deepseek-coder: When the large language model meets programming – the rise of code intelligence.

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.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

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.

Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. 2024. Next-Generation Database Interfaces: A Survey of LLM-based Text-to-SQL. *arXiv preprint*. ArXiv:2406.08426 [cs] version: 1.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

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

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

Hyunjae Kim, Hyeon Hwang, Jiwoo Lee, Sihyeon Park, Dain Kim, Taewhoo Lee, Chanwoong Yoon, Jiwoong Sohn, Jungwoo Park, Olga Reykhart, and 1 others. 2025. Small language models learn enhanced reasoning skills from medical textbooks. *NPJ digital medicine*, 8(1):240.

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, 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. 2024a. 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. 2024b. 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.

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].

Zhenyan Lu, Xiang Li, Dongqi Cai, Rongjie Yi, Fangming Liu, Xiwen Zhang, Nicholas D. Lane, and Mengwei Xu. 2025. Small language models: Survey, measurements, and insights. *Preprint*, arXiv:2409.15790.

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.

Chien Van Nguyen, Xuan Shen, Ryan Aponte, Yu Xia, Samyadeep Basu, Zhengmian Hu, Jian Chen, Mihir Parmar, Sasidhar Kunapuli, Joe Barrow, Junda Wu, Ashish Singh, Yu Wang, Jiuxiang Gu, Franck Dernoncourt, Nesreen K. Ahmed, Nedim Lipka, Ruiyi Zhang, Xiang Chen, and 9 others. 2024. A survey of small language models. *Preprint*, arXiv:2410.20011.

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 and Shuai-Shuai Xu. 2025. Csc-sql: Corrective self-consistency in text-to-sql via reinforcement learning. *Preprint*, arXiv:2505.13271.

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.

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

Gaurav Srivastava, Shuxiang Cao, and Xuan Wang. 2025. Towards reasoning ability of small language models. *Preprint*, arXiv:2502.11569.

Shreyas Subramanian, Vikram Elango, and Mecit Gungor. 2025. Small language models (slms) can still pack a punch: A survey. *arXiv preprint arXiv:2501.05465*.

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.

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.

Rui Wang, Zhiyong Gao, Liuyang Zhang, Shuaibing Yue, and Ziyi Gao. 2025. Empowering large language models to edge intelligence: A survey of edge efficient llms and techniques. *Computer Science Review*, 57:100755.

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.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

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

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 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. 2025. A survey on test-time scaling in large language models: What, how, where, and how well? *Preprint*, arXiv:2503.24235.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, and 3 others. 2025. A survey of large language models. *Preprint*, arXiv:2303.18223.

A Implementation details

We employed five models as the foundation for our experiments: Qwen2.5-Coder-0.5B-Instruct, Qwen3-0.6B (Yang et al., 2025), Llama-3.2-1B-Instruct (Grattafiori et al., 2024), DeepSeek-Coder-1.3B-Instruct (Daya Guo, 2024), and Qwen2.5-Coder-1.5B-Instruct. Based on these models, we trained corresponding SQL generation models. Additionally, we trained two merge revision models using Qwen2.5-Coder-0.5B-Instruct and Qwen2.5-Coder-1.5B-Instruct.

For supervised fine-tuning (SFT) of SLM-SQL, the training configuration included a learning rate of 2.0e-5 with linear scheduling, a warm-up rate of 0.1, an effective batch size of 1024, and training over 2 epochs. For reinforcement learning (RL), we adopted the GRPO algorithm with a learning rate of 3e-6, a cosine learning rate scheduler, a warm-up rate of 0.1, and 1 training epoch. GRPO rollouts were set to 6, with clip ratios of 0.2 and 0.28. All experiments were conducted on a machine equipped with four NVIDIA GPUs, each with 80 GB of VRAM. For additional implementation details, please refer to the open-source code.

Unless otherwise specified, each SQL generation experiment produces 64 samples, while the merge revision process generates 8 samples. We report the average performance over three independent runs for each experimental setting. During inference, SLM-SQL employs a 0.5B-sized merge revision model for all three base models: Qwen3-0.6B, Llama-3.2-1B-Instruct, and DeepSeek-Coder-1.3B-Instruct.

B Reward Design

During GRPO post-training for the Text-to-SQL task, we adopt a simple reward function composed of two components: execution accuracy reward (R_{EX}) and format reward (R_{Format}).

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

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

The final reward is calculated as the weighted sum of R_{EX} and R_{Format} :

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

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 (%)
SLM-SQL-1.5B	1.5B	SFT + RL + CSC	16 64	0.63 2.43	0.2 0.25	32 10	0.83 2.68	0.22 0.7	0.00014 0.00046	64.84 67.08

Table 5: The table shows the cost and performance of SQL generation on the BIRD development set using our SLM-SQL-1.5B. The inference cost analysis uses an NVIDIA 4090D with 24GB of memory (rented at $0.26 per hour) as an example.

C Additional Analysis

C.1 Test-time Compute Analysis

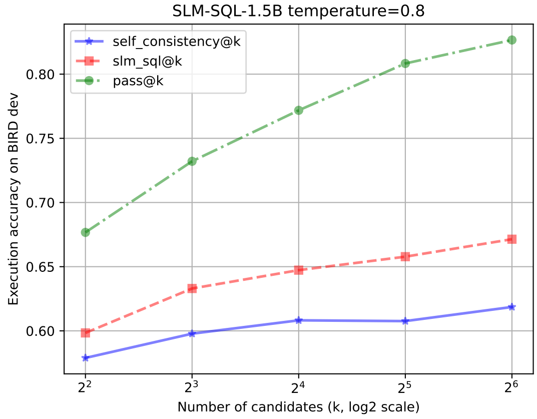

Figure 3: Trend chart of various metrics of SLM-SQL-1.5B under different sampling numbers on the BIRD development set. self_consistency@k and slm_sql@k represent the results of using the self-consistency method and SLM-SQL method respectively.

We further analyzed the impact of varying the number of samples on the performance of SLM-SQL. Experiments were conducted on the BIRD development set with sample sizes ranging from 4 to 64, as illustrated in Figure 3. The results indicate that increasing the number of samples improves the EX of SLM-SQL-1.5B. While the performance gains from the self-consistency method plateau beyond 16 samples, the SLM-SQL method continues to show significant improvement, suggesting its superior effectiveness. The pass@k metric also increases notably with larger sample sizes, reaching up to 0.8264 for pass@64. These findings demonstrate the strong potential of SLMs in solving the Text-to-SQL task.

Furthermore, increasing the number of generated samples leads to higher inference time and computational cost. Table 5 presents the inference time and cost of SLM-SQL-1.5B. Using an NVIDIA 4090D GPU (rental cost approximately $0.26 per hour), increasing the number of generated samples from 16 to 64 raises the total inference time on the BIRD development set from 0.83 to 2.68 hours, and the average cost per question from $0.00014 to $0.00046. This results in roughly a threefold increase in both time and cost, while the EX improves from 64.84% to 67.08%. In practical applications, setting the number of samples to 16 offers a balanced trade-off between performance and efficiency.

C.2 Difficulty Analysis

Table 6 presents the performance of SLM-SQL-0.5B and SLM-SQL-1.5B across varying difficulty levels using different training methods on the BIRD development set. The results demonstrate that each training method (SFT, RL, and CSC) yields notable performance improvements depending on the difficulty level.

Model	Train Method	Dev EX(%)			
		Simple	Moderate	Challenge	All
SLM-SQL-0.5B	-	25.41	7.11	4.14	17.86
	SFT	53.30	30.60	17.93	43.09
	SFT + RL	60.54	38.36	26.21	50.59
	SFT+ RL+ CSC	65.51	46.77	31.72	56.65
SLM-SQL-1.5B	-	56.43	33.41	28.28	46.81
	SFT	61.73	43.97	38.62	54.17
	SFT + RL	69.19	52.37	45.52	50.59
	SFT+ RL+ CSC	73.30	59.91	51.03	67.14

Table 6: The performance of SLM-SQL at different difficulty levels on the BIRD development set.

D Output Example

In this section, we show examples of the output results of the Qwen2.5-Coder-1.5B-Instruct model using different training methods.

Qwen2.5-Coder-1.5B-Instruct original model output results for question 12

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

1. **Identify the relevant tables**: We need to access data from the ‘frpm‘ table for the SAT scores and the ‘satscores‘ table for the eligibility rates.

2. **Determine the join condition**: Since the ‘frpm‘ table and the ‘satscores‘ table share the common column ‘cds‘, we will join these two tables on this column.

3. **Filter the data for schools with an SAT excellence rate over 0.3**: We will use a WHERE clause to filter the rows where the ‘Excellence rate‘ is greater than 0.3.

4. **Calculate the eligible free rate for students aged 5-17**: We will use the formula ‘Eligible free rate = ‘Free Meal Count (Ages 5-17)‘ / ‘Enrollment (Ages 5-17)‘. We will create a subquery to calculate this for each school and then join this subquery with the main ‘frpm‘ table to get the corresponding district name.

5. **Order the results by the eligible free rate in descending order**: To get the highest eligible free rate, we will order the results by the ‘Eligible free rate‘ column in descending order.

6. **Limit the results to the top school**: Since we are only interested in the highest eligible free rate, we will limit the results to the top 1 row.

7. **Select the required columns**: We will select the ‘School‘, ‘District Name‘, ‘Eligible free rate‘, and ‘Excellence rate‘ columns from the result set.

By following these steps, we can generate the final SQL query.
</think>
<answer>SELECT T2.School, T2.District Name, CAST(T2.‘Free Meal Count (Ages 5-17)‘ AS REAL) / T2.‘Enrollment (Ages 5-17)‘ AS Eligible_free_rate, T1.Excellence_rate FROM frpm AS T1 INNER JOIN satscores AS T2 ON T1.cds = T2.cds WHERE T1.‘Excellence rate‘ > 0.3 ORDER BY Eligible_free_rate DESC LIMIT 1
</answer>

Qwen2.5-Coder-1.5B-Instruct after SFT output results for question 223

<think>
To translate the natural language question into an executable SQLite query, let’s break it down step by step:

Step 1: Understand the Question
The question asks for the **atom IDs** associated with a specific bond, identified by ‘TR000_2_5‘. The bond is described in the ‘bond‘ table, and the atom IDs are stored in the ‘connected‘ table.

Step 2: Identify Relevant Tables
1. **‘bond‘ Table**:
- This table contains information about bonds, including their ‘bond_id‘.
- We need to find the ‘atom_id1‘ and ‘atom_id2‘ associated with ‘TR000_2_5‘.

2. **‘connected‘ Table**:
- This table links ‘atom_id1‘, ‘atom_id2‘, and ‘bond_id‘.
- We will use this table to find the atom IDs that are part of the bond identified by ‘TR000_2_5‘.

Step 3: Define the Relationship Between Tables
- The ‘bond‘ table and the ‘connected‘ table are related through the ‘bond_id‘ field.
- Specifically, ‘bond.bond_id‘ is the foreign key in the ‘connected‘ table, linking it to the ‘bond‘ table.

Step 4: Filter for the Specific Bond
- We need to filter the ‘bond‘ table for the row where ‘bond_id = ’TR000_2_5’‘.

Step 5: Retrieve Atom IDs
- After filtering the ‘bond‘ table, we need to retrieve the ‘atom_id1‘ and ‘atom_id2‘ associated with this bond.
- The ‘connected‘ table will help us join the ‘atom_id‘ values to the bond.

Step 6: Write the SQL Query
We will use a ‘JOIN‘ to combine the ‘bond‘ and ‘connected‘ tables, and then filter for the specific bond ‘TR000_2_5‘. Finally, we select the ‘atom_id1‘ and ‘atom_id2‘.

Here is the SQL query:
</think>
<answer>
SELECT connected.atom_id1, connected.atom_id2 FROM bond JOIN connected ON bond.bond_id = connected.bond_id WHERE bond.bond_id = ’TR000_2_5’;
</answer>

Qwen2.5-Coder-1.5B-Instruct after SFT and RL output results for question 20

<think>
To translate the natural language question into an executable SQLite query, let’s break it down step by step:

Step 1: Understand the Question
The question asks: **"How many schools in Amador which the Low Grade is 9 and the High Grade is 12?"**

- **Key points**:
1. We need to count the number of schools (‘COUNT(*)‘).
2. The criteria for the schools are:
- The ‘County Name‘ should be "Amador".
- The ‘Low Grade‘ should be "9".
- The ‘High Grade‘ should be "12".

Step 2: Identify the Relevant Table
From the schema, the ‘frpm‘ table contains the columns:
- ‘County Name‘: To filter schools by county.
- ‘Low Grade‘: To filter schools by the low grade.
- ‘High Grade‘: To filter schools by the high grade.

This table is the primary focus of the query because:
- ‘County Name‘ directly corresponds to the county of interest ("Amador").
- ‘Low Grade‘ and ‘High Grade‘ are used to filter by grade levels.

Step 3: Apply the Filters
The query needs to:
1. Filter rows where ‘County Name = ’Amador’‘.
2. Filter rows where ‘Low Grade = ’9’‘.
3. Filter rows where ‘High Grade = ’12’‘.

These filters ensure that only schools meeting all three conditions are counted.

Step 4: Count the Schools
Once the rows are filtered, the query should count the number of rows that satisfy the conditions. The ‘COUNT(*)‘ function is used for this purpose.

Step 5: Write the SQL Query
Based on the above reasoning, the SQL query is:
</think>
<answer> SELECT COUNT(*) FROM frpm WHERE County Name = ’Amador’ AND ‘Low Grade‘ = ’9’ AND ‘High Grade‘ = ’12’;
</answer>

SLM-SQL-1.5B after CSC output results for question 30

<think>
To answer this question, we need to find the cities with the top 5 lowest enrollment numbers for students in grades 1 through 12. Let's analyze the given draft SQLs:

1. **First SQL Query**:
“‘sql
SELECT s.City FROM frpm AS f INNER JOIN schools AS s ON f.CDSCode = s.CDSCode ORDER BY f.‘Enrollment (Ages 5-17)‘ ASC LIMIT 5
‘‘‘
- This query joins the ‘frpm‘ and ‘schools‘ tables based on the ‘CDSCode‘.
- It orders the results by the enrollment number in ascending order (‘ASC‘).
- The ‘LIMIT 5‘ clause restricts the result to the top 5 cities.

2. **Second SQL Query**:
“‘sql
SELECT s.City FROM schools s JOIN frpm f ON s.CDSCode = f.CDSCode GROUP BY s.City ORDER BY MAX(f.‘Enrollment (Ages 5-17)‘) ASC LIMIT 5
‘‘‘
- This query also joins the ‘frpm‘ and ‘schools‘ tables based on the ‘CDSCode‘.
- It groups the results by city.
- It orders the results by the maximum enrollment number for each city in ascending order (‘ASC‘).
- The ‘LIMIT 5‘ clause restricts the result to the top 5 cities.

Both queries are designed to find the top 5 cities with the lowest enrollment numbers for students. However, they approach the problem differently:
- The first query is more straightforward and directly orders by enrollment.
- The second query groups by city and then orders by the maximum enrollment, which is less intuitive for this specific question.

Given that the question asks for the cities with the top 5 lowest enrollment numbers, the first query is the correct one to use. It directly addresses the requirement by ordering the cities by their enrollment numbers.

<executionresult>

$$('Fremont',),('Oakland',),('Oakland',),('Willows',),('LongBeach',)$$

</executionresult>
</think>
<answer>
SELECT s.City FROM frpm AS f INNER JOIN schools AS s ON f.CDSCode = s.CDSCode ORDER BY f.‘Enrollment (Ages 5-17)‘ ASC LIMIT 5
</answer>

E Prompt Templates

E.1 Prompt for SQL Generation

The SQL generation prompt adopts the prompt proposed by CSC-SQL(Sheng and Xu, 2025).

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.

E.2 Prompt for SQL Merge Revision

The SQL merge revision prompt is slightly adjusted based on the CSC-SQL(Sheng and Xu, 2025) merge revision prompt. As shown in the red part below, we let the model analyze each draft SQL first, compare their differences, and finally generate the final SQL. At the same time, we emphasize that one of the two draft SQLs is correct, guiding the model to make the final decision.

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:
- You should first carefully analyze each draft SQL, compare their differences, and then conduct further analysis based on user questions to determine which draft SQL is correct in the end.
- Remember that one of the draft SQLs is correct. You do not need to generate a new SQL combining their characteristics. Instead, output the draft SQL that you think is correct after careful analysis.
- 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.