ReSQL: Retrieval-augmented Error Reasoning for Text-to-SQL Generation

Anonymous ACL submission

Abstract

Text-to-SQL systems enable users to query databases using natural language, bridging the gap between non-expert users and structured data retrieval. A key challenge for these models is the high frequency of execution errors, particularly in small language models. In this paper, we present ReSQL (**R**etrieval-augmented **e**rror reasoning for Text-to-**SQL**), a framework that enhances the self-debugging capabilities of Text-to-SQL models. ReSQL employs direct fine-tuning on a self-generated error reasoning dataset to improve a model's ability to debug and correct SQL execution errors. We demonstrate that a 7–9B parameter model fine-tuned with ReSQL surpasses GPT-4 on the BIRD and SPIDER benchmarks and outperforms state-of-the-art self-correction methods, achieving more than double the error correction rate compared to standard fine-tuning approaches. Additionally, we show the Retrieval-Augmented SQL Generation further enhances correction capabilities for rare execution error types. We believe ReSQL provides a robust and efficient self-debugging framework for Text-to-SQL models, making it especially valuable for resource-constrained small models.

1 Introduction

Text-to-SQL generation has emerged as a critical component in the field of natural language interfaces to databases, enabling non-expert users to query relational databases using natural language (Katsogiannis-Meimarakis and Koutrika, 2023). By democratizing data access and analysis, it empowers a broader range of users to extract valuable insights from complex database systems.

The evolution of Text-to-SQL systems has been marked by several key phases. Early approaches were often domain-specific, relying on controlled natural language or rule-based methods (Popescu et al., 2004; Meo et al., 1996). Later, researchers developed more domain-independent solutions using supervised models trained on diverse datasets such as BERT (Deng et al., 2020; Lin et al., 2020; Zhong et al., 2020). The advent of deep learning brought about neural models trained on large text and code repositories (Guo et al., 2019; Katsogiannis-Meimarakis and Koutrika, 2021), further improving performance and generalization. Recently, Large Language Models (LLMs) have demonstrated promising performance in Text-to-SQL tasks through in-context learning, including zero-shot and few-shot settings (Gao et al., 2024; Chang et al., 2020; Gu et al., 2023; Dong et al., 2023). However, despite significant advancements in recent years, generating SQL queries from natural language remains a challenging task, particularly for complex queries that involve multiple tables, joins, nested structures, and intricate conditions (Qi et al., 2022; Rai et al., 2023). Benchmarks like SPIDER (Yu et al., 2018) and BIRD (Li et al., 2024b) often expose these difficulties through execution failures that reflect deeper reasoning gaps, demanding explicit identification and systematic correction.

In this paper, we present ReSQL, a novel framework that leverages a self-generated dataset to enhance step-by-step error debugging with reasoning capabilities in Text-to-SQL generation. Designed to expose models to a diverse range of execution errors, ReSQL enables systematic analysis and correction, integrating error correction knowledge directly into training. This approach aims to improve the robustness and reliability of Text-to-SQL models, particularly smaller models that struggle with complex, real-world database querying tasks. Unlike prior approaches that rely primarily on inference-time reranking (Wang et al., 2022) or prompting (Gao et al., 2024), ReSQL introduces structured error reasoning directly into the training process, enabling the model to internalize correction strategies more robustly.

Additionally, we incorporate a retrieval-

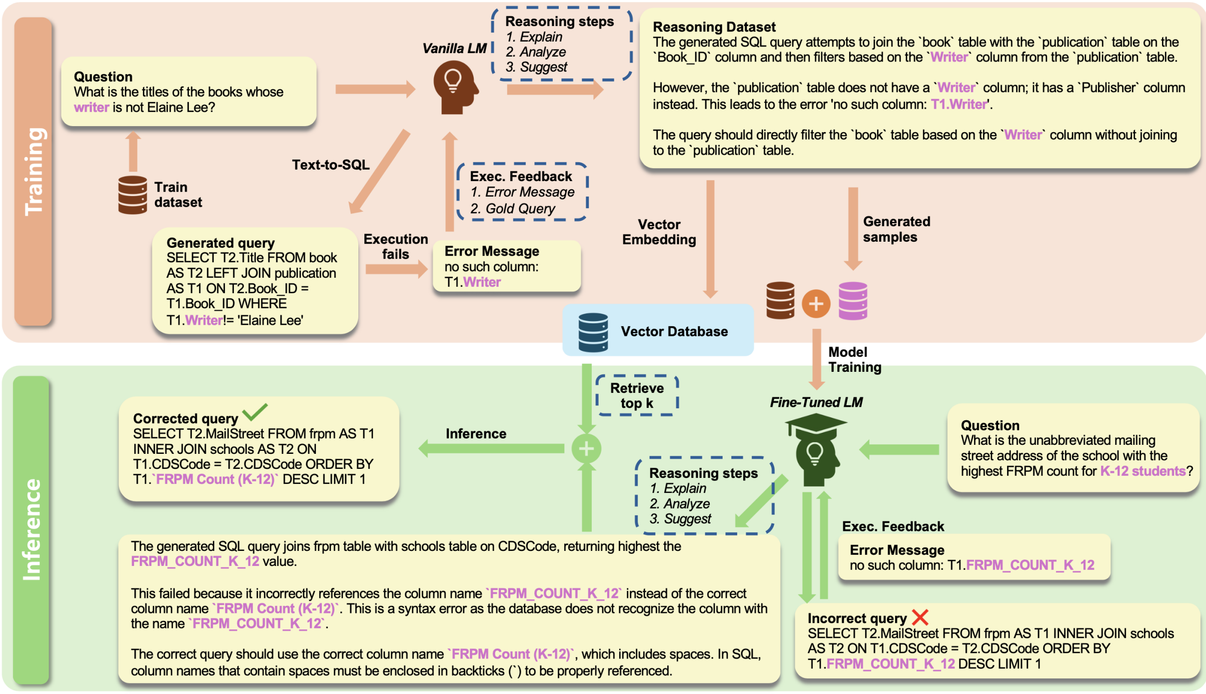

Figure 1: Overview of the ReSQL framework. The training stage (Top) begins with a vanilla language model (vLM) making inferences on a training dataset from either BIRD or SPIDER. If the model fails to execute a query correctly, the vLM generates a self-supervised reasoning dataset by following a structured error analysis process: (1) explaining the behavior of the incorrect query, (2) diagnosing the root cause of the error, and (3) suggesting a correction. This self-generated reasoning dataset is then incorporated into the original training set for model fine-tuning. In the inference stage (Bottom), when an execution error occurs, the fine-tuned model first generates a reasoning explanation. It then retrieves the top-3 relevant RAG examples, based on the question and execution error type, for few-shot setting. Using this additional context, the model generates the corrected SQL query, improving execution accuracy and robustness.

augmented generation (RAG)-based fine-tuning framework that enhances a model's reasoning ability to recognize and correct execution errors. By incorporating retrieval-based augmentation, ReSQL provides explicit error correction guidance, allowing models to systematically improve their understanding of execution failures and learn effective correction strategies. Through extensive experiments, we demonstrate that ReSQL-trained models consistently achieve significant performance gains on both the SPIDER and BIRD benchmarks, surpassing all state-of-the-art in-context learning methods. Notably, 7B–9B parameter models trained with ReSQL outperform GPT-4, which achieves execution accuracies of 79.5% on SPIDER and 46.35% on BIRD 13. Our results indicate that even smaller models, when equipped with effective error reasoning mechanisms, can achieve 40–60% higher performance compared to standard supervised fine-tuning (SFT), significantly improving their error correction capabilities across diverse query complexities and execution challenges.

In summary, ReSQL enhances Text-to-SQL model robustness by leveraging self-generated datasets for error reasoning, narrowing the gap between general-purpose language models and specialized task-specific models. Our approach offers a scalable solution for improving execution reliability, particularly benefiting smaller models prone to frequent execution errors in complex querying scenarios. To the best of our knowledge, ReSQL is the first approach to self-correct execution errors through direct fine-tuning, offering valuable insights for the Text-to-SQL research community.

2 Related Work

In recent years, LLMs have significantly enhanced the capabilities of Text-to-SQL systems by integrating self-correction mechanisms with various prompting methods, which we explore in depth.

Self-Correction in Text-to-SQL: There is growing interest in enabling models to self-correct, thereby improving the accuracy and reliability of SQL generation. Early efforts, such as self-consistency (Wang et al., 2022), rely on generating multiple candidate SQL queries and choosing the most consistent one through voting mechanisms. Further advances include self-debugging (Chen et al., 2023), where explanations for predicted SQL are generated and used to correct initial outputs. DIN-SQL (Pourreza and Rafiei, 2024) utilizes a human-written guideline to revise SQL queries based on common errors made by the model. MAGIC (Askari et al., 2024) extends this line of work by introducing an automatically generated self-correction guideline, which contrasts with DIN-SQL's human-crafted approach. This allows for more scalable and flexible error correction in SQL generation, independent of the specific in-context learning method or prompting strategy used.

Prompting and Retrieval-Augmented Techniques for Text-to-SQL: Recent Text-to-SQL systems leverage in-context learning and zero-shot prompting (Pourreza and Rafiei, 2024). Several models have advanced the field by incorporating techniques like Chain-of-Thought (CoT) (Zhang et al., 2023) and decompositional approaches such as DAIL-SQL (Gao et al., 2023) and MAC-SQL (Wang et al., 2023). Additionally, retrieval-augmented generation (RAG) has emerged as a promising approach to enhance in-context learning by dynamically retrieving relevant context from external sources. Thorpe et al. (2024) introduce Dubo-SQL, which employs a diverse RAG pipeline to improve execution accuracy in SQL generation by selecting informative few-shot examples rather than relying on simple nearest-neighbor retrieval. This method demonstrates that leveraging diverse retrieval improves SQL accuracy over static fine-tuned models while maintaining cost efficiency. Similarly, Ziletti and D'Ambrosi (2024) integrate RAG for epidemiological question answering over electronic health records (EHRs), showing significant performance gains when augmenting LLM-generated SQL queries with domain-specific retrieved examples.

Beyond prompt optimization, recent work has explored multi-staged prompting strategies to enhance LLM performance in SQL generation. Xiong et al. (2024) introduce a two-stage method leveraging schema-aware prompts and schema linking to generate more accurate SQL queries.

Reasoning for Text-to-SQL: Text-to-SQL parsing enables non-experts to query databases using natural language, but models often struggle with execution errors, particularly small language models. Recent methods enhance reasoning in various ways: CHASE-SQL (Pourreza et al., 2024) generates diverse SQL candidates using multi-path reasoning and selects the best query through pairwise ranking. SQL-CRAFT (Xia et al., 2024) refines SQL through an interactive correction loop and Python-enhanced reasoning. graph-SQL (Gong and Sun, 2024) encodes schema relationships using graph-based self-attention to improve query structure understanding. While these approaches enhance SQL generation and selection, they do not incorporate a reasoning process to identify and correct execution errors.

Early efforts have laid a solid foundation for Text-to-SQL, but incorporating self-correction techniques remains underdeveloped. While much of the previous research has concentrated on improving retrieval strategies for in-context learning and prompt-based SQL generation, our work makes a significant contribution by proposing direct fine-tuning on a self-correction dataset.

Model	SPIDER (%)	BIRD (%)
Llama-3.2 1B	**49.60**	**70.85**
Llama-3.2 3B	20.91	45.16
Llama-3.1 8B	7.07	28.69
Llama-3.3 70B	1.86	15.49
Qwen-2.5 1.5B	28.31	58.15
Qwen-2.5 3B	17.73	43.63
Qwen-2.5 7B	5.14	28.59
Qwen-2.5 32B	1.04	10.48
Mistral-v0.3 7B	18.03	45.85
Gemma-2 2B	<u>37.60</u>	<u>67.04</u>
Gemma-2 9B	8.41	27.49

Table 1: The percentage of execution errors used to construct the training dataset. The SPIDER dataset comprises a total of 7,000 training instances, while BIRD contains 8,556. For each dataset, the model with the highest error percentage is highlighted in bold, while the second highest is underlined.

3 ReSQL framework

ReSQL framework follows a structured process to generate a dataset from SPIDER and BIRD dataset, later used for execution error feedback focused on improving model robustness in handling execution errors. The components shown in Figure 1 are outlined below:

Method	Llama				Qwen				Mistral	Gemma		CodeS (Li et al., 2024a)		
	1B	**3B**	**8B**	**70B**	**1.5B**	**3B**	**7B**	**32B**	**7B**	**2B**	**9B**	**1B**	**3B**	**7B**
Baseline	3.78	22.75	38.27	46.28	13.82	24.45	44.39	50.85	28.23	13.36	37.16	17.21	23.01	26.66
Simple	13.56	26.66	43.74	50.85	17.28	25.10	48.50	55.74	41.59	18.51	40.42	22.36	28.94	36.11
Self-Correction Guideline (Askari et al., 2024)	9.84	25.55	45.89	51.76	15.51	26.53	49.74	57.04	43.61	15.45	43.22	21.71	29.60	38.07
Self-Debugging (Chen et al., 2023)	13.49	27.31	44.00	51.04	16.88	24.58	48.91	55.93	42.24	18.77	41.33	23.34	30.44	37.87
Self-Consistency (Wang et al., 2022)	14.34	26.86	44.46	51.56	18.25	25.81	49.15	56.98	42.37	19.23	40.29	23.08	30.25	38.92
ReSQL w/o RAG (ours)	23.79	42.83	48.24	51.69	26.73	38.33	52.74	56.39	47.26	25.95	**50.26**	33.96	40.68	51.04
ReSQL (ours)	**24.84**	**43.88**	**48.83**	**52.09**	**28.23**	**39.18**	**53.78**	**57.69**	**47.65**	**26.92**	49.74	**34.94**	**41.33**	**52.28**

Table 2: Execution accuracy (EX) on the BIRD-dev dataset. The reported scores represent execution accuracy after the second iteration of error correction. The models are assessed against two baselines: (1) the unmodified baseline (without fine-tuning) and (2) Simple, which is fine-tuned on BIRD training datasets. Our proposed method, ReSQL, and ReSQL without RAG are compared alongside other state-of-the-art approaches, including Self-Correction Guideline, Self-Debugging, and Self-Consistency. The best performance for each evaluation is highlighted in bold, while the second-best is underlined.

3.1 Generating self-reinforcing error reasoning dataset

The core component of our framework is the generation of an error reasoning dataset to enhance the model's reasoning capabilities. For each execution error encountered in the SPIDER and BIRD datasets, we provide the ground-truth SQL query and corresponding error message, enabling the model to analyze the incorrect query. The reasoning process consists of three key steps: (1) explaining the behavior of the incorrect SQL query, (2) identifying specific issues within it, and (3) suggesting corrections to produce the correct query.

Even with a model size of 1B parameters, leveraging gold queries and structured guidelines allows for accurate error analysis. We fine-tune the model using this reasoning data alongside the original training set of either BIRD or SPIDER, depending on the task. This approach not only strengthens generic Text-to-SQL capabilities but also enhances the model's reasoning ability, enabling it to identify and correct frequent errors through structured analysis. Rather than simple error correction, our method fosters a step-by-step reasoning process akin to chain-of-thought (CoT) reasoning (Liu et al., 2023a). This reasoning dataset combined with original train dataset are used for fine-tuning.

Recognizing the importance of data quality in reasoning-driven fine-tuning, we verify our reasoning dataset using G-Eval (Liu et al., 2023b), an LLM-based evaluator that assesses whether the generated reasoning correctly justifies the transformation from incorrect to correct SQL. Motivated by prior work such as LIMA (Zhou et al., 2024), which highlights the impact of small but high-quality fine-tuning data, we applied G-Eval to the reasoning dataset and observed high correctness rates, confirming the reliability of ReSQL as presented. Further details on the evaluation format and illustrative examples can be found in Appendix 8.

3.2 Retrieval-Augmented Text-to-SQL Generation for Error Correction

For each model, we maintain a distinct set of execution error samples derived from SPIDER and BIRD. During inference, we incorporate this set into a retrieval-augmented generation (RAG) framework. Specifically, when the model encounters an execution error, it retrieves the top three most similar error instances based on vector similarity between the given question and the error message. These retrieved samples serve as in-context learning examples, particularly aiding in handling underrepresented error types in the training dataset.

Since these error samples are already part of the training set, the fine-tuned model has previously encountered and learned from them. This raises two critical questions: (1) Does retrieving the same error samples during inference provide additional benefits, or is it redundant? (2) Does the model continue to struggle with errors it has already been trained on?

As shown in Table 5, training exclusively on error samples does not ensure perfect generalization, as rare or domain-specific errors are often underrepresented. In this context, retrieval acts as an external recall mechanism, enhancing our unified approach by exposing rare yet relevant patterns during inference. We demonstrate that ReSQL with RAG significantly reduces these error types, resulting in a more robust model.

Method	Llama				Qwen				Mistral	Gemma		CodeS (Li et al., 2024a)		
	1B	3B	8B	70B	1.5B	3B	7B	32B	7B	2B	9B	1B	3B	7B
Baseline	20.02	49.03	59.67	71.18	38.01	46.71	68.76	78.63	44.00	44.78	61.12	43.70	58.50	59.70
Simple	52.03	57.35	75.53	76.60	53.09	57.16	76.11	83.46	75.82	55.32	75.15	62.67	68.09	73.21
Self-Correction Guideline (Askari et al., 2024)	51.35	58.41	76.60	**78.82**	51.26	57.64	78.34	83.95	77.76	58.22	78.92	61.61	68.86	73.89
Self-Debugging (Chen et al., 2023)	52.42	57.74	76.60	77.37	54.35	58.32	79.69	<u>83.56</u>	77.66	56.29	77.95	63.15	68.09	73.50
Self-Consistency (Wang et al., 2022)	54.26	59.86	75.92	77.18	54.35	58.99	77.47	<u>83.56</u>	76.60	55.90	75.24	63.06	69.63	73.31
ReSQL w/o RAG (ours)	<u>61.25</u>	<u>68.41</u>	<u>77.48</u>	77.55	<u>64.39</u>	<u>68.25</u>	<u>80.95</u>	**84.14**	<u>76.95</u>	<u>60.95</u>	**81.32**	<u>67.99</u>	<u>72.63</u>	<u>77.57</u>
ReSQL (ours)	**62.34**	**69.25**	**77.58**	<u>77.85</u>	**66.29**	**70.36**	**81.29**	**84.14**	**78.53**	**62.02**	<u>80.51</u>	**68.86**	**73.11**	**78.53**

Table 3: Execution accuracy (EX) on the SPIDER-dev dataset. The best performance for each evaluation is highlighted in bold, while the second-best is underlined.

4 Experimental Setup

4.1 Models

We evaluate the impact of directly fine-tuning on a self-debugging dataset by extensively testing various sizes of well-known instruction-tuned models: Llama-3.2 (1B, 3B), Llama-3.1 8B, Llama-3.3 70B (Dubey et al., 2024), Qwen-2.5 (1.5B, 3B, 7B, 32B) (Yang et al., 2024), Mistral-v0.3 7B (Jiang et al., 2023), and Gemma2 (2B, 9B) (Team et al., 2024). We also add CodeS (Li et al., 2024a), the state-of-the-art model for Text-to-SQL. CodeS leverages a large-scale corpus of synthetic (NL, SQL) pairs to pretrain LLMs specifically for Text-to-SQL tasks, achieving strong generalization across diverse domains and complex SQL structures.

For benchmarking, we consider both baseline and fine-tuned models. The baseline consists of the instruction-tuned version of each model without additional supervised fine-tuning (SFT). In contrast, the simple fine-tuning approach involves training the models separately on two datasets, SPIDER and BIRD, treating each as an independent task.

To evaluate self-correction methods for handling SQL query errors during inference, we employ several state-of-the-art approaches. The MAGIC framework automates self-correction for text-to-SQL tasks using three specialized agents: Manager, Feedback, and Correction. The Feedback agent identifies SQL query errors, while the Correction agent revises them iteratively based on 34 predefined query correction guidelines. The Self-Debugging method enables large language models to iteratively debug their own generated SQL queries. It does so by executing the queries, generating natural language explanations, and using feedback to refine them—all without human intervention. Meanwhile, the Self-Consistency Model runs 10 inference iterations per input and selects the most frequent query through a voting mechanism. If execution errors occur, the model retries with an updated prompt, allowing up to two correction attempts.

4.2 Dataset

We evaluated the models on two distinct cross-domain datasets: SPIDER and BIRD. SPIDER consists of 10,181 questions paired with 5,693 unique SQL queries spanning 200 databases and 138 domains. The dataset is divided into 8,659 training examples and 1,034 development examples, and the SQL queries are categorized into four levels of difficulty (Easy, Medium, Hard, Extra Hard). The complex nature of SPIDER, due to its diverse schemas and queries, makes it an ideal dataset for benchmarking generalization in text-to-SQL tasks. BIRD contains 12,751 question-SQL pairs across 95 databases, covering over 37 professional domains, such as blockchain, healthcare, and education. In addition to SQL queries, the dataset incorporates four sources of external knowledge: numeric reasoning, domain-specific information, synonyms, and value illustrations. SQL queries in BIRD are generally more challenging than those in SPIDER and are classified into three difficulty levels (Simple, Medium, Challenging). To aid in schema linking, we provided sample rows from the database tables, as well as external knowledge, as hints. Each model is trained on a distinct set of instances, derived from their execution errors on train set. The percentages of the execution errors for all models are shown in Table 1. Here, smaller models creates significantly more execution errors compared to larger sized models. During inference, these collected execution errors, paired with their corresponding reasoning data, are utilized for RAG.

4.3 Metric

The model performance is evaluated using execution accuracy (EX) and SQL query correction rate (CR), which offer a more nuanced assessment than traditional exact match metrics. Execution accuracy compares the execution results of generated SQL queries with the ground truth, reflecting the flexibility in writing correct queries in text-to-SQL tasks. Performance is reported after up to two correction iterations, as further iterations provide diminishing correction (See Figure 3). The CR measures the proportion of errors successfully corrected, highlighting a model's self-correction ability. This metric is crucial, as models with fewer initial errors have fewer chances to improve through error correction. CR is the ratio of successful corrections to total errors. This measure provides a straightforward percentage of errors that are fixed by the model.

5 Result

5.1 Text-to-SQL self-correction benchmark

Tables 2 and 3 present the execution accuracy (EX) of various models on the BIRD-dev and SPIDER-dev datasets, respectively. Across all model scales, our proposed method, ReSQL, consistently outperforms both baseline methods and other state-of-the-art techniques, demonstrating its efficacy in reducing execution errors. Notably, the performance gap between ReSQL and prior approaches is particularly pronounced in smaller models (e.g., Llama-1B, Qwen-1.5B, Gemma-2B, CodeS-1B), where the presence of execution errors is more significant. This highlights the ability of ReSQL to refine SQL generation effectively, even in models with limited capacity. For example, in BIRD-dev, ReSQL enhances Llama-1B's accuracy from 3.78% (Baseline) to 24.84%, a nearly sevenfold improvement, significantly outperforming Self-Consistency (14.34%). Similarly, ReSQL improves Gemma-2B from 13.36% to 26.92%, and CodeS-1B, a pretrained Text-to-SQL model, from 17.21% to 34.94%, demonstrating that ReSQL enhances performance even on models already specialized for SQL generation, whereas other methods struggle to provide such a robust correction.

For larger models, where execution accuracy is inherently higher, ReSQL still demonstrates meaningful improvements, emphasizing the importance of fine-grained error correction even in high-capacity models. For instance, in SPIDER-dev, even with few execution errors (See Figure 2), Llama-70B sees a 1.25% boost from Simple fine-tuning (76.60%) to ReSQL (77.85%), while Qwen-32B reaches 84.14% with ReSQL, surpassing all prior approaches. Furthermore, the comparison between ReSQL and ReSQL w/o RAG highlights the effectiveness of retrieval-augmented generation (RAG) in refining SQL generation, particularly for rare or complex queries. This is evident in models like Mistral-7B, where ReSQL improves execution accuracy from 76.95% to 78.53% on SPIDER-dev. Notably, Qwen-3B shows the largest RAG-driven gain, improving from 68.25% to 70.36%, showcasing the benefits of incorporating retrieval-based corrections.

Overall, these results affirm that ReSQL provides a robust, scalable, and generalizable error-correction framework across varying model sizes, establishing a new benchmark for Text-to-SQL generation.

5.2 Error correction result

	SPIDER					**BIRD**			
	Easy	Medium	Hard	Extra	**Avg**	Easy	Medium	Hard	**Avg**
Simple	55.56	17.65	42.38	4.83	26.78	12.75	12.14	8.67	10.45
ReSQL	**88.89**	**68.63**	**70.36**	**39.32**	**64.38**	**38.55**	**28.23**	**27.96**	**30.56**

Table 4: Comparison of Correction Rates (%) Between the Simple SFT and ReSQL Framework. The baseline model used for evaluation is Llama-3.1 8B. The SPIDER dataset is categorized into four difficulty levels: Easy, Medium, Hard, and Extra, while the BIRD dataset comprises three levels: Simple, Moderate, and Challenging. The reported values represent the average CR across all difficulty levels.

Error types		ReSQL		
	Simple	ReSQL(*w/o RAG*)	ReSQL(bottom-3)	ReSQL(top-3)
Gold Error	59.32	42.37	43.02	**42.11**
No such column	15.97	6.13	6.98	**6.00**
No such function	2.87	**1.56**	1.83	1.63
No such table	2.09	**1.17**	1.50	1.30
Ambiguous column name	2.35	0.52	0.52	**0.33**
Syntax error	5.48	**2.35**	2.74	2.54
Unrecognized token	1.56	0.91	0.91	**0.85**
More than one statement	0.85	0.33	0.33	**0.20**
Incomplete input	1.63	0.26	0.26	**0.20**
Misuse of aggregate function	0.46	0.46	0.46	**0.40**
Misuse of window function	0.39	0.33	0.33	**0.26**
Wrong number of arguments	0.20	0.20	0.20	**0.13**
Aggregate with GROUP BY	0.26	0.20	0.20	**0.00**
ORDER BY before UNION ALL	0.33	0.26	0.26	**0.07**
1st ORDER BY does not match	0.20	**0.13**	**0.13**	**0.13**
Incorrect prop.	77.25	57.17	59.67	**56.12**

Table 5: Error type analysis of incorrect SQL queries on BIRD-dev, showing the proportion (%) of each error category across different methods: Simple (no ReSQL), ReSQL without RAG, ReSQL with bottom-3 retrieval, and ReSQL with top-3 retrieval. The Llama-3.2 3B model serves as the baseline. Lower values indicate better performance. The best result for each error type is highlighted in bold, while the second-best is underlined.

We evaluate the effectiveness of the ReSQL framework in reducing execution errors and im-

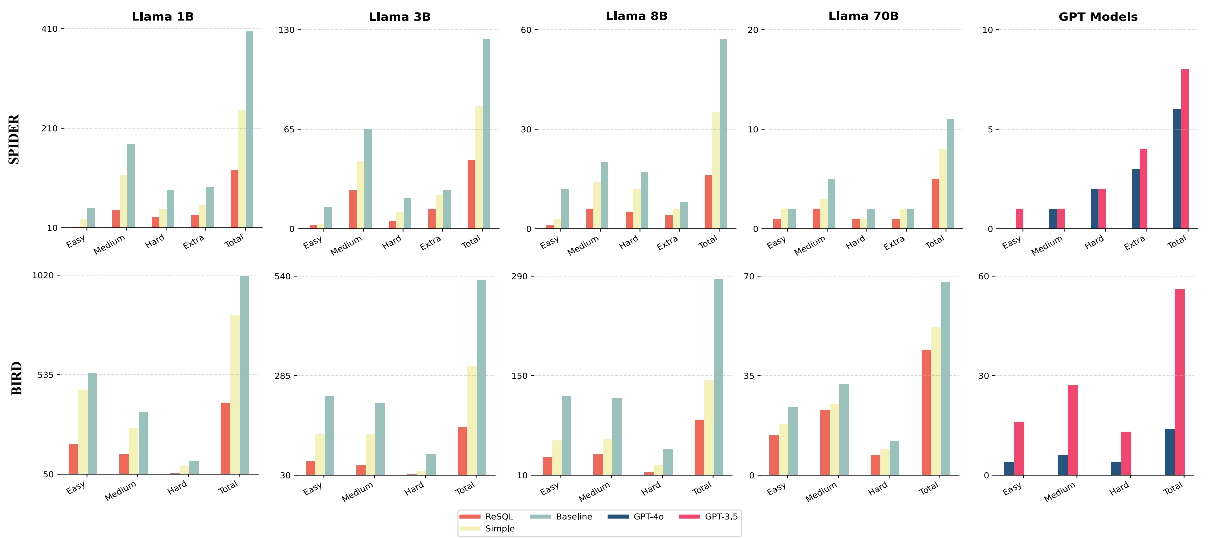

Figure 2: Number of execution errors remaining across difficulties on SPIDER and BIRD after second iteration of correction. The instruct versions of the open-source Llama family (1B, 3B, 8B, 70B) are evaluated. The comparison includes three model variants: (1) Baseline (no SFT), (2) Simple SFT (fine-tuned on the respective training dataset), and (3) the ReSQL framework. For reference, GPT models are presented separately on the right.

proving correction rates across two benchmark datasets: SPIDER and BIRD. Table 4 highlights ReSQL's superior error correction across all difficulty levels on SPIDER and BIRD. On SPIDER, ReSQL achieves an average CR of 64.38%, more than doubling Simple SFT (26.78%), with notable gains in medium (68.63% vs. 17.65%) and hard (70.36% vs. 42.38%) queries. Even for extra-hard cases, ReSQL significantly outperforms Simple SFT (39.32% vs. 4.83%), showcasing its robustness in handling complex queries. Similarly, on BIRD, ReSQL attains 30.56% CR, nearly three times that of Simple SFT (10.45%), with the largest improvement in simple queries (38.55% vs. 12.75%). These results validate ReSQL's scalability and effectiveness in refining Text-to-SQL generation.

Figure 2 demonstrates that across all Llama model sizes, the ReSQL framework consistently reduces execution errors compared to the Baseline and Simple SFT variants. This pattern is evident in both SPIDER and BIRD datasets, regardless of difficulty level. The error reduction effect is particularly pronounced in larger models (Llama 8B and 70B), suggesting that ReSQL effectively leverages increased model capacity to minimize execution failures. The total number of errors decreases markedly across all Llama model variants, with ReSQL leading to the lowest error count. In BIRD, a similar trend is observed, with ReSQL substantially lowering execution errors, particularly in the Medium and Hard categories. The total error count remains higher than SPIDER, indicating the dataset's increased difficulty.

These results collectively indicate that ReSQL is highly effective in both reducing execution errors and improving correction rates across all model sizes and difficulty levels. The consistent improvements across multiple datasets and various model sizes underscore its robustness. Notably, ReSQL's impact scales with larger model sizes, which suggests that future work could explore further optimization strategies to maximize performance on extreme difficulty levels.

	SPIDER		**BIRD**	
	EX	ΔEX	EX	ΔEX
All tools	69.25	–	43.88	–
w/o error feedback	55.71	-13.54	35.66	-8.22
w/o error reasoning	58.22	-11.03	26.27	-17.61
w/o RAG	68.41	-0.84	42.83	-1.05
with 1-time revise	65.86	-3.39	40.10	-3.78

Table 6: Ablation study results on the SPIDER and BIRD datasets using Llama-3.2 3B as the baseline model. Error feedback refers to utilizing training data corrections from each dataset. 1-time revise denotes a single correction pass without further iterative refinement. The results demonstrate the impact of removing specific components on execution accuracy (EX) and its relative change (ΔEX).

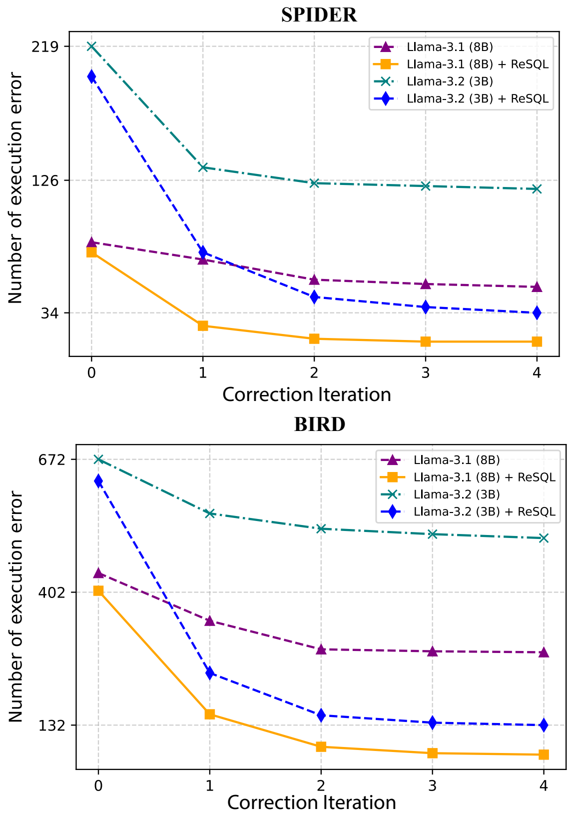

Figure 3: Execution error reduction across correction iterations on SPIDER and BIRD datasets. Comparison of LLaMA-3.1 (8B) and LLaMA-3.2 (3B) models with and without ReSQL framework.

5.3 Evaluating RAG in SQL Error Correction

Table 5 compares SQL error types across Simple, ReSQL w/o RAG (top-3 and bottom-3), and ReSQL using the Llama-3.2 3B model. Lower values indicate better performance. Overall, ReSQL with RAG top-3 achieves the lowest incorrect proportion (56.12%), demonstrating the effectiveness of RAG in reducing SQL errors.

For Gold Error, ReSQL (42.11%) marginally outperforms ReSQL w/o RAG (42.37%) but significantly improves over Simple (59.32%). In schema-related errors (No such column, No such table), ReSQL performs comparably to ReSQL w/o RAG but significantly reduces errors from Simple, suggesting that retrieval aids schema reasoning but does not fully resolve it. For structure errors (More than one statement, ORDER BY issues), ReSQL consistently achieves lower error rates, showing that retrieval improves query formulation. Notably, for rare errors (Misuse of window function, Wrong number of arguments), ReSQL performs best, indicating that RAG is particularly effective in handling low-frequency error cases, likely because these errors are underrepresented in training data.

5.4 Ablation study

Table 6 presents the ablation study results on the SPIDER and BIRD datasets using llama-3.2 3B as the baseline. The most significant performance drop occurs when error reasoning is removed, particularly on BIRD, where execution accuracy (EX) drops by 17.61%. This highlights the critical role of reasoning in handling complex queries, especially in less structured datasets like BIRD. Similarly, removing error feedback leads to a substantial decline (-13.54% on SPIDER, -8.22% on BIRD), demonstrating that leveraging training data corrections is essential for improving model predictions. The removal of RAG has a smaller effect, suggesting that the model can rely on internal representations in most cases. The 1-time revise setting improves results compared to ablated versions but remains inferior to the full system, reinforcing the importance of iterative refinement. Overall, these results underscore that both explicit reasoning and feedback from the original training dataset are crucial for maximizing execution accuracy.

6 Conclusions

In this paper, we introduce ReSQL, a retrieval-augmented error reasoning framework for Text-to-SQL models. ReSQL enhances self-debugging capabilities by fine-tuning models on a self-generated error reasoning dataset and incorporating retrieval-augmented generation to improve execution accuracy. The framework systematically identifies, analyzes, and corrects execution errors, addressing a key challenge in Text-to-SQL generation.

Experimental results on SPIDER and BIRD benchmarks show that ReSQL significantly improves execution accuracy and error correction rates, outperforming existing self-correction and prompting-based methods. Notably, ReSQL enables 1–3B parameter models to achieve substantial accuracy gains, reducing execution errors and narrowing the performance gap with larger models. Ablation studies confirm that explicit error reasoning is essential for self-correction, while RAG further enhances robustness, particularly for rare error types. We believe ReSQL will provide a scalable and generalizable approach to improving Text-to-SQL models, demonstrating its effectiveness across different model sizes and query complexities.

7 Limitations

ReSQL demonstrates significant improvements in execution accuracy and error correction for Text-to-SQL tasks, but several limitations remain.

The framework primarily focuses on post-execution correction and does not proactively prevent errors before query execution. A proactive reasoning mechanism could further reduce the need for iterative debugging. Fine-tuning large models also requires substantial computational resources, with models like Llama-3.1 8B requiring at least two A100 40GB GPUs, making widespread adoption challenging. Furthermore, we did not evaluate ReSQL in closed or proprietary models such as DeepSeek-R1 due to limited accessibility and resource constraints. This remains an area for future exploration. While RAG helps resolve less frequent execution errors, its effectiveness is limited for rare SQL errors that were underrepresented in training data. Data augmentation and dynamic retrieval strategies may help further improve error resolution, especially for rare SQL patterns.

Lastly, we acknowledge the limitations of execution accuracy (EX), our primary evaluation metric. While EX, implemented as test suite accuracy in SPIDER and BIRD, can detect semantically equivalent but syntactically different queries, it may yield false positives when distinct queries return the same result on a limited test suite, and false negatives due to annotation noise. Despite these shortcomings, we adopt EX for comparability with prior work, while recognizing the value of complementary, semantically-aware metrics for more rigorous evaluation.

References

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

Shuaichen Chang, Pengfei Liu, Yun Tang, Jing Huang, Xiaodong He, and Bowen Zhou. 2020. Zero-shot text-to-sql learning with auxiliary task. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7488–7495.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*.

Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2020. Structure-grounded pretraining for text-to-sql. *arXiv preprint arXiv:2010.12773*.

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

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

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*.

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

Zheng Gong and Ying Sun. 2024. Graph reasoning enhanced language models for text-to-sql. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2447–2451.

Zihui Gu, Ju Fan, Nan Tang, Lei Cao, Bowen Jia, Sam Madden, and Xiaoyong Du. 2023. Few-shot text-to-sql translation using structure and content prompt learning. *Proceedings of the ACM on Management of Data*, 1(2):1–28.

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. *arXiv preprint arXiv:1905.08205*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

George Katsogiannis-Meimarakis and Georgia Koutrika. 2021. A deep dive into deep learning approaches for text-to-sql systems. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2846–2851.

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

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. *Proceedings of the ACM on Management of Data*, 2(3):1–28.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 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.

Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2020. Bridging textual and tabular data for cross-domain text-to-sql semantic parsing. *arXiv preprint arXiv:2012.12627*.

Max Liu, Nathan Liu, Zuohui Fu, and Jason Liu. 2023a. Chain-of-thought reasoning without prompting. *arXiv preprint arXiv:2306.08739*.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023b. G-eval: Nlg evaluation using gpt-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2511–2522.

Rosa Meo, Giuseppe Psaila, Stefano Ceri, et al. 1996. A new sql-like operator for mining association rules. In *VLDB*, volume 96, pages 122–133. Citeseer.

Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates. 2004. Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, pages 141–147.

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

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

Jiexing Qi, Jingyao Tang, Ziwei He, Xiangpeng Wan, Yu Cheng, Chenghu Zhou, Xinbing Wang, Quanshi Zhang, and Zhouhan Lin. 2022. Rasat: Integrating relational structures into pretrained seq2seq model for text-to-sql. *arXiv preprint arXiv:2205.06983*.

Daking Rai, Bailin Wang, Yilun Zhou, and Ziyu Yao. 2023. Improving generalization in language model-based text-to-sql semantic parsing: Two simple semantic boundary-based techniques. *arXiv preprint arXiv:2305.17378*.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. 2024. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*.

Dayton G Thorpe, Andrew J Duberstein, and Ian A Kinsey. 2024. Dubo-sql: Diverse retrieval-augmented generation and fine tuning for text-to-sql. *arXiv preprint arXiv:2404.12560*.

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

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Hanchen Xia, Feng Jiang, Naihao Deng, Cunxiang Wang, Guojiang Zhao, Rada Mihalcea, and Yue Zhang. 2024. Sql-craft: Text-to-sql through interactive refinement and enhanced reasoning. *arXiv preprint arXiv:2402.14851*.

Guanming Xiong, Junwei Bao, Hongfei Jiang, Yang Song, and Wen Zhao. 2024. Interactive-t2s: Multi-turn interactions for text-to-sql with large language models. *arXiv preprint arXiv:2408.11062*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. 2024. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

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

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

Victor Zhong, Mike Lewis, Sida I Wang, and Luke Zettlemoyer. 2020. Grounded adaptation for zero-shot executable semantic parsing. *arXiv preprint arXiv:2009.07396*.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. 2024. Lima: Less is more for alignment. In *Advances in Neural Information Processing Systems*, volume 36.

Angelo Ziletti and Leonardo D'Ambrosi. 2024. Retrieval augmented text-to-sql generation for epidemiological question answering using electronic health records. *arXiv preprint arXiv:2403.09226*.

A Appendix

Model	Init	C1	C2	C3	C4
LLaMA-3.1 (8B)	83	71	57	54	52
LLaMA-3.1 (8B) + ReSQL	76	25	16	14	14
LLaMA-3.2 (3B)	219	135	124	122	120
LLaMA-3.2 (3B) + ReSQL	198	76	45	38	34

Table 7: Execution errors across Llama-3.1 8B and Llama-3.2 3B models on the SPIDER dataset. 'Init' represents the initial errors, while 'C1' to 'C4' indicate subsequent correction steps aimed at reducing these errors.

Model	Init	C1	C2	C3	C4
LLaMA-3.1 (8B)	441	344	286	282	280
LLaMA-3.1 (8B) + ReSQL	405	154	88	75	72
LLaMA-3.2 (3B)	672	562	531	520	512
LLaMA-3.2 (3B) + ReSQL	628	238	152	137	132

Table 8: Execution errors across Llama-3.1 8B and Llama-3.2 3B models on the BIRD dataset.

SPIDER	Error message	Number of errors
Syntax Errors		
	Syntax error	45
	Unrecognized token	11
	ORDER BY clause should come after UNION	1
Reference Errors		
	No such column	542
	No such function	30
	No such table	7
	Ambiguous column name	11
Function Misuse Errors		
	Misuse of aggregate function	11
	Aggregate functions are not allowed in the GROUP BY clause	2
Execution Errors		
	Timeout	11
	Incorrect number of bindings supplied	1

Table 9: Summary of BIRD SQL Execution Errors and Their Frequencies for Llama-3.2 3B.

All Error Type	Error message	Number of errors
Syntax Errors		
	Syntax error	2
	Unrecognized token	1
	Sub-select returns 2 columns	1
Reference Errors		
	No such column	199
	No such function	2
	No such table	10
	Ambiguous column name	2
Encoding and Format Errors		
	Could not decode to UTF-8	1
	Row value misused	1

Table 10: Summary of SPIDER SQL Execution Errors and Their Frequencies for Llama-3.2 3B.

Model	Avg. Reasoning Tokens
Qwen-2.5 1.5B	100.37
Qwen-2.5 3B	99.27
Qwen-2.5 7B	96.37
Llama-3.2 1B	105.37
Llama-3.2 3B	100.39
Llama-3.1 8B	98.68

Table 11: Average number of reasoning tokens generated during ReSQL inference across models. Short reasoning outputs (100 tokens) indicate minimal generation overhead, supporting ReSQL's practical applicability. Results are shown for representative instruct-tuned models (LLaMA: 1B, 3B, 8B; Qwen: 1.5B, 3B, 7B).

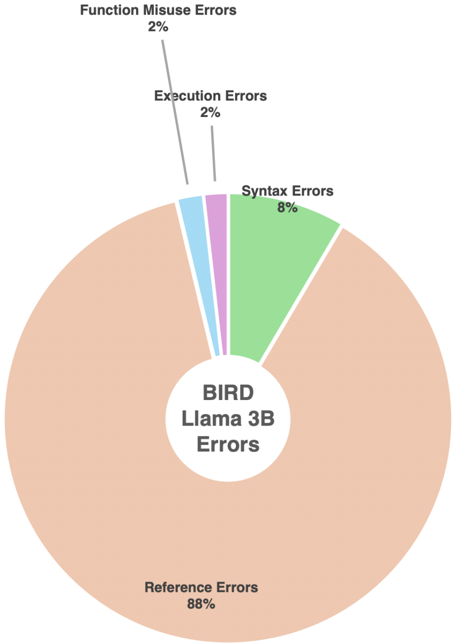

Figure 4: Donut chart representing the distribution of BIRD SQL Execution errors for Llama-3.2 3B, categorized into reference, syntax, execution and function misuse error.

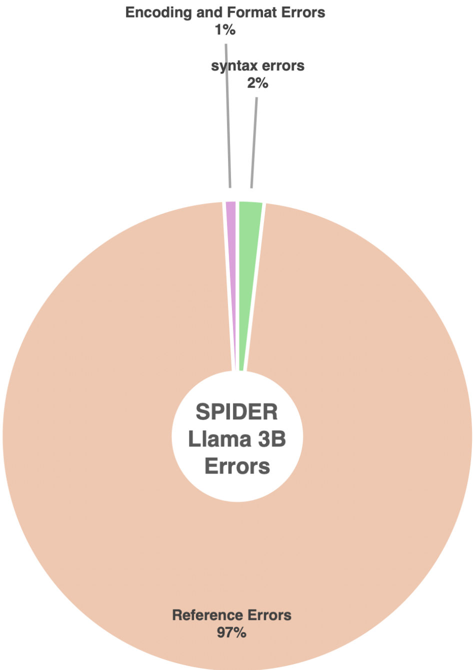

Figure 5: Donut chart representing the distribution of SPIDER SQL Execution errors for Llama-3.2 3B, categorized into reference, syntax, encoding and format error.

Dataset / Method	LLaMA-3.2 1B	LLaMA-3.2 3B	LLaMA-3.1 8B
BIRD-dev			
ReSQL	24.84	43.88	48.83
ReSQL w/ SPIDER Reasoning	24.38	40.55	47.07
SPIDER-dev			
ReSQL	62.34	69.25	77.58
ReSQL w/ BIRD Reasoning	61.12	68.86	77.47

Table 12: Cross-domain transferability of ReSQL reasoning data. Performance on BIRD and SPIDER when reasoning supervision is sourced from the same domain or transferred from the other. Results show that ReSQL remains effective with cross-domain reasoning, though in-domain supervision performs best. Results are shown for representative instruct-tuned models (LLaMA: 1B, 3B, 8B).

Dataset	**Easy / Simple**	**Medium / Moderate**	**Hard / Challenging**	**Extra**	**Overall**
BIRD	56.43	35.34	17.24	–	46.35
SPIDER	89.92	86.32	75.86	49.40	79.50

Table 13: GPT-4 performance on BIRD-dev and SPIDER-dev dataset by difficulty. '–' denotes not applicable.

[System prompt]
You are SQL query master. Only return predicted SQL query.

[User prompt]
Return SQLite query that answers the question given the table info. Provided row values are the first three rows of the table.

Context
Name: {table_0_name}
Info: {table_0_cols}
Rows: {table_0_first_three_rows}

⋮

Name: {table_N_name}
Info: {table_N_cols}
Rows: {table_N_first_three_rows}

Primary keys: {db_primary_key}
Foreign keys: {db_foreign_key}

Hint: {evidence} *// Only apply to BIRD dataset*

Output
SQL query:

Figure 6: Template for generating SQL queries from table information, showing the prompt structure, including context formatting and expected output.

[System prompt]
You are an SQL query master, a knowledgeable assistant for writing SQLite queries.

[User prompt]
Task: Your task is to analyze 'why the generated SQL query failed' and provide an explanation of the error. Generate in 3 steps following below format:
```
1. Explain:[Explain the behavior of the incorrect query]
2. Analyze:[analyze the root cause of the error]
3. Suggest:[Suggest the correction]
```

You are given:
- A 'Question' that needs to be answered using SQL.
- A 'Database information' that describes the tables and columns.
- A 'Gold SQL Query'that correctly answers the question.
- A 'Genearted SQL query' that was produced by the model but resulted in an execution error.
- The 'Error message' that was returned when executing the generated query.
Your task is to analyze 'why the generated SQL query failed' and provide an explanation of the error.

Guidelines for Analysis:
1. **Identify the Error Type**
- Syntax Error: Issues like incorrect SQL syntax or missing keywords.
- Semantic Error: The query structure is valid but references nonexistent tables/columns.
- Logical Error: The query does not match the intended question meaning.

2. **Compare Against the Gold Query**
- Identify key differences between the 'generated query' and 'gold query'.
- Explain which specific mistakes led to the execution error.

Response Format (JSON)
```json

"Reasoning": "<Your generated analysis here>",
"Error Type": "<Syntax Error / Semantic Error / Logical Error>"

```

Context
Question: {generated_question}
Hint: {evidence} *// Only apply to BIRD dataset*

Name: {table_0_name}
Info: {table_0_cols}
Rows: {table_0_first_three_rows}
⋮
Name: {table_N_name}
Info: {table_N_cols}
Rows: {table_N_first_three_rows}

Primary keys: {db_primary_key}
Foreign keys: {db_foreign_key}

Gold SQL Query: {gold_query}
Wrong SQL: {prediction_query}
Execution error: {execution_error_message}

Output
Reasoning:
Error Type:

Figure 7: Template for Analyzing Execution Errors in Generated SQL Queries.

[System prompt]
You are an expert in evaluating SQLite queries.

[User prompt]
Given the following inputs:
- A database schema
- A natural language question
- Supporting evidence
- An incorrectly generated SQL query
- The correct (ground truth) SQL query
- A proposed set of reasoning steps used to correct the incorrect query

Your task is to evaluate whether the provided reasoning steps correctly justify the transformation from the incorrect query to the correct one.

Return:
- 1 if the reasoning steps are valid and logically sound
- 0 if the reasoning steps are flawed or do not adequately justify the correction

Figure 8: Template for Assessing the Validity of SQL Reasoning Dataset Using G-Eval.

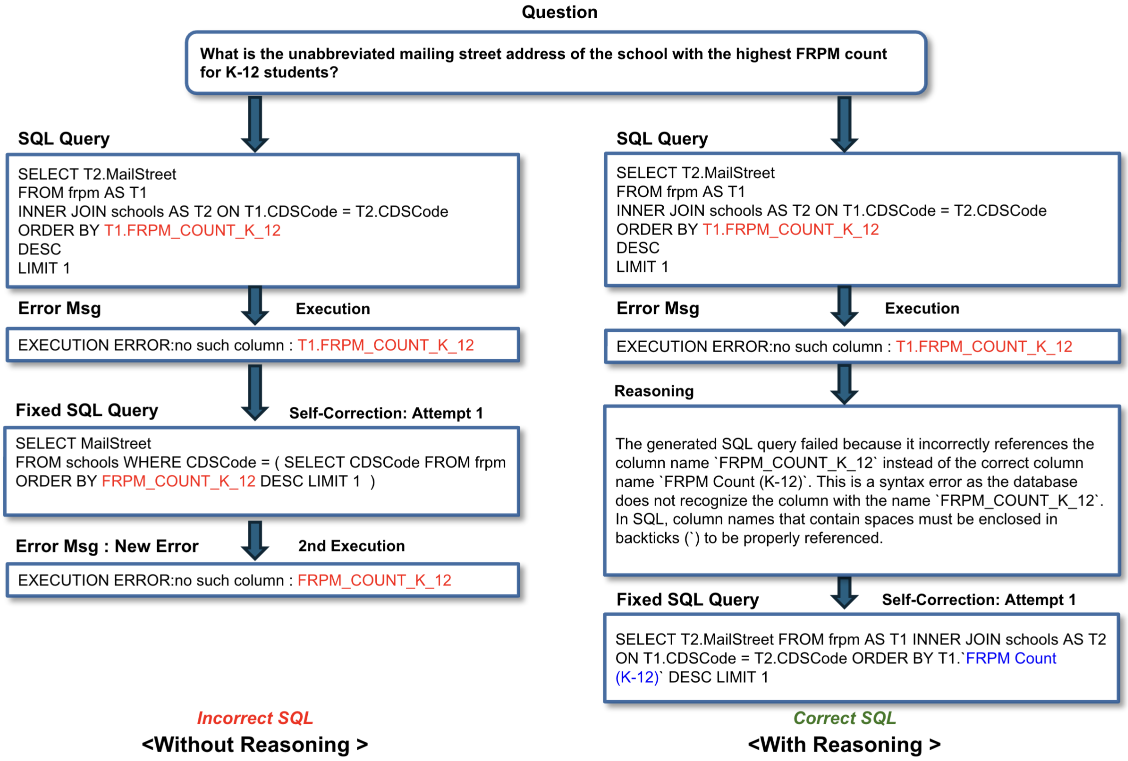

Figure 9: Comparison of SQL self-correction with and without reasoning. The model without reasoning fails to correctly self-correct the initial SQL query, generating another incorrect query even after attempting self-correction. In contrast, our model (with reasoning) identifies the root cause of the error, correctly fixes the query, and ensures execution accuracy.

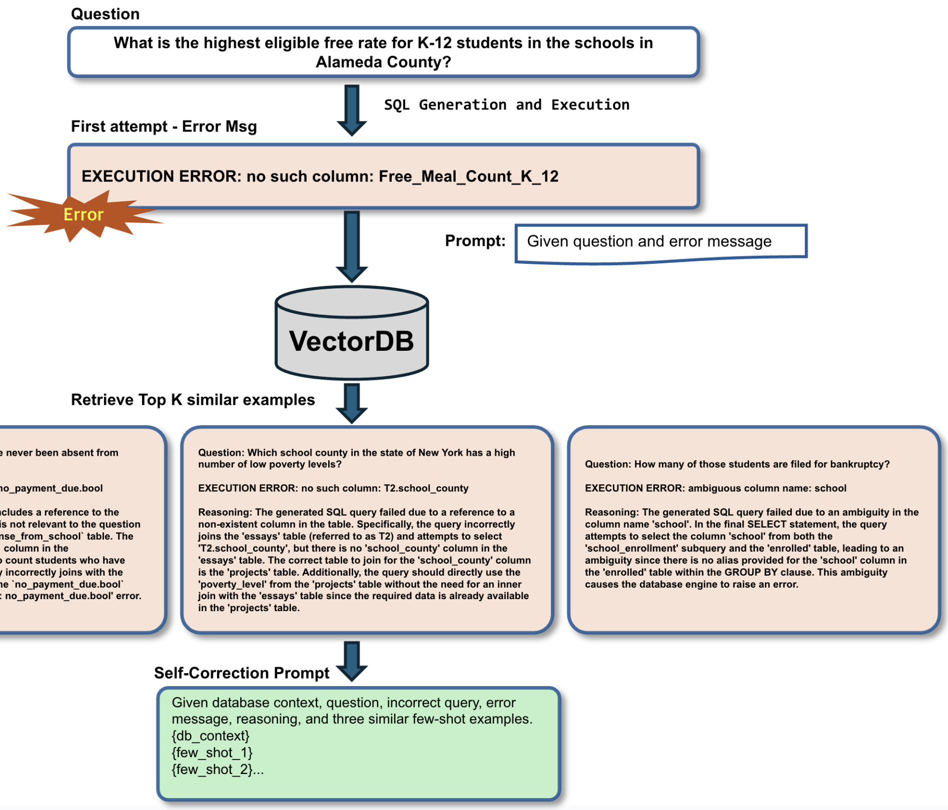

Figure 10: Overview of the Retrieval-Augmented Generation (RAG) framework for SQL error correction. When an execution error occurs, the system retrieves the top three most similar error cases from a database of past execution errors using vector similarity. These retrieved examples serve as in-context learning references, helping the model resolve underrepresented error types and improve robustness.