ReSQL: Self-Improving Framework for Reasoning-Aware Text-to-SQL Dataset Generation

Minjun Park*, **Yongju Seong***
Myoseop Sim, **Kyungkoo Min**, **Stanley Jungkyu Choi**
LG AI Research
{minjun.park, yongju.seong}@lgresearch.ai

Abstract

Recent advances in Text-to-SQL have greatly benefited from large language models, yet small and medium-sized models still suffer from frequent execution errors and limited self-correction ability. We present ReSQL (**R**etrieval-augmented **e**rror reasoning for Text-to-**SQL**), a self-improving framework that generates and learns from its own error-reasoning dataset, enabling models to autonomously refine their SQL generation and correction capabilities. ReSQL combines feedback-driven fine-tuning with retrieval-based inference: it gathers model-generated errors, analyzes them through structured feedback prompts, and retrieves relevant correction examples during inference. This unified approach allows models to internalize robust error-reasoning patterns and dynamically apply them to unseen queries. Experimental results on the SPIDER and BIRD benchmarks show that ReSQL substantially improves execution accuracy and self-correction ability over strong baselines, achieving competitive performance with much larger proprietary models such as GPT-4. Our findings highlight ReSQL as a promising step toward self-improving, reasoning-aware Text-to-SQL systems that can continually enhance their reliability and interpretability without external supervision. All code and generated reasoning datasets are available to facilitate application to open-source LLMs and reproducible baseline training.

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 (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 such as SPIDER (Yu et al., 2018) and BIRD (Li et al., 2024b) reveal these limitations by exposing models to execution failures that often stem from deeper reasoning deficiencies. These benchmarks highlight the persistent gap between syntactic accuracy and true semantic understanding, emphasizing the need for mechanisms that can identify, explain, and systematically correct such errors. Execution error rates in current Text-to-SQL systems remain high-reaching up to 70.85% on challenging benchmarks (See Table 1).

To address these persistent execution errors, we propose ReSQL, a novel framework designed to enhance step-by-step error diagnosis and correction through explicit reasoning. ReSQL constructs a self-generated dataset to systematically expose models to diverse execution errors, enabling structured learning of error detection and repair strate-

*These authors contributed equally.

Findings of the Association for Computational Linguistics: ACL 2026, pages 33582–33602

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
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
Gemma-2 2B	37.60	67.04
Gemma-2 9B	8.41	27.49
Mistral-v0.3 7B	18.03	45.85

Table 1: Percentage of execution errors in SQL queries generated by each model, used for constructing the training dataset. The SPIDER dataset contains 7,000 training instances, while BIRD contains 8,556.

gies. By integrating this correction knowledge directly into training, ReSQL improves both the robustness and reliability of Text-to-SQL models, particularly for smaller architectures that struggle with complex, real-world database queries. In contrast to prior methods that primarily rely on inference-time reranking (Wang et al., 2022) or prompting techniques (Gao et al., 2024), ReSQL embeds structured error reasoning into the model's training loop, allowing it to internalize correction strategies and generalize more effectively.

Additionally, we incorporate a retrieval-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, significantly reducing rare execution errors. 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 79.35% on SPIDER and 46.35% on BIRD. Our results indicate that even smaller models, when equipped with effective error reasoning mechanisms, can achieve up to 83% higher performance (Llama 1B) 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. While SQL execution feedback and RAG have been separately studied (Pourreza and Rafiei, 2023; Ziletti and D'Ambrosi, 2024), their joint integration remains underexplored. Our work reframes this integration as a form of externalized metacognition, where retrieval provides episodic memory over past reasoning failures and feedback fine-tuning reinforces procedural reasoning. This view bridges previously independent research threads in error-driven learning and retrieval-augmented reasoning. Our contribution is not a single new prompting trick or retrieval component in isolation, but a closed-loop framework that systematically connects three stages: automated execution-error capture, structured reasoning synthesis, and retrieval-augmented correction. The novelty of ReSQL lies in orchestrating these components into a reusable training-and-inference pipeline that converts execution failures into supervision signals and then reuses them at inference time for recovery.

2 Related Work

Large language models (LLMs) have advanced Text-to-SQL systems, notably through self-correction mechanisms and prompting-based approaches.

Self-Correction in Text-to-SQL: Prior work explores enabling models to self-correct generated SQL queries, often within in-context learning frameworks, to improve execution accuracy. Early approaches such as self-consistency (Wang et al., 2022) select among multiple candidate queries via voting, while subsequent methods introduce explicit correction signals. For example, self-debugging (Chen et al., 2023) revises erroneous outputs using model-generated explanations, DIN-SQL (Pourreza and Rafiei, 2023) applies human-written guidelines to correct common error patterns, and MAGIC (Askari et al., 2025) automates this process by generating correction guidelines.

Unlike prior approaches such as Self-Debugging or MAGIC, which primarily rely on inference-time prompting, revision loops, or externally supplied correction guidance, ReSQL transfers error-correction behavior into the model parameters through fine-tuning. This distinction is especially important for small and medium-sized models,

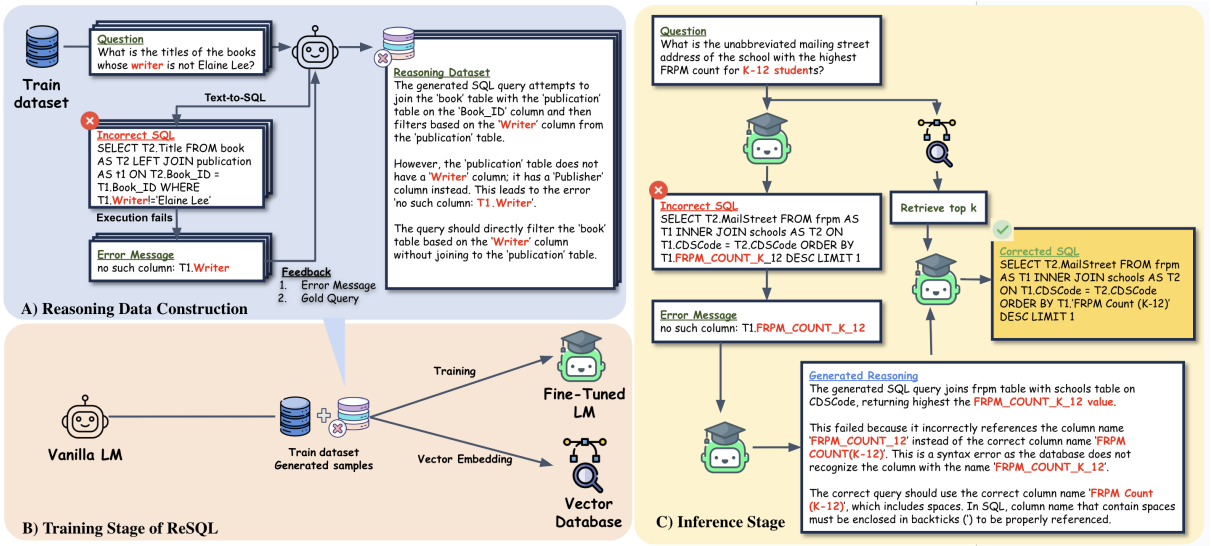

Figure 1: Overview of the ReSQL framework. Figure (A) illustrates how the base language model (LM) generates a self-supervised reasoning dataset from the training data via a structured error analysis process consisting of three stages: (1) explaining the incorrect query, (2) debugging the error, and (3) suggesting a correction. This self-generated reasoning dataset is then combined with the original training data for fine-tuning, as shown in Figure (B). In Figure (C), the reasoning dataset is vectorized and used for retrieval. Based on the input question and the type of execution error, the top-3 relevant RAG examples are selected for few-shot prompting. With this additional contextual guidance, the model produces a corrected SQL query, thereby improving execution accuracy and robustness.

which often lack the zero-shot reasoning capacity needed to benefit fully from prompting-only methods. In ReSQL, the goal is not only to elicit reasoning at inference time, but to internalize a reusable reasoning prior that improves both initial generation and subsequent correction.

Prompting and Retrieval-Augmented Techniques for Text-to-SQL: Text-to-SQL systems commonly rely on in-context learning and zero-shot prompting, often enhanced with retrieval-augmented techniques. Prompting strategies such as Chain-of-Thought reasoning (Zhang et al., 2023) and decompositional methods (e.g., DAIL-SQL (Gao et al., 2023), MAC-SQL (Wang et al., 2023)) improve query generation. Retrieval-Augmented Generation (RAG) further enhances in-context learning by selecting relevant examples, as demonstrated in Text-to-SQL systems such as Dubo-SQL (Thorpe et al., 2024) and Ziletti and D'Ambrosi (2024), particularly in domain-specific settings.

In addition, multi-stage prompting approaches, including schema-aware and schema-linking prompts, improve SQL generation (Xiong et al., 2024).

Reasoning for Text-to-SQL: Text-to-SQL parsing enables non-experts to query databases using natural language, yet models often struggle with execution errors, particularly small language models. To address this, several methods incorporate reasoning mechanisms to improve SQL generation. CHASE-SQL (Pourreza et al., 2025) explores multi-path reasoning with candidate selection, SQL-CRAFT (Xia et al., 2024) employs an interactive correction loop with Python-based reasoning, and GRL-SQL (Gong and Sun, 2024) models schema relationships via graph-based self-attention.

Early efforts have laid a solid foundation for Text-to-SQL, but the integration of self-correction techniques remains underdeveloped. While SQL execution feedback and retrieval-augmented generation (RAG) have been studied separately (Xu et al., 2025; Gao et al., 2023; Pourreza and Rafiei, 2023; Wang et al., 2024), their joint integration remains underexplored. Our approach builds on this line of work by combining feedback fine-tuning with retrieval, where fine-tuning supports general reasoning and retrieval facilitates the recall of rare error cases.

3 ReSQL Framework

The ReSQL framework introduces a unified, self-supervised learning paradigm in which a single lan-

guage model autonomously generates, diagnoses, and corrects its own Text-to-SQL errors, without relying on model distillation. Beginning with the SPIDER and BIRD benchmarks, the model first produces SQL queries, including erroneous ones, and then leverages execution feedback to construct structured reasoning explanations, forming the ReSQL reasoning dataset. This self-generated dataset is subsequently used to fine-tune the same model, enhancing its robustness, self-correction capabilities, and awareness of reasoning errors.

The overall architecture of ReSQL, illustrated in Figure 1, consists of three tightly integrated stages: (1) Reasoning data generation, (2) Self-improving training, and (3) retrieval-augmented inference. Each stage plays a distinct yet complementary role in the framework and is described in detail below.

From a systems perspective, ReSQL can be viewed as a closed-loop pipeline: errors produced during generation are automatically captured, transformed into structured reasoning supervision, and later reused through retrieval to support correction. This pipeline view is central to the framework's novelty, as it operationalizes self-improvement through repeated error accumulation, synthesis, and reuse.

3.1 Generating Self-Improving 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.

Formally, the Text-to-SQL task can be expressed as a mapping:

$$s_i = f_G(d_i, q_i), \tag{1}$$

where q_i is a natural-language question, d_i is the corresponding database schema, and s_i is the generated SQL query. Executing s_i on d_i produces feedback:

$$e_i = f_E(s_i, d_i), \tag{2}$$

with e_i representing either an execution result or an error message.

The augmented reasoning data maintains the same input–output format as the original training pairs (natural language to SQL), but we embed the error analysis and correction process into the training examples using structured templates (See Appendix A.10.2). This ensures consistency across datasets and enables the model to learn from both original (NL-SQL) and rectified (NL-Reasoning-Correct SQL) formats. During fine-tuning, both formats are mixed, with special tokens delimiting reasoning sequences, so the model can generalize across heterogeneous supervision without confusion.

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, and observe high correctness rates across all models. We also provide a human evaluation, further confirming the reliability of the generated reasoning data. Details on the evaluation criteria and results can be found in Appendix A.5. The generated reasoning datasets for each model are openly released to support reproducibility and further research in reasoning-based Text-to-SQL modeling.

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 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.

Given an error feedback e_i, ReSQL retrieves the k most similar reasoning exemplars:

$$r_i = \big(\mathrm{sim}\big((q_i, e_i),\, \mathcal{E}\big),\, k\big), \tag{3}$$

where $\mathcal{E}$ is the vector store of prior error–reasoning pairs and $\mathrm{sim}(\cdot)$ denotes embedding similarity. The model then refines its initial SQL prediction $s_i^{(0)}$ using this retrieved context:

$$s_i^{(k+1)} = f_R\big(q_i, d_i, s_i^{(k)}, e_i, r_i\big), \tag{4}$$

and this retrieval–refinement process is repeated until a correct or converged query is obtained.

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 Appendix A.9, the distribution of error types is highly imbalanced, leaving rare or domain-specific cases underrepresented and limiting the model's ability to generalize even when trained on error samples. In this context, retrieval acts as an external recall mechanism, enhancing our unified approach by exposing rare yet relevant patterns during inference. The retrieval mechanism is not redundant with fine-tuning; rather, fine-tuning equips the model with the reasoning scaffold needed to utilize retrieved examples effectively. We demonstrate that ReSQL with RAG significantly reduces these error types, indicating enhanced robustness in handling text-to-SQL errors.

4 Experimental Setup

4.1 Models

We evaluate instruction-tuned LLMs across diverse families and scales to study the effect of fine-tuning on a self-debugging dataset, including Llama-3.2 (1B, 3B), Llama-3.1 8B (Dubey et al., 2024), Qwen-2.5 (1.5B, 3B, 7B) (Yang et al., 2024), Mistral-v0.3 7B (Jiang et al., 2023), and Gemma2 (2B, 9B) (Team et al., 2024). We also include CodeS (Li et al., 2024a), a state-of-the-art Text-to-SQL model pretrained on large-scale synthetic (NL, SQL) data.

For comparison, we evaluate instruction-only baselines and standard supervised fine-tuning. The baseline uses the instruction-tuned model without additional fine-tuning, while Simple SFT applies conventional supervised fine-tuning on the SPIDER or BIRD training splits without error reasoning or retrieval. This setup represents standard fine-tuning pipelines and provides a reference point for evaluating self-correction methods.

To evaluate inference-time self-correction for handling SQL execution errors, we compare several representative approaches. MAGIC (Askari et al., 2025) performs iterative correction using a multi-agent framework that identifies errors and revises queries based on predefined correction guidelines. Self-Debugging (Chen et al., 2023) enables models to refine their own SQL outputs by executing queries, generating natural language explanations, and using the resulting feedback for revision. Self-Consistency (Wang et al., 2022) generates multiple candidate queries and selects the most frequent one via voting, with additional retries when execution errors occur.

All fine-tuning used LoRA (Hu et al., 2022) parameter-efficient updates to make large models manageable while keeping training consistent across scales. The detailed setup is summarized in Appendix A.7.

The training objective minimizes the token-level loss between the refined output and the gold SQL:

$$\mathcal{L}_{\mathrm{ReSQL}} = \sum_i \ell\Big(f_R(q_i, d_i, s_i', e_i, r_i),\, s_i^*\Big), \tag{5}$$

where s_i' is the model's initial SQL, s_i^* the ground-truth SQL, and ℓ the cross-entropy loss.

4.2 Dataset

We evaluated the models on two cross-domain datasets: SPIDER and BIRD. SPIDER contains 10,181 natural language questions paired with 5,693 unique SQL queries covering 200 databases and 138 domains. It is divided into 8,659 training and 1,034 development examples, with SQL queries categorized into four difficulty levels: Easy, Medium, Hard, and Extra Hard. BIRD comprises 12,751 question–SQL pairs across 95 databases and 37 professional domains, including blockchain, healthcare, and education. Beyond SQL queries, it incorporates four types of external knowledge—numeric reasoning, domain-specific information, synonyms, and value illustrations—and groups queries into three difficulty levels: Simple, Medium, and Challenging. To facilitate schema linking, we provided sample table rows and external knowledge as hints. Each model

Method	Llama			Qwen			Mistral	Gemma		CodeS			GPT-4
	1B	3B	8B	1.5B	3B	7B	7B	2B	9B	1B	3B	7B	
Baseline	3.78	22.75	38.27	13.82	24.45	44.39	28.23	13.36	37.16	22.23	29.07	30.77	46.35
Simple SFT	13.56	26.66	43.74	17.28	25.10	48.50	41.59	18.51	40.42	26.86	31.94	36.11	-
Self-Correction Guideline (Askari et al., 2025)	9.84	25.55	45.89	15.51	26.53	49.74	43.61	15.45	43.22	27.84	34.10	38.07	49.87
Self-Debugging (Chen et al., 2023)	13.49	27.31	44.00	16.88	24.58	48.91	42.24	18.77	41.33	26.92	33.57	37.87	49.41
Self-Consistency (Wang et al., 2022)	14.34	26.86	44.46	18.25	25.81	49.15	42.37	19.23	40.29	27.05	32.79	38.92	49.67
ReSQL w/o RAG	23.79	42.83	48.24	26.73	38.33	52.74	47.26	25.95	**50.26**	33.96	40.68	51.04	-
ReSQL	**24.84**	**44.04**	**48.83**	**28.23**	**39.18**	**53.78**	**47.65**	**26.92**	49.74	**34.94**	**41.33**	**52.28**	-

Table 2: Execution accuracy (EX) on the BIRD-dev dataset after the second error-correction iteration. GPT-4 results are reported only for methods not requiring fine-tuning. Best scores are in **bold**; second-best are underlined.

Method	Llama			Qwen			Mistral	Gemma		CodeS			GPT-4
	1B	3B	8B	1.5B	3B	7B	7B	2B	9B	1B	3B	7B	
Baseline	20.02	49.03	59.67	38.01	46.71	68.76	44.00	44.78	61.12	55.71	64.60	66.05	79.35
Simple SFT	52.03	57.35	75.53	53.09	57.16	76.11	75.82	55.32	75.15	62.67	68.09	73.21	-
Self-Correction Guideline (Askari et al., 2025)	51.35	58.41	76.60	51.26	57.64	78.34	77.76	58.22	78.92	61.61	68.86	73.89	79.98
Self-Debugging (Chen et al., 2023)	52.42	57.74	76.60	54.35	58.32	79.69	77.66	56.29	77.95	63.15	68.09	73.50	79.78
Self-Consistency (Wang et al., 2022)	54.26	59.86	75.92	54.35	58.99	77.47	76.60	55.90	75.24	63.06	69.63	73.31	79.98
ReSQL w/o RAG	61.25	68.41	77.48	64.39	68.25	80.95	76.95	60.95	**81.32**	67.99	72.63	77.57	-
ReSQL	**62.34**	**69.25**	**77.58**	**66.29**	**70.36**	**81.29**	**78.53**	**62.02**	80.51	**68.86**	**73.11**	**78.53**	-

Table 3: Execution accuracy (EX) on the SPIDER-dev dataset after the second error-correction iteration. GPT-4 results are reported only for methods not requiring fine-tuning. Best scores are in **bold**; second-best are underlined.

was trained on a distinct subset of instances derived from its execution errors on the training set.

4.3 Metric

We assess model performance using Execution Accuracy (EX) and Correction Rate (CR).

Execution Accuracy (EX) measures whether the execution results of generated SQL queries match those of the reference (gold) query. This metric accounts for semantically equivalent but syntactically different SQL formulations. Performance is reported after up to two correction iterations, as improvements beyond the second iteration are typically marginal (see Appendix A.2).

Correction Rate (CR) quantifies the proportion of initially incorrect queries that are successfully corrected through the model's error-revision process. A correction is deemed successful only when the revised query yields execution results consistent with the gold query, not merely when it becomes executable. Thus, CR captures the model's ability to transform erroneous outputs into semantically correct ones.

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 22.23% 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.

Even for larger models with inherently higher execution accuracy, ReSQL provides consistent improvements, demonstrating that fine-grained error correction remains valuable across scales. Results for 32B and 70B models confirm the framework's scalability and robustness (See Appendix A.3).

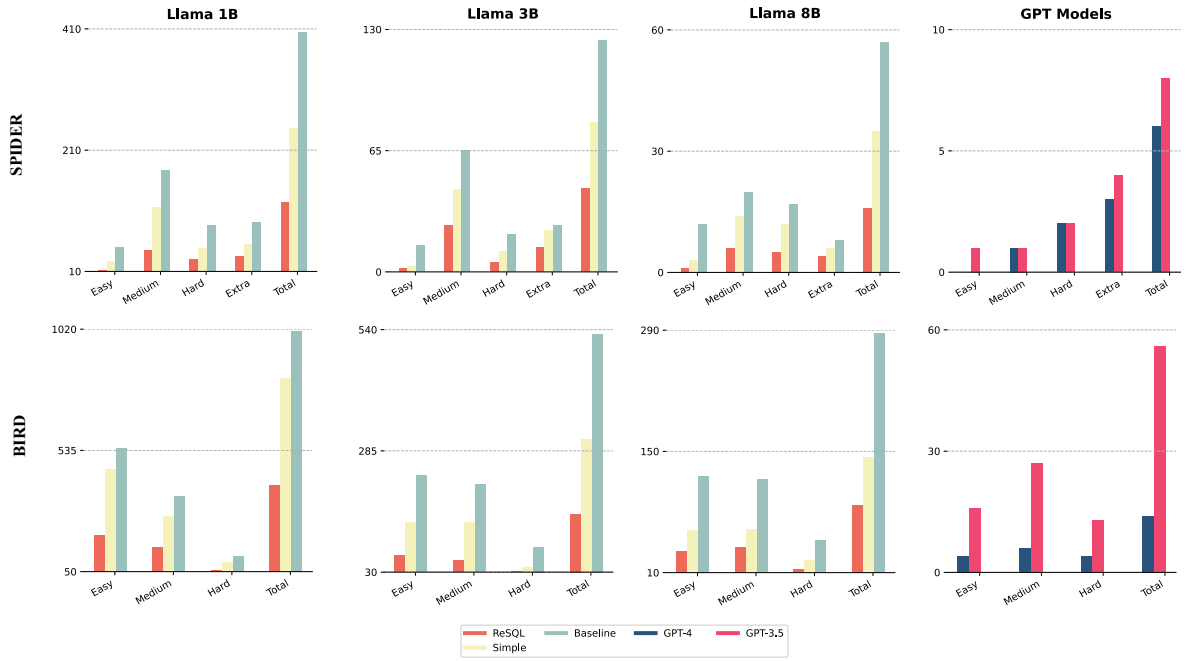

Figure 2: Remaining execution errors after the second correction iteration on SPIDER and BIRD (by difficulty). We compare Baseline (no fine-tuning), Simple SFT (fine-tuned on task data), and ReSQL for Llama Instruct models (1B/3B/8B). GPT results are shown on the right.

The comparison between ReSQL and ReSQL w/o RAG further highlights the benefit of RAG, particularly for rare or complex queries. For instance, Mistral-7B achieves a performance increase from 76.95% to 78.53% on SPIDER-dev, while Qwen-3B shows the largest RAG-driven gain—from 68.25% to 70.36%, demonstrating the effectiveness of retrieval-based correction.

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

Table 4: Correction rate (CR) of Simple SFT vs. ReSQL (Llama-3.1 8B) across difficulty levels.

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. To contextualize these results, we directly compare against GPT-4 equipped with self-correction methods. GPT-4 achieves 79.98% execution accuracy on SPIDER and 49.87% on BIRD. Our ReSQL-trained 7–9B models achieve up to 81.29% (SPIDER) and 53.78% (BIRD), thereby surpassing GPT-4 on the more challenging BIRD benchmark and approaching parity on SPIDER. This demonstrates that ReSQL's structured reasoning and retrieval augmentation provide complementary benefits even beyond advanced correction strategies.

5.2 Error Correction Result

We further compared the correction rates of simple SFT and ReSQL to examine whether correcting execution errors also results in semantic correction.

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%),

Error types	Freq.	w/o RAG	top-3	Rel. Δ (%)
Semantic error	Common	42.37	**42.11**	↓ 0.6
No such column	Common	6.13	**6.00**	↓ 2.1
No such function	Common	**1.56**	1.63	↑ 4.5
No such table	Common	**1.17**	1.30	↑ 11.1
Ambiguous column name	Rare	0.52	**0.33**	↓ 36.5
Syntax error	Common	**2.35**	**2.35**	0.0
Unrecognized token	Rare	0.91	**0.85**	↓ 6.6
More than one statement	Rare	0.33	**0.20**	↓ 39.4
Incomplete input	Rare	0.26	**0.20**	↓ 23.1
Misuse of agg. function	Rare	0.46	**0.40**	↓ 13.0
Misuse of window func.	Rare	0.33	**0.26**	↓ 21.2
Wrong num. of arguments	Rare	0.20	**0.13**	↓ 35.0
Aggregate with GROUP BY	Rare	0.20	**0.00**	↓ 100.0
ORDER BY before UNION	Rare	0.26	**0.07**	↓ 73.1
1st ORDER BY mismatch	Rare	**0.13**	**0.13**	0.0
Incorrect prop.		57.17	**55.96**	↓ 2.1

Table 5: Error-type analysis of incorrect SQL queries on BIRD-dev using the Llama 3.2 3B model. Errors are grouped by frequency (*Rare* if <1% of total). **Rel.** Δ (%) shows how much the *top-3* results differ from the *w/o RAG* baseline. Lower values, indicating better performance, are shown in **bold**.

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 illustrates the execution error counts across varying model sizes for both the SPIDER and BIRD datasets. Across all Llama models (1B, 3B, and 8B), ReSQL consistently outperforms the baseline and Simple SFT approaches, yielding fewer execution errors across all difficulty levels.

For the SPIDER dataset, ReSQL substantially reduces errors across all difficulty levels, achieving the lowest total error among compared methods. A similar pattern appears in the BIRD dataset, where ReSQL shows strong robustness, particularly in the Medium and Hard categories. Although BIRD's overall error counts are higher due to its greater complexity, ReSQL consistently yields significant improvements across both datasets and difficulty tiers.

These results confirm that ReSQL provides a reliable mechanism for mitigating execution errors and enhancing correction accuracy, with consistent performance gains across model scales and dataset complexities.

5.3 Evaluating RAG in SQL Error Correction

Table 5 presents an error-type breakdown of incorrect SQL queries on BIRD-dev using the Llama-3.2 3B model, comparing ReSQL w/o RAG and ReSQL (top-3 retrieval). Overall, ReSQL (top-3) reduces the total incorrect proportion from 57.17% to 55.96% (↓2.1%), indicating consistent gains from retrieval augmentation.

The dominant remaining failure mode is semantic error, which accounts for 42.11% of incorrect queries even after retrieval augmentation. These failures typically arise when the model produces executable SQL that does not faithfully capture the intended semantics of the question. In many cases, the ambiguity is logical rather than syntactic, involving incorrect aggregation scope, mistaken join paths, misinterpreted comparison targets, or failure to preserve question constraints. We manually inspected the remaining semantic errors and identified four recurring subtypes: *(1) aggregation mismatch*, where the model selects an incorrect counting or grouping scope; *(2) join-path confusion*, where the query is executable but follows an incorrect table relationship; *(3) constraint omission*, where one condition expressed in the question is omitted; and *(4) ambiguity resolution failure*, where multiple plausible SQL interpretations exist and the model selects the wrong one.

Semantic and syntax errors, the dominant categories, show marginal improvements (↓0.6% and 0%, respectively), suggesting retrieval modestly enhances query intent understanding. This is expected, as such common patterns are already well captured by the fine-tuned model through supervised learning. Schema-related errors (e.g., *No such column*, *No such table*) remain challenging, with mixed effects (↓2.1% and ↑11.1%, respectively).

Notably, ReSQL achieves large relative reductions in rare, reasoning-intensive errors, such as *Ambiguous column name* (↓36.5%), *More than one statement* (↓39.4%), *ORDER BY before UNION* (↓73.1%), and *Aggregate with GROUP BY* (↓100%), demonstrating stronger robustness to long-tail query formulation failures. These findings indicate that retrieval augmentation primarily benefits structural and low-frequency error types, improving overall SQL generation reliability.

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. The removal

	SPIDER		BIRD	
	EX	Δ	EX	Δ
All tools	69.25	–	44.04	–
w/o reasoning	58.22	-11.03	26.27	-17.61
w/o RAG	68.41	-0.84	42.83	-1.21
1-pass revise	65.86	-3.39	40.10	-3.78

Table 6: Ablation on SPIDER and BIRD using Llama-3.2 3B. EX is execution accuracy (%), and Δ is the difference from All tools in percentage points.

of RAG has a smaller effect, yet it provides critical information for fixing rare errors. The 1-time revise setting improves results compared to ablated variants but remains inferior to the full system, highlighting the necessity of iterative refinement for achieving robust generalization.

6 Conclusion

In this paper, we introduce the concept of retrieval-augmented error reasoning, a self-improving paradigm in which a model learns to generalize its own correction strategies across contexts. Unlike prior self-correction or retrieval-based methods, ReSQL integrates these mechanisms into a unified reasoning process, effectively endowing Text-to-SQL systems with a reusable error reasoning prior. More broadly, ReSQL suggests that progress in Text-to-SQL may come not only from larger base models, but also from better pipeline design that systematically converts model failures into reusable correction knowledge.

Our findings highlight that explicit error reasoning is central to robust Text-to-SQL generalization, while retrieval augmentation further strengthens resilience against rare or complex query failures. Together, these components establish a scalable, cost-efficient, and interpretable framework for building self-improving Text-to-SQL systems. We hope ReSQL represents a meaningful step toward developing reliable, reasoning-aware database interfaces that are broadly accessible across model scales and application domains. The code for the ReSQL self-generation framework, along with all reasoning datasets generated from 1–9B parameter models, is publicly available in our GitHub repository.

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.

Another limitation concerns generalizability under real-world distribution shift. Our experiments are conducted on SPIDER and BIRD, which are strong cross-domain benchmarks but do not fully represent truly out-of-distribution scenarios. In practice, deployed systems may encounter unseen schemas, changing database conventions, diverse user query formulations, and domain shifts that are not captured by these benchmarks. Therefore, while ReSQL demonstrates improved robustness within benchmark settings, its behavior under stronger OOD shifts remains to be validated.

ReSQL is not label-free. Although the framework autonomously generates reasoning traces and correction explanations, it relies on gold SQL queries during dataset construction and training. Accordingly, we use the term self-improving to refer to autonomous reasoning-data generation rather than fully unsupervised learning. Reducing or removing dependence on gold SQL supervision is an important direction for future work.

Lastly, we acknowledge the limitations of execution accuracy (EX), our primary evaluation metric. EX, implemented as test suite accuracy in SPIDER and BIRD, remains the most widely adopted standard for evaluating Text-to-SQL models. Although it can detect semantically equivalent but syntactically different queries, it may still produce false positives when distinct SQL queries return identical results on limited test suites, and false negatives arising from annotation noise or incomplete test coverage. Despite these limitations, we adopt EX to ensure comparability with prior work, while recognizing the importance of developing more semantically aware evaluation metrics for future research.

References

Arian Askari, Christian Poelitz, and Xinye Tang. 2025. Magic: Generating self-correction guideline for in-context text-to-sql. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 23433–23441.

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*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank adaptation of large language models. In *Proceedings of the Tenth International Conference on Learning Representations*.

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. 2025. Chase-sql: Multi-path reasoning and preference optimized candidate selection in text-to-sql. In *Proceedings of the 13th International Conference on Learning Representations*, Singapore. OpenReview.net.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems*, 36:36339–36348.

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*.

Dingzirui Wang, Longxu Dou, Xuanliang Zhang, Qingfu Zhu, and Wanxiang Che. 2024. Improving demonstration diversity by human-free fusing for text-to-sql. *arXiv preprint arXiv:2402.10663*.

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*.

Bo Xu, Shufei Li, Hongyu Jing, Ming Du, Hui Song, Hongya Wang, and Yanghua Xiao. 2025. Boosting text-to-sql through multi-grained error identification. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 4282–4292.

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*.

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

A.1 Notation Table

Symbol	Meaning
$q_i \in \mathcal{Q}$	Natural language question
$d_i \in \mathcal{D}$	Database input (schema, hints)
$s_i, s_i', s_i^{(k)}$	SQL: gold/predicted/iterative refinement
e_i	Execution feedback (result/error)
r_i	Retrieved error–reasoning exemplars
f_G	Base Text-to-SQL generator
f_E	Execution function producing e_i
f_R	ReSQL refinement function
$\mathcal{E}$	Vector store of error–reasoning exemplars

Table 7: Notation used in ReSQL.

A.2 Execution Error Reduction

We analyze execution-error reduction across iterative correction steps (C1–C4) on the SPIDER and BIRD datasets. Tables 8 and 9 report the quantitative results, while Figures 3 and 4 provide visual illustrations of the error-reduction trends on SPIDER and BIRD, respectively. Overall, ReSQL achieves consistent error reduction across all model sizes. Smaller models (e.g., Llama-3.2 3B) benefit the most, while larger models (e.g., Llama-3.1 8B) also show notable gains, demonstrating that ReSQL improves robustness regardless of model scale.

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: Reduction of execution errors on the BIRD dataset across correction iterations (C1–C4).

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 9: Reduction of execution errors on the SPIDER dataset across correction iterations (C1–C4).

A.3 Scalability and Consistency across Model Sizes

While ReSQL primarily targets models under 10B parameters, we further evaluate its scalability to larger models including Llama-3.1 70B and Qwen-2.5 32B. As summarized in Tables 10 and 11, ReSQL continues to yield improvements even at larger scales, demonstrating that its error-reasoning and correction mechanism generalizes effectively across different model sizes. Although the relative gains are smaller compared to lightweight models, the results confirm that ReSQL provides consistent performance benefits without requiring architecture-specific adaptations.

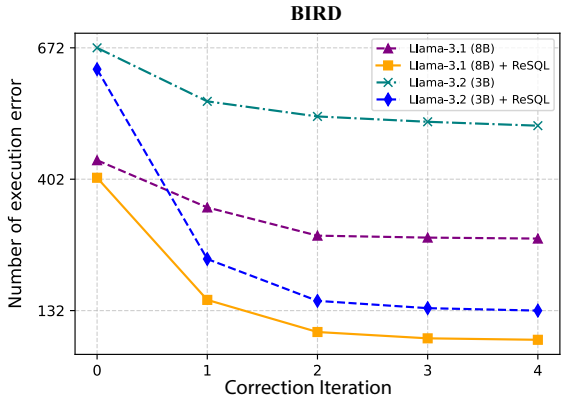

Figure 3: Execution error reduction across correction iterations on BIRD dataset.

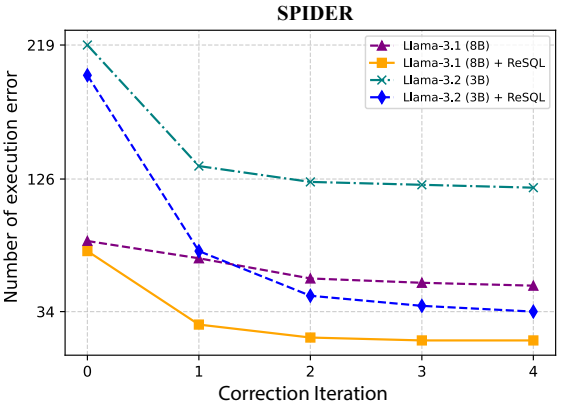

Figure 4: Execution error reduction across correction iterations on SPIDER dataset.

Method	**Llama-3.1 70B**	**Qwen-2.5 32B**
Baseline	46.28	50.85
Simple SFT	50.85	55.74
Self-Correction Guideline (Askari et al., 2025)	51.76	57.04
Self-Debugging (Chen et al., 2023)	51.04	55.93
Self-Consistency (Wang et al., 2022)	51.56	56.98
ReSQL w/o RAG	51.69	56.39
ReSQL	**52.09**	**57.69**

Table 10: Execution accuracy (EX) on the BIRD-dev dataset for large-scale models.

Method	Llama-3.1 70B	Qwen-2.5 32B
Baseline	71.18	78.63
Simple SFT	76.60	83.46
Self-Correction Guideline (Askari et al., 2025)	**78.82**	83.95
Self-Debugging (Chen et al., 2023)	77.37	83.56
Self-Consistency (Wang et al., 2022)	77.18	83.56
ReSQL w/o RAG	77.55	**84.14**
ReSQL	77.85	**84.14**

Table 11: Execution accuracy (EX) on the SPIDER-dev dataset for large-scale models.

A.4 Effectiveness of Retrieval-Augmented Correction

To assess whether retrieval augmentation enhances generalization to unseen error types, we compare *top-3* and *bottom-3* retrieval strategies on the BIRD-dev dataset using Llama-3.2 3B. As shown in Table 12, the top-3 retrieval consistently achieves lower error proportions across nearly all categories, with substantial reductions (up to 100%) on rare or unseen error cases. This demonstrates that retrieval-augmented reasoning improves robustness to unseen execution errors, providing better generalization beyond memorized error patterns.

Error types	bottom-3	top-3	Rel. Δ (%)
Semantic error	43.02	**42.11**	↓ 2.1
No such column	6.98	**6.00**	↓ 14.0
No such function	1.83	**1.63**	↓ 10.9
No such table	1.50	**1.30**	↓ 13.3
Ambiguous column name	0.52	**0.33**	↓ **36.5**
Syntax error	2.74	**2.35**	↓ 14.6
Unrecognized token	0.91	**0.85**	↓ 6.6
More than one statement	0.33	**0.20**	↓ **39.4**
Incomplete input	0.26	**0.20**	↓ **23.1**
Misuse of agg. function	0.46	**0.40**	↓ 13.0
Misuse of window func.	0.33	**0.26**	↓ 21.2
Wrong num. of arguments	0.20	**0.13**	↓ **35.0**
Aggregate with GROUP BY	0.20	**0.00**	↓ **100.0**
ORDER BY before UNION	0.26	**0.07**	↓ **73.1**
1st ORDER BY mismatch	**0.13**	**0.13**	0.0
Incorrect prop.	59.67	**56.12**	↓ 5.9

Table 12: Comparison of top-3 vs bottom-3 retrieval on BIRD-dev (Llama-3.2 3B). Lower values indicate better performance.

A.5 Quality Validation of Reasoning Data

We validate the quality of the self-generated reasoning dataset through both automatic and human evaluation.

Automatic Evaluation. Using the G-Eval framework (Liu et al., 2023b), all evaluated models achieve high correctness rates (above 85%), as shown in Table 13, indicating that ReSQL consistently produces accurate reasoning data suitable for fine-tuning.

Model	G-Eval Accuracy (%)
Llama-3.1 8B	97.20
Llama-3.2 3B	92.21
Llama-3.2 1B	85.78

Table 13: Automatic evaluation of ReSQL reasoning data via G-Eval.

Human Evaluation. Although ReSQL is designed as a self-improving framework that operates without human intervention, we conduct a human evaluation to verify that the autonomously generated reasoning traces are genuinely reliable. We sample 200 traces generated by smaller models (under 3B) and evaluate them on two criteria defined in Figure 5. As shown in Table 14, even these smaller models achieve 97.5% correctness and 97.0% helpfulness (score ≥ 2), confirming that ReSQL can autonomously generate reliable reasoning data without manual annotation.

Correctness — *Is the diagnosis right?*
3 Accurate diagnosis and fix direction
2 Error identified, root cause incomplete
1 Root cause contradicts schema

Helpfulness — *Useful as training data?*
3 Schema-specific, concrete references
2 Right direction but generic
1 Vague or misleading

Figure 5: Evaluation rubric for reasoning traces.

Criterion	Score	Count	%
Correctness	3 (Correct)	123	61.5%
	2 (Partially Correct)	72	36.0%
	1 (Incorrect)	5	2.5%
Helpfulness	3 (Helpful)	103	51.5%
	2 (Somewhat Helpful)	91	45.5%
	1 (Not Helpful)	6	3.0%

Table 14: Human evaluation results on 200 sampled traces.

A.6 Reasoning Overhead

Table 15 reports the average reasoning tokens and inference time generated during ReSQL inference. Sequences are short (∼100 tokens), completing in under 1.2 seconds across all models, imposing negligible overhead compared to full query gener-

ation. This demonstrates ReSQL's practicality for real-world systems.

Model	Avg. Tokens	Avg. Time (s)
Qwen-2.5 1.5B	100.37	0.55
Qwen-2.5 3B	99.27	0.78
Qwen-2.5 7B	96.37	1.14
Llama-3.2 1B	105.37	0.58
Llama-3.2 3B	100.39	0.75
Llama-3.1 8B	98.68	1.15

Table 15: Average reasoning tokens and inference time on A100 80GB (vLLM, BF16).

A.7 LoRA parameter setting

Table 16 summarizes the LoRA fine-tuning configuration applied to all model variants. The setup ensures consistent optimization conditions across experiments, allowing performance differences to be attributed primarily to the ReSQL framework rather than training hyperparameters.

Parameter	Value
Optimizer	AdamW
Peak learning rate	2×10^{-5} (cosine decay schedule)
Effective batch size	128
Epochs	2
Precision	bfloat16

Table 16: LoRA fine-tuning configuration applied across all model variants.

A.8 Cross-Domain Transferability

Table 17 evaluates transfer between SPIDER and BIRD reasoning supervision. While in-domain supervision performs best, cross-domain reasoning remains effective. Importantly, even on the more challenging BIRD dataset, ReSQL still provides strong gains, indicating that our method leverages self-generated reasoning data to address intrinsic model weaknesses rather than merely memorizing domain-specific patterns. This highlights ReSQL's ability to internalize corrections and generalize beyond the training domain.

A.9 Error Type Distribution

Figures 6 and 7 show the distribution of execution errors by category: BIRD is dominated by execution/function misuse, while SPIDER skews toward syntax/format. Tables 18 and 19 further detail error messages and frequencies, with "No such column" emerging as the most common issue in both datasets. Together, they provide a concise overview of dataset-specific weaknesses and highlight the need for error-type-aware strategies.

Dataset / Method	Llama-3.2 1B	Llama-3.2 3B	Llama-3.1 8B
BIRD-dev			
Baseline	3.78	22.75	38.27
ReSQL	24.84	44.04	48.83
ReSQL w/ SPIDER Reasoning	24.38	40.55	47.07
SPIDER-dev			
Baseline	20.02	49.03	59.67
ReSQL	62.34	69.25	77.58
ReSQL w/ BIRD Reasoning	61.12	68.86	77.47

Table 17: Cross-domain transferability of ReSQL reasoning data.

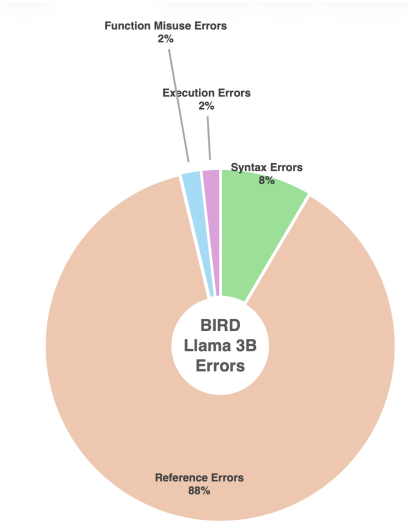

Figure 6: Distribution of BIRD SQL execution errors for Llama-3.2 3B, categorized into reference, syntax, execution and function misuse error.

A.10 Prompts

A.10.1 SQL Query Generation Prompt

Figure 8 illustrates the template we use to generate SQL queries from table schemas. The prompt provides schema details, example rows, and optional hints (for BIRD only), while the system constrains the output to a single valid SQL query.

A.10.2 Execution Error Analysis Prompt

Figure 9 shows the template for analyzing failed SQL executions. Given a question, gold query, erroneous query, and error message, the model is guided to (i) explain the failure, (ii) analyze the root cause, and (iii) suggest corrections. The response is structured in JSON with explicit error type classification.

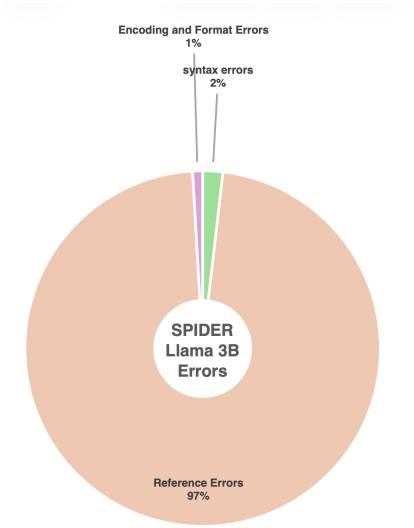

Figure 7: Distribution of SPIDER SQL execution errors for Llama-3.2 3B, categorized into reference, syntax, encoding and format error.

A.10.3 Reasoning Validation Prompt

Figure 10 presents the template for evaluating reasoning validity. The model determines whether proposed reasoning steps justify the transformation from an incorrect SQL query to the correct one, returning a binary decision (1/0). This ensures consistency in reasoning annotations.

A.11 Effect of Reasoning in Self-Correction

Figure 11 compares model behavior with and without reasoning during self-correction. Without reasoning, the model often produces another erroneous SQL query, failing to recover from the initial error. By contrast, reasoning enables the model to diagnose the root cause and generate a correct query, substantially improving execution accuracy.

A.12 Retrieval-Augmented Generation for Error Correction

Figure 12 illustrates our RAG framework. When an execution error occurs, the system retrieves similar past error cases using vector similarity and injects them as in-context examples. This approach helps the model handle underrepresented error types and improves robustness in SQL correction.

All Error Type	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 18: 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 19: Summary of SPIDER SQL execution errors and their frequencies for Llama-3.2 3B.

[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 8: 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 'Generated 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 9: 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 10: Template for Assessing the Validity of SQL Reasoning Dataset Using G-Eval.

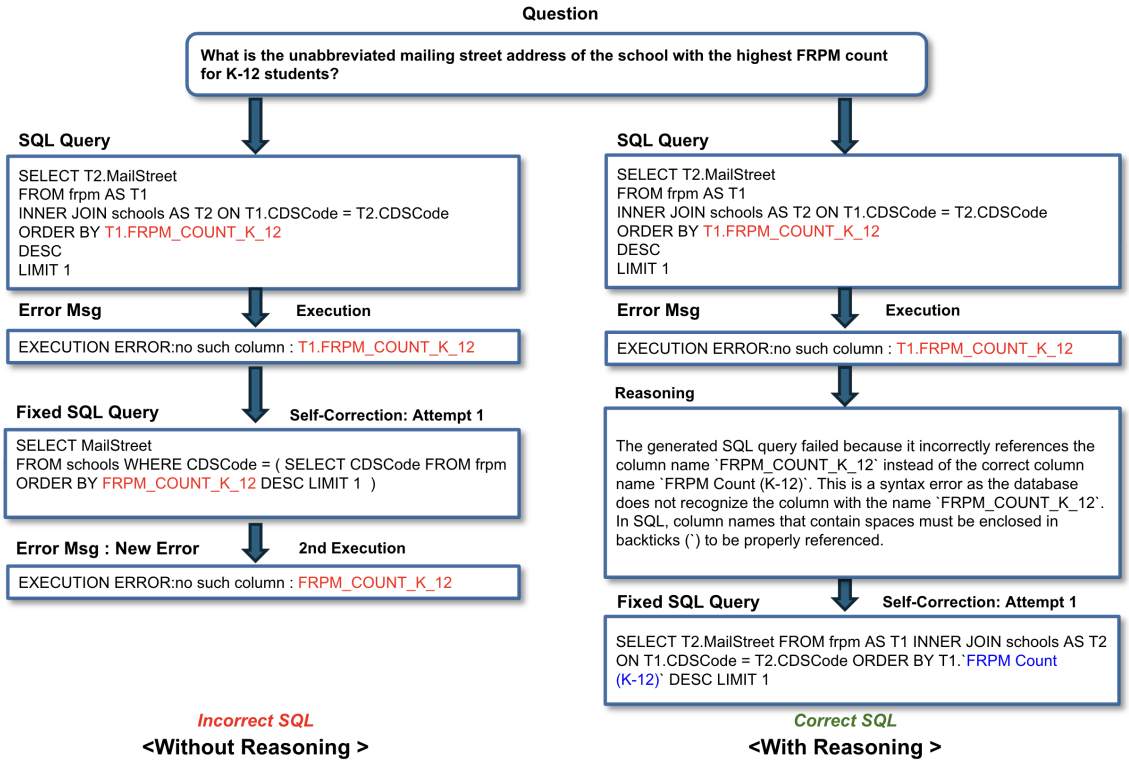

Figure 11: Comparison of SQL self-correction with and without reasoning. The model without reasoning fails to correctly self-correct the initial SQL query, generating another incorrect query even after attempting self-correction. In contrast, our model (with reasoning) identifies the root cause of the error, correctly fixes the query, and ensures execution accuracy.

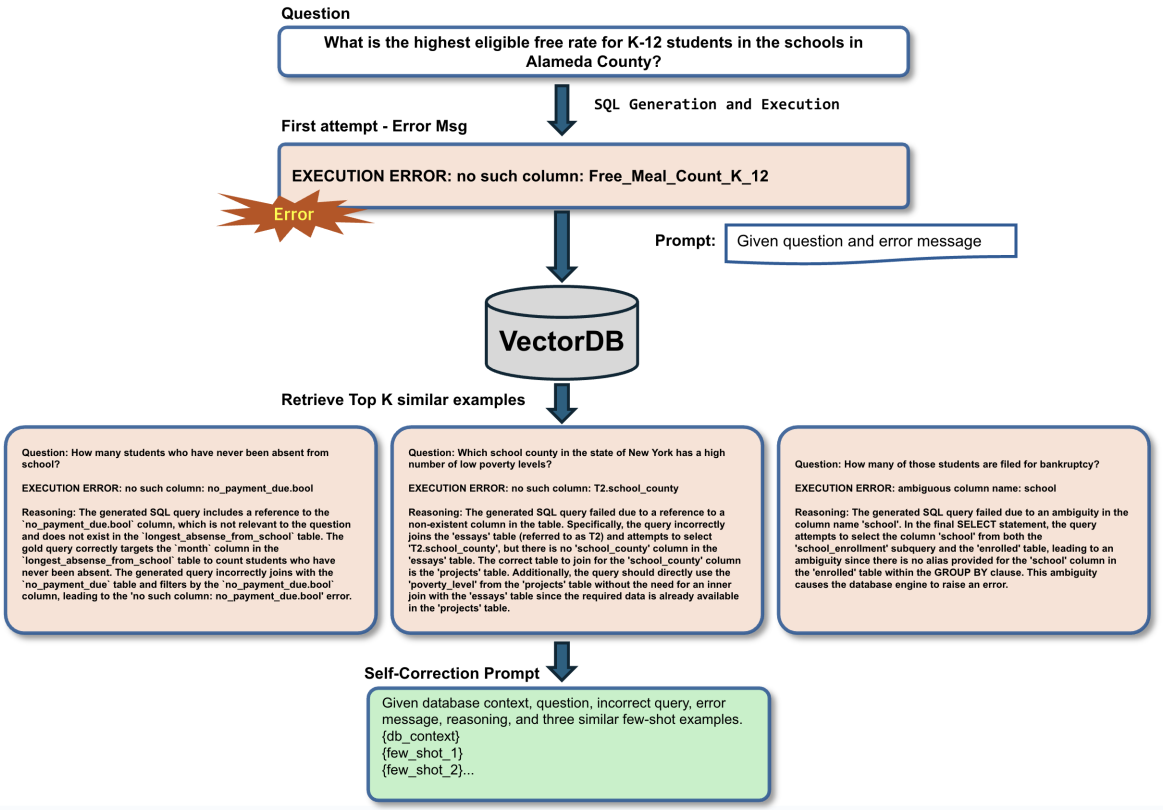

Figure 12: Overview of the RAG framework for SQL error correction. When an execution error occurs, the system retrieves the top three most similar error cases from a database of past execution errors using vector similarity. These retrieved examples serve as in-context learning references, helping the model resolve underrepresented error types and improve robustness.