SEEING: Improving Self-Debugging through Targeted Finetuning for Text-to-SQL Generation

Anonymous ACL submission

Abstract

Text-to-SQL generation has become a crucial component in natural language interfaces to databases, but it often suffers from frequent execution errors. In this paper, we introduce SEEING (**s**ql **s**elf-**d**ebug targeted **f**ine-tu**n**i**ng**), a novel training approach to enhance the self-debugging capabilities of Text-to-SQL models. We automate the generation of a self-debugging dataset designed to train models to anticipate potential execution error types and correct them using error messages in Text-to-SQL generation. Our comprehensive experiments on the SPIDER and BIRD benchmarks demonstrate that models trained with SEEING achieve execution accuracies of 79.9% on SPIDER and 45.9% on BIRD, surpassing current state-of-the-art prompting methods. Notably, SEEING exhibits higher SQL query correction rates compared to GPT-4 on both datasets. Furthermore, we show that SEEING enhances a model's error correction abilities across various levels of query complexity, not just for simple tasks. By collecting execution errors and automatically generating error datasets from the LLM, SEEING significantly advances the accuracy and robustness of Text-to-SQL models. We release both the self-debugging dataset and the accompanying code to foster further research in this area.

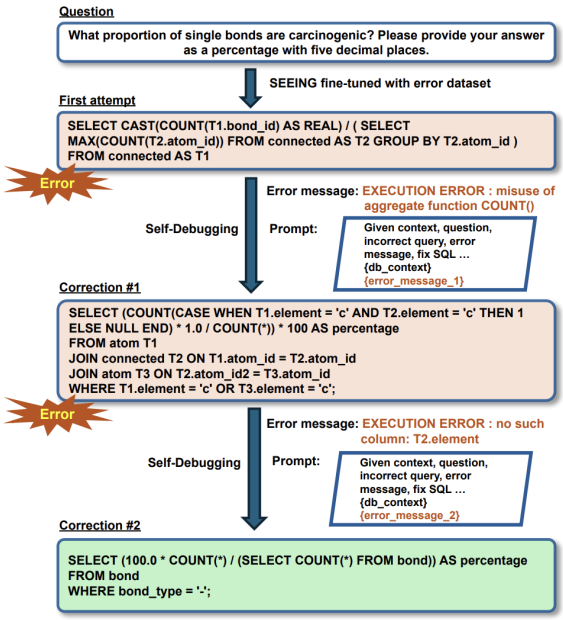

Figure 1: Overview of SQL query correction process using the fine-tuned SEEING model based on Llama-3.1-it (8B). The initial query results in an execution error, which prompts SEEING to enter a self-debugging phase. Here, the execution error message and the incorrect query are incorporated into the prompt for generating a corrected SQL query. In this example, the first correction still triggers an error, leading to a second debugging attempt. The second correction successfully resolves the issue, showcasing SEEING's ability to adjust based on context, error message, and prior query.

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). This technology has the potential to democratize data access and analysis, making it easier for a wider range of users to extract valuable insights from complex database systems. However, despite significant advancements in recent years, Text-to-SQL generation remains a challenging task, particularly for complex queries involving multiple tables, joins, nested structures, and intricate conditions. 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). As the field progressed, 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. Most recently, Large Language Models (LLMs) have shown promising results in Text-to-SQL tasks, even in zero-shot and few-shot settings (Chang et al., 2020; Gu et al., 2023; Dong et al., 2023). However, these models still fall short of state-of-the-art performance on challenging benchmarks, particularly for medium and complex queries (Yu et al., 2018; Qi et al., 2022; Rai et al., 2023). A significant challenge in Text-to-SQL generation is the prevalence of execution errors, often stemming from incorrect schema linking and SQL syntax issues. These problems are especially pronounced in multi-table schemas and complex query structures. While previous research has focused on providing better prompting methods to prevent these errors, our work takes a novel approach by addressing the problem from a different angle. In this paper, we introduce a self-debugging dataset specifically designed to teach models how to identify and rectify execution errors in Text-to-SQL tasks. Our approach is based on the premise that recognizing SQL error patterns is effective both for preventing and correcting errors. By incorporating error debugging into the learning process, we aim to enhance the robustness and reliability of Text-to-SQL models, particularly in handling complex, real-world database querying tasks. Our contributions are as follows:

We introduce a groundbreaking self-debugging dataset designed to train models to recognize and correct execution errors in Text-to-SQL generation. Through comprehensive experiments, we show that models trained with our dataset, SEEING, achieve execution accuracies of 79.9% on SPIDER (Yu et al., 2018) and 45.9% on BIRD (Li et al., 2024), surpassing all current state-of-the-art prompting methods. Notably, in the SPIDER benchmark, SEEING outperforms the GPT-4 model in both execution accuracy and fix rate, demonstrating its superior performance in handling execution errors. SEEING significantly enhances a model's error correction abilities, regardless of question complexity.

To promote further research in natural language processing and database querying, we are releasing both the self-debugging dataset and its accompanying code. By addressing execution errors directly, this work improves the robustness of Text-to-SQL model and bridges the gap between the capabilities of large language models (LLMs) and specialized task-specific models, leading to more dependable Text-to-SQL generation model.

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 Techniques for SQL Generation: 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). In addition, recent work has explored multi-staged prompting strategies to enhance LLM performance in SQL generation. Xiong et al. (2024) introduces a two-stage method to enhance LLM-to-SQL systems using schema-aware prompts and schema linking to generate more accurate SQL queries.

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 generating useful in-context learning guidelines and prompts, our work makes a significant contribution by proposing an direct fine-tuning of self-

Model	SPIDER	BIRD
Gemma2-it (9B)	636	1090
Llama-3.1-it (8B)	463	782
Total	1099	1872

Table 1: Composition of the error dataset generated by Gemma2-it (9B) and Llama-3.1-it (8B) across SPIDER and BIRD datasets

correction dataset, training the model to debug and correct execution errors during inference, which outperforms self-correction mechanisms based on various prompting techniques and make it adaptable to various LLM architectures.

3 Self-debugging dataset

Our data generation pipeline follows a structured process to create a dataset focused on improving model robustness in handling execution errors. The steps shown in Figure 3 are outlined below:

3.1 Sampling from Database Pool

We begin by randomly sampling, with replacement, from two datasets: 10,000 examples from SPIDER and 10,000 from BIRD. These datasets are chosen due to their wide variety of schemas and query structures, essential for effective model training.

3.2 Question Generation

Using the database schema and sampled rows from each dataset, we employ GPT-4 [1] to generate natural language questions. The input context differs between SPIDER and BIRD datasets. BIRD contains an additional element called "evidence," which provides external information necessary for answering the question. This evidence is included in the input during question generation for BIRD to produce more accurate and contextually relevant queries.

3.3 Error Dataset Collection

The objective of the pipeline is to enhance model handling of execution errors. We predict SQL queries from the generated questions using two open-source baseline models: Gemma2-it (9B) (Team et al., 2024) and Llama-3.1-it (8B) (Dubey et al., 2024). This dual-model approach ensures diversity in error types, leveraging the two most popular models. We then execute the predicted SQL queries and collect only those samples that result in execution errors. Samples without execution errors were discarded.

[1]We used the latest model GPT-4o-2024-08-06, with training data updated through October 2023.

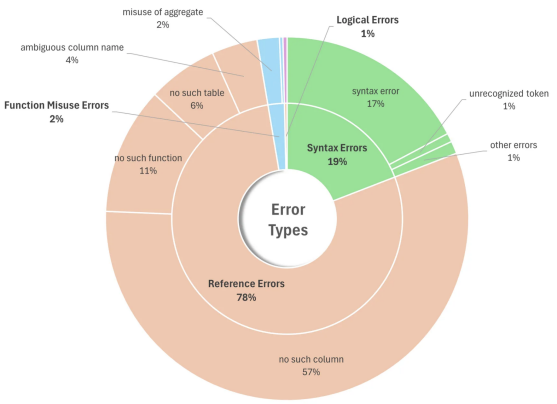

Figure 2: Donut chart illustrating the composition of SQL-generated error types from a dataset of 2,971 samples. Major error categories include Reference Errors, Syntax Errors, Function Misuse Errors, and Logical Errors.

3.4 Error Correction

To increase the robustness of the error dataset, we first employ GPT-4 to correct the erroneous SQL queries. For this, we use the database context, generated question, predicted SQL from either Gemma2-it or Llama-3.1-it, and the execution error message as input. GPT-4 returns both an execution error analysis and a corrected SQL query. These corrected SQLs are further split into two subsets: with and without error analysis, allowing us to examine the impact of error analysis in correcting execution issues. If the corrected SQL still fails upon execution, the sample is discarded, retaining only those successfully fixed by GPT-4. The final composition of the generated data is presented in Figure 2.

3.5 Human Validation

To ensure the correctness of GPT's corrected SQL queries, we conduct a human-validation process. This cross-checking step ensures that only the SQL queries passing quality checks are included in the final dataset. These validated samples are then used for fine-tuning the baseline models.

4 Experimental Setup

4.1 Models

To evaluate the impact of directly fine-tuning on a self-debugging dataset, we utilize several baseline models: i) W/o SFT: This is the instruction-tuned version of Llama 3.1 (8B) without any additional supervised fine-tuning (SFT), serving as a direct comparison. No error-handling datasets were used for training this baseline. ii) Simple SFT: We fine-tuned the Llama 3.1 model on the SPIDER and BIRD datasets, using only their respective training sets. This baseline does not incorporate self-debugging or error-handling data. iii) GPT-4 (Achiam et al., 2023): GPT-4, one of the top-performing LLMs, is included as a benchmark for comparison.

For the self-correction approach, we employ several state-of-the-art techniques to handle SQL query errors during inference: i) MAGIC: This multi-agent framework automates building the guideline for self-correction for text-to-SQL tasks through three agents—a manager, a feedback agent, and a correction agent. The feedback agent identifies SQL query errors, while the correction agent revises the queries accordingly. We applied 34 different query correction guidelines iteratively to address execution errors. ii) Self-Debugging: The SELF-DEBUGGING system enables large language models to iteratively debug their own generated code by executing it, explaining it in natural language, and using feedback to identify and correct errors—all without human intervention. iii) Self-Consistency Model: In this model, the system runs 10 inference iterations per input, selecting the most frequent query using a voting mechanism. If execution errors occur, the model retries with an updated prompt until a correct query is produced, with the maximum of 2 attempts.

In our Debug-SQL setup, we expanded the original training data (SPIDER and BIRD) by incorporating the self-debugging dataset. This dataset includes corrected SQL queries, both with and without detailed analysis, allowing us to evaluate the effectiveness of the error analysis in enhancing SQL generation.

4.2 Dataset

We evaluated our models on two challenging 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

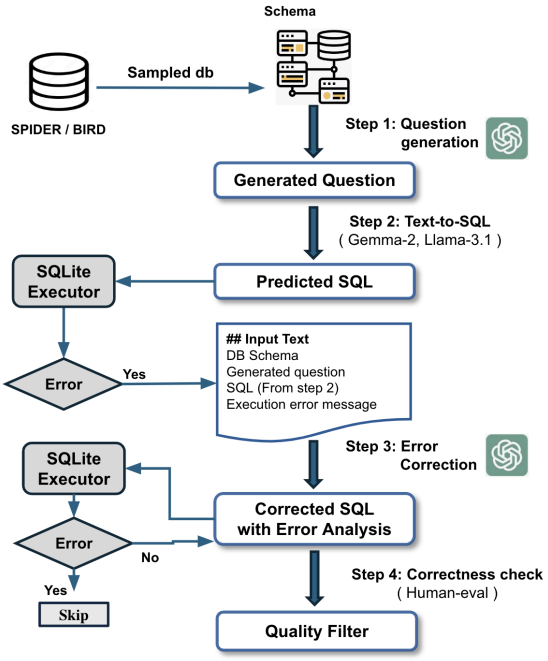

Figure 3: Error dataset generation pipeline for Text-to-SQL generation. It starts by sampling databases from SPIDER or BIRD, followed by question generation based on the schema (Step 1). The generated question is converted into a predicted SQL query using Text-to-SQL models like Gemma2-it or Llama-3.1-it (Step 2). The predicted SQL is executed via SQLite, and if errors occur, the input (including the schema, generated question, SQL, and error message) is used for error correction (Step 3). The corrected SQL is re-executed, and if errors persist, the query is skipped. Finally, correct SQL queries undergo human-evaluation-based correctness checks to ensure quality (Step 4).

hints.

As shown in Table 1, we generated 2,971 additional samples (1,099 from SPIDER and 1,872 from BIRD) through self-debugging, with an equal number of examples including and excluding detailed error analysis. These datasets were combined during training to enhance cross-domain robustness and expose the models to a broader range of error scenarios.

4.3 Metric

The model performance is evaluated using execution accuracy (EX) and SQL query fix rate (FR), 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 returns (See Figure 5).

The fix rate (FR) 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. FR is the ratio of successful corrections to total errors. Formally, this is expressed as:

$$\text{FR} = \frac{\#\,\text{Corrections}}{\#\,\text{Errors}} \tag{1}$$

This measure provides a straightforward percentage of errors that are fixed by the model. However, it does not take into account the varying difficulty levels of the errors, meaning that it treats all corrections equally, whether the error was simple or complex.

To account for the limitation of overlooking error difficulty, we introduce a log-weighted fix rate. In this approach, the model is penalized more heavily for failing to correct more challenging issues. By applying a logarithmic transformation to the fix rates, this metric minimizes the influence of extreme variations in difficulty levels.

The log-weighted fix rate formula can be defined as:

$$\text{Weighted FR (WFR)} = \frac{\sum(\log(\text{FR}) \times \mathbf{W})}{\sum \mathbf{W}} \tag{2}$$

Where $\mathbf{W}$ are corresponding integers to each difficulty. For example, easy = 1, medium = 2, hard = 3, extra hard = 4 for the SPIDER, and simple = 1, medium = 2, challenge = 3 for the BIRD.

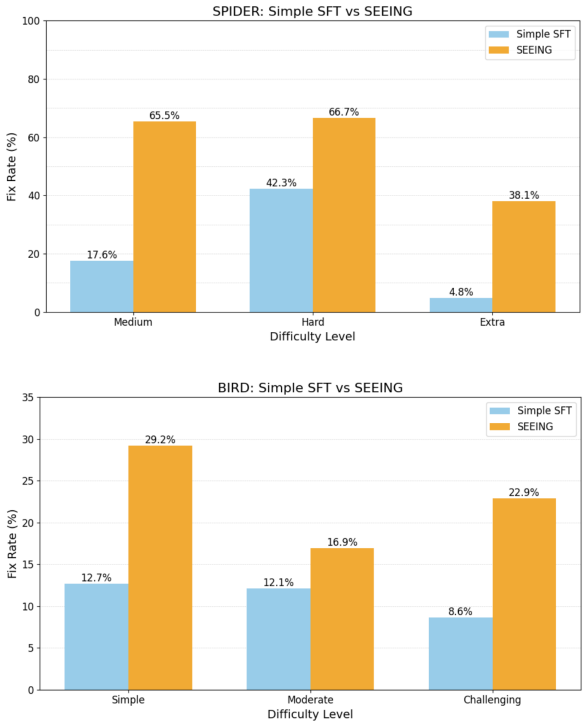

Figure 4: Comparison of fix rates (%) between the Simple SFT and SEEING models across different difficulty levels for both the SPIDER (top) and BIRD (bottom) datasets after the second correction. The SPIDER dataset includes four difficulty levels: Easy, Medium, Hard, and Extra, with the Easy level excluded due to both models only fixing a single execution error. The BIRD dataset consists of three difficulty levels: Simple, Moderate, and Challenging. Overall, SEEING demonstrates notable improvement over Simple SFT across all difficulty levels in both datasets.

In this calculation, the log of the fix rate is taken, and the result is weighted by the corresponding difficulty level of the error. Weights can be assigned based on a predetermined scheme or derived from empirical data regarding error difficulty. This approach is particularly useful for avoiding situations where models may achieve high overall fix rates by focusing on easier problems while failing to address more complex issues. By reducing the impact of extreme fix rate values, the log-weighted fix rate offers a more nuanced view of the model's performance across different levels of challenge.

This metric ensures that models capable of handling a variety of error types, especially harder cases, are rewarded accordingly, leading to a more fair comparison between models that attempt to correct text-to-SQL errors under varying circumstances.

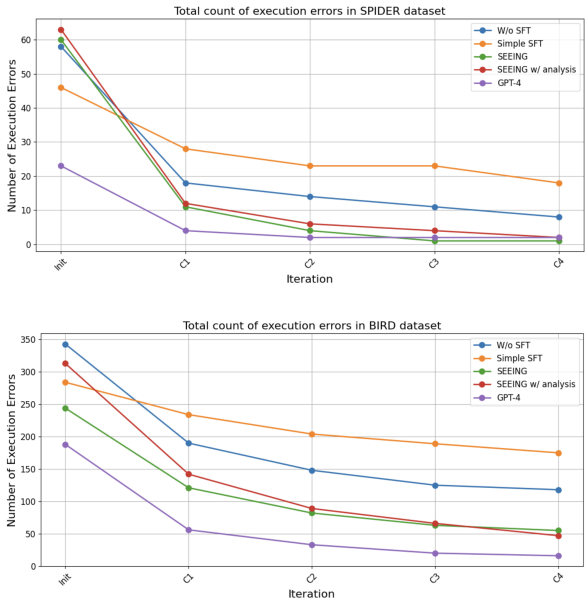

Figure 5: The graphs show the total count of execution errors across multiple correction iterations for the SPIDER (top) and BIRD (bottom) datasets. The initial attempt (Init) is followed by successive correction steps (C1, C2, etc.). Various methods are compared, including "W/o SFT," "Simple SFT," "SEEING," "SEEING w/ analysis," and "GPT-4." The results indicate a reduction in errors with each correction step, with the cutoff at C2 used for reporting purposes.

5 Result

5.1 SQL Execution Accuracy Benchmark

In Table 2, we evaluate the overall performance of different models and prompting methods on the SPIDER and BIRD benchmarks, focusing on execution accuracy (EX). The term "Init" represents the model's first attempt at predicting SQL queries. If the predicted SQL query results in an execution error, the model is allowed to iterate corrections based on feedback from the SQLite execution message. We report the results for up to two rounds of correction (C1 and C2), as further iterations show minimal improvement, reaching a performance plateau.

For these experiments, we use the Llama-3.1 (8B) instruction-tuned model, known for its state-of-the-art performance in general knowledge, mathematics, and reasoning tasks, as the backbone for Text-to-SQL tasks.

On both the SPIDER and BIRD datasets, our model, SEEING, fine-tuned with an execution error dataset, demonstrates impressive accuracy throughout the iterative correction process. Fine-tuning the model on this specific error dataset allows it to effectively handle a wide range of SQL execution errors, leading to significant improvements in correcting these errors during inference.

In the SPIDER benchmark, SEEING shows superior performance in initial SQL predictions and error corrections compared to other methods. While the Simple SFT model improves from an initial accuracy of 75.5% to 76.3% (+0.8% after correction), SEEING improves from 76.5% to 79.9% (+3.4% after correction), demonstrating a much stronger capability to recover from execution errors. This improvement highlights the value of including execution error data during training, allowing the model to generalize better to unseen error cases.

Similarly, in the BIRD benchmark, SEEING significantly outperforms the Simple SFT model. The Simple SFT model's accuracy increases from 41.1% to 43.7% after two corrections (+2.6%), while SEEING improves from 42.2% to 45.9% (+3.7%). This reinforces the utility of adding error-related data during training, particularly for challenging benchmarks like BIRD.

Comparing SEEING to other state-of-the-art models and methods like Self-Correction Guideline, Self-Debugging, Self-Consistency, and GPT-4, we find that SEEING surpasses all in the SPIDER dataset. SEEING outperforms GPT-4 in SPIDER with a final EX of 79.9%, compared to GPT-4's 79.5%. However, on the BIRD dataset, GPT-4 exhibits stronger reasoning capabilities, leading to a 11.5% performance gap in its favor. This difference is likely due to GPT-4's superior ability to utilize reasoning and evidence to handle more complex query requirements effectively.

In summary, SEEING delivers strong performance gains across both benchmarks, particularly in iterative error correction. Its ability to handle SQL execution errors more effectively than other models, including state-of-the-art prompting methods, demonstrates the advantages of fine-tuning on error-specific data.

5.2 SQL Execution Error Correction Performance

The experimental results, as presented in Table 3, highlight the effectiveness of our SEEING approach in addressing execution errors in SQL generation tasks. We report results based on two primary metrics: Fix Rate (FR) (Equation 1), which indicates the percentage of execution error cases corrected after the second iteration, and Weighted Fix Rate (WFR) (Equation 2), which weights cor-

Model	SPIDER			BIRD		
	Init	C1	C2	Init	C1	C2
W/o SFT	71.5	73.2	73.5	40.4	42.4	42.7
Debugging w/o SFT						
Self-Correction Guideline (Askari et al., 2024)	71.5	73.2	74.2	38.4	44.5	45.6
Self-Debugging (Chen et al., 2023)	71.5	72.7	73.1	38.4	42.4	42.5
Self-Consistency (Wang et al., 2022)	74.1	75.1	75.2	41.2	44.4	45.0
GPT-4	**79.1**	**79.5**	79.5	**55.3**	**57.3**	**57.4**
Debugging w/ SFT						
Simple SFT	75.5	76.3	76.3	41.1	42.7	43.7
SEEING **(ours)**	76.5	**79.5**	**79.9**	42.2	45.4	45.9
SEEING w/ analysis	76.6	78.8	79.0	41.7	45.2	46.0

Table 2: Execution accuracy (EX) across different models on the SPIDER and BIRD datasets. The table compares initial (Init) accuracy and the accuracy after the first (C1) and second (C2) error correction iterations. Models are evaluated with and without Self-Supervised Fine-Tuning (SFT). Benchmark models use instruction-tuned version of Llama-3.1-it (8B) as the base model. SEEING (ours) and SEEING w/ analysis are shown alongside other state-of-the-art models, including GPT-4, Self-Correction Guideline, Self-Debugging, and Self-Consistency. Bold text indicates the highest performance, while underlined numbers represent the second-highest performance.

Model	SPIDER		BIRD	
	FR	WFR	FR	WFR
W/o SFT	17.4	11.5	12.0	11.9
SEEING	**56.7**	**54.7**	23.8	23.1
SEEING w/ analysis	39.7	38.3	21.4	19.2
GPT-4	33.3	37.0	**34.3**	**31.8**

Table 3: SQL execution error correction results on the SPIDER and BIRD datasets. The table reports two metrics: Fix Rate (FR), which indicates the percentage of execution error cases corrected, and Weighted Fix Rate (WFR), which weights corrections based on query difficulty.

rections based on difficulty.

The SEEING model demonstrates a significant advantage in the SPIDER, correcting 56.7% of execution errors (FR), which is 39.3% higher than the baseline model without execution error-specific fine-tuning (W/o SFT). This indicates that pre-training models on datasets containing execution errors dramatically improves their ability to resolve such issues. Moreover, the model's performance in terms of WFR remains robust at 54.7%, showcasing only a minor drop of 1.4% from the raw FR. This minimal decrease suggests that SEEING not only handles simpler queries but also performs effectively on more challenging SQL queries, such as those requiring joins and nested structures.

In the BIRD dataset, the overall fix rates across all models decline, consistent with the fact that BIRD poses more difficult SQL generation challenges than SPIDER. Nevertheless, SEEING continues to perform strongly with an FR of 23.8% and a WFR of 23.1%. Though lower than SPIDER, SEEING maintains its lead over most other methods, proving its generalizability to harder datasets.

Interestingly, SEEING outperforms GPT-4 in the SPIDER dataset by a significant margin, achieving an FR that is 23.4% higher. This highlights that SEEING excels in resolving SQL errors that do not require sophisticated reasoning, likely benefiting from targeted fine-tuning on execution errors. However, when external reasoning capabilities are required, as seen in more complex queries, GPT-4 narrows this gap and even surpasses SEEING in the BIRD dataset (+10.5% in WFR), showing the advantage of large generalist models in reasoning-intensive tasks.

Furthermore, as shown in Figure 4, SEEING achieves significantly higher fix rates compared to the Simple SFT model, demonstrating the effectiveness of training on an execution error dataset.

Overall, SEEING delivers state-of-the-art performance in handling execution errors in SQL generation, particularly excelling in the SPIDER dataset and showing competitive results on the more challenging BIRD dataset. Its ability to maintain high performance across both simple and complex queries makes it a powerful tool for improving the reliability of SQL execution tasks.

5.3 Application of SEEING Framework

We evaluated the effectiveness of training error datasets on two popular open-source models,

Model	SPIDER	BIRD
Gemma2-it	80.5	48.0
Gemma2-it + SEEING	**81.1**	**49.2**
Mistral-it	79.4	42.0
Mistral-it + SEEING	**79.8**	**44.0**

Table 4: Execution accuracy of Gemma2-it (9B) and Mistral-it-v0.2 (7B) models on the SPIDER and BIRD datasets, with and without SEEING fine-tuning.

Gemma2-it (9B) and Mistral-it (7B) (Jiang et al., 2023), by reporting execution accuracy on the SPIDER and BIRD datasets. As shown in Table 4, we observe performance improvements across both models when incorporating the SEEING framework, which leverages error patterns from SQL queries.

For the Gemma2-it model, applying SEEING improves performance slightly on both SPIDER (from 80.5% to 81.1%) and BIRD (from 48.0% to 49.2%). This improvement, though modest, indicates that the error dataset generated by SEEING effectively aids in correcting query execution failures.

The Mistral model, which was not involved in generating the error dataset, also shows notable gains when fine-tuned with SEEING. Specifically, Mistral's performance on SPIDER increases from 79.4% to 79.8%, and its performance on BIRD improves from 42.0% to 44.0%. These gains, particularly on the BIRD dataset, demonstrate that the error dataset successfully generalizes beyond the models used for its creation. This highlights SEEING's ability to learn and apply SQL error patterns effectively, improving execution accuracy even for models that did not contribute to the error data generation process. These results validate that SEEING's fine-tuning methodology enhances the robustness of SQL query execution across different model architectures.

5.4 SQL Error Dataset Composition

As shown in Figure 2, our SQL error dataset is composed of four main error categories: Reference Errors, Syntax Errors, Function Misuse Errors, and Logical Errors (Figure 2). The largest portion of the dataset, 78%, consists of Reference Errors, which include issues like missing columns (57%), functions (11%), and tables (6%). These errors occur when queries reference non-existent or incorrectly named database objects. Syntax Errors account for 19% of the dataset, with common issues including general syntax mistakes (17%) and unrecognized tokens (1%).

Function Misuse Errors and Logical Errors make up smaller portions of the dataset, representing 2% and 1%, respectively. Function Misuse Errors occur when SQL functions are used incorrectly, such as misuse of aggregate function. Logical Errors, though rare, reflect flaws in query logic that result in incorrect outcomes. This distribution highlights the need for improving model robustness, particularly in handling references and syntax to reduce error frequency.

6 Conclusions

In this paper, we introduce SEEING, a novel approach that significantly improves the self-debugging capabilities of Text-to-SQL models. By developing a groundbreaking self-debugging dataset and applying targeted fine-tuning, we achieve state-of-the-art execution accuracies on both SPIDER and BIRD benchmarks, surpassing current prompting methods and even outperforming GPT-4 in some cases. Our approach demonstrates superior error correction abilities across query complexity levels and proved its generalizability by showing consistent improvements when applied to different model architectures.

Our work represents a significant step forward in developing models with robust self-debugging capabilities for Text-to-SQL tasks. By exposing the model to various types of SQL errors during training, SEEING enables it to not only correct execution errors when they arise but also anticipate and avoid common pitfalls in advance. This proactive approach to error handling enhances the overall reliability of SQL query generation. The release of our self-debugging dataset and accompanying code opens up new avenues for future research, potentially leading to more resilient and error-resistant Text-to-SQL systems that can adapt to complex, real-world querying scenarios.

7 Limitations

A key limitation of this study is the imbalance in the distribution of SQL execution errors within our dataset. The majority of errors are concentrated in Reference Errors, which account for 78% of the total, with "no such column" errors alone comprising 57%. In contrast, other categories, such as Logical Errors and Function Misuse Errors, contribute

only 1-2%. This uneven distribution may hinder the model's ability to generalize effectively, as errors that occur less frequently, such as "misuse of aggregate function" or "ambiguous column name," are underrepresented in training. Consequently, the model might perform well on common errors but struggle with less frequent ones, reducing its robustness when handling a broader range of SQL errors. Future work should address this imbalance through data augmentation or targeted sampling to enhance the model's ability to handle rare error types.

Furthermore, the process of generating a self-debugging dataset, which involves multiple steps of model prediction, error collection, correction via large language models (LLMs), and human validation, is resource-intensive. This limits the scalability of SEEING to other domains, especially for tasks that may require frequent updates or the handling of rapidly evolving data.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

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*.

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.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024. 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*.

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 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*.

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*.

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*.

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*.

A Appendix

Model	Init	C1	C2	C3	C4
W/o SFT	58	18	14	11	8
Simple SFT	46	28	23	23	18
SEEING	60	11	4	1	1
SEEING w/ analysis	63	12	6	4	2
GPT-4	23	4	2	2	2

Table 5: The number of execution errors across different models on the SPIDER dataset. 'Init' shows the initial errors, while 'C1' to 'C4' represent subsequent correction steps for reducing these errors.

Model	Init	C1	C2	C3	C4
W/o SFT	343	190	148	125	118
Simple SFT	284	234	204	189	175
SEEING	244	121	82	63	55
SEEING w/ analysis	313	142	89	66	47
GPT-4	188	56	33	20	16

Table 6: The number of execution errors across different models on the BIRD dataset.

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

[User prompt]
Generate a SQL question using the given table info. Provided row values are the first three rows of the table. Only return the question.

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
Generated question:

Figure 6: Template for generating SQL queries from table information, showing system and user prompts, along with the context structure.

[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}

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

Output
SQL query:

Figure 7: Template for generating an SQLite query using Llama-3.1-it (8B) and Gemma2-it (9B) models based on table information and a generated question.

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

[User prompt]
Given context, question, incorrect query, error message, fix SQL in the following format:

Analyze: *// Only apply to "with analysis" dataset*
Fixed SQL:
```sql
```

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}

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

Wrong SQL: {prediction_query}
Execution error: {execution_error_message}

Output
Fixed SQL query:

Figure 8: Template used by GPT-4 for correcting SQL queries with and without error analysis based on context, question, and execution errors.

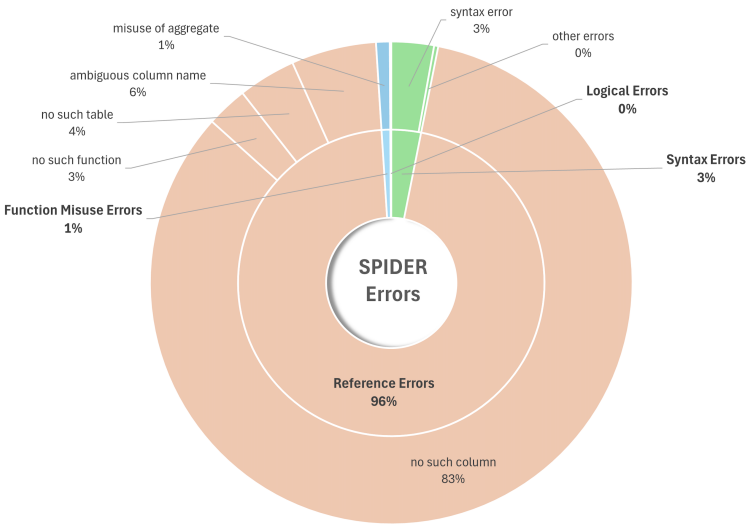

Figure 9: Donut chart representing the distribution of SPIDER SQL errors, categorized into reference, syntax, logical, and function misuse errors.

All Error Type	Error message	Number of errors
Syntax Errors		
	Syntax error	512
	Unrecognized token	23
	You can only execute one statement at a time	27
	Incomplete input	5
	ORDER BY clause should come after UNION ALL, not before	1
Reference Errors		
	No such column	1678
	No such function	337
	No such table	186
	Ambiguous column name	121
Function Misuse Errors		
	Misuse of aggregate	12
	Misuse of aggregate function	47
	Misuse of window function	4
	Wrong number of arguments to function	5
Logical Errors		
	1st ORDER BY term does not match any column in the result set	8
	Aggregate functions are not allowed in the GROUP BY clause	2
	SELECTs to the left and right of INTERSECT do not have the same number of result columns	1

Table 7: Summary of SQL Errors and their Frequencies (SPIDER and BIRD combined)

SPIDER	Error message	Number of errors
Syntax Errors		
	Syntax error	31
	Unrecognized token	1
	You can only execute one statement at a time	9
	ORDER BY clause should come after UNION ALL, not before	1
Reference Errors		
	No such column	908
	No such function	30
	No such table	42
	Ambiguous column name	62
Function Misuse Errors		
	Misuse of aggregate	3
	Misuse of aggregate function	7
	Wrong number of arguments to function	3
Logical Errors		
	SELECTs to the left and right of INTERSECT do not have the same number of result columns	1

Table 8: Summary of SPIDER SQL Errors and their Frequencies

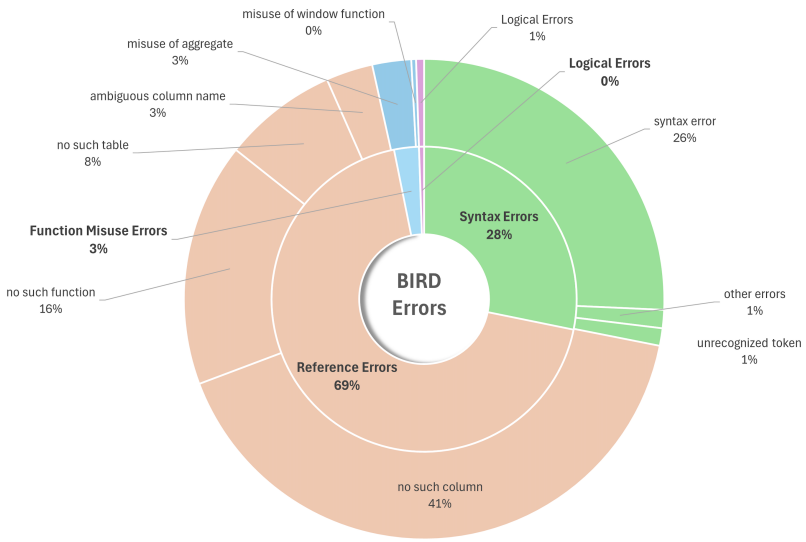

Figure 10: Donut chart representing the distribution of BIRD SQL errors, categorized into reference, syntax, logical, and function misuse errors.

BIRD	Error message	Number of errors
Syntax Errors		
	Syntax error	481
	Unrecognized token	22
	You can only execute one statement at a time.	18
	Incomplete input	5
Reference Errors		
	No such column	770
	No such function	307
	No such table	144
	Ambiguous column name	59
Function Misuse Errors		
	Misuse of aggregate	9
	Misuse of aggregate function	40
	Misuse of window function	4
	Wrong number of arguments to function	2
Logical Errors		
	1st ORDER BY term does not match any column in the result set	8
	Aggregate functions are not allowed in the GROUP BY clause	2

Table 9: Summary of BIRD SQL Errors and their Frequencies

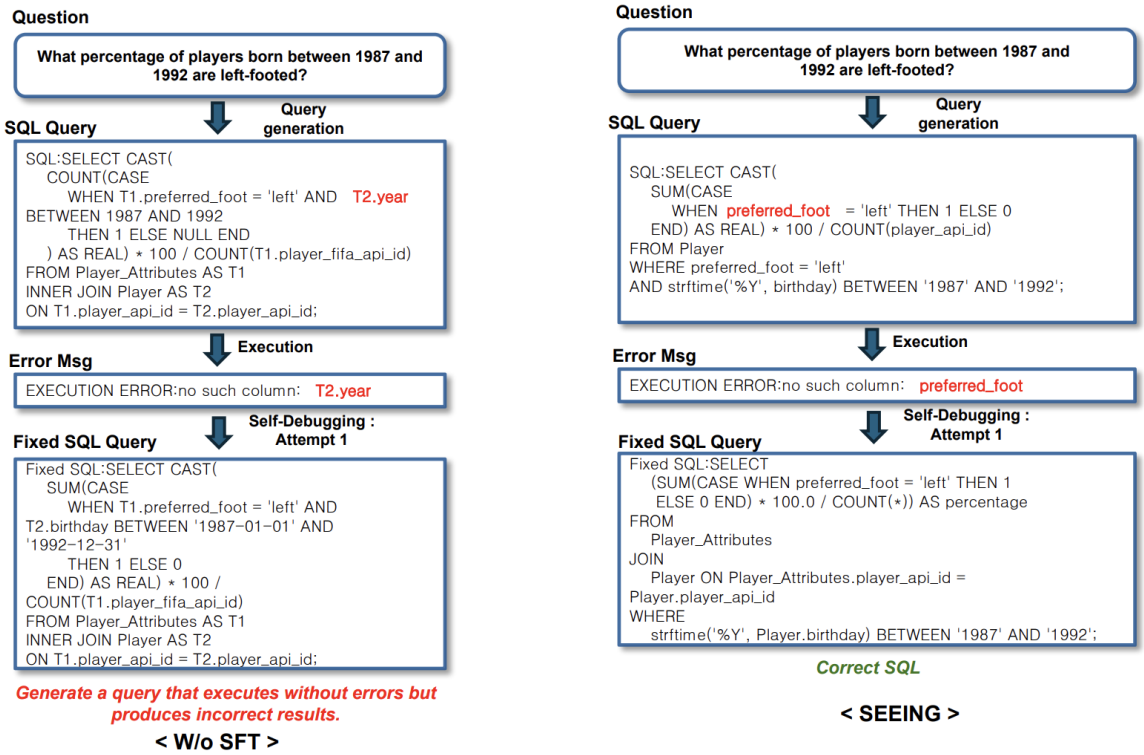

Figure 11: Comparison between the W/o SFT model and SEEING. In the W/o SFT model, although the execution error is resolved in the fixed SQL query, the result is incorrect compared to the gold query. In contrast, SEEING corrects both the execution error and aligns the output with the gold query, ensuring correct execution accuracy.

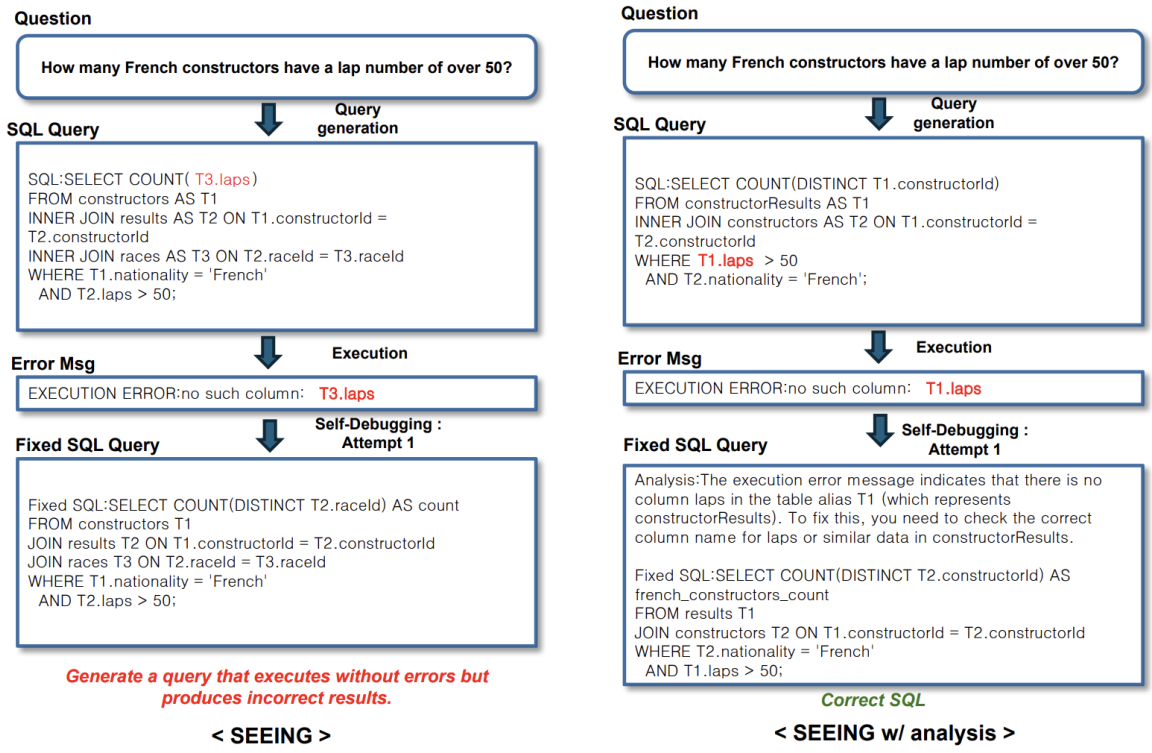

Figure 12: Comparison between SEEING and SEEING with analysis. In SEEING, although the execution error in the SQL query is fixed, the output is incorrect compared to the gold query. SEEING with analysis, however, corrects both the execution error and the output to match the gold query (ensuring correct execution and accuracy). While SEEING performs well overall, the analysis in SEEING with analysis sometimes provides additional reasoning that resolves errors which SEEING alone may not address.

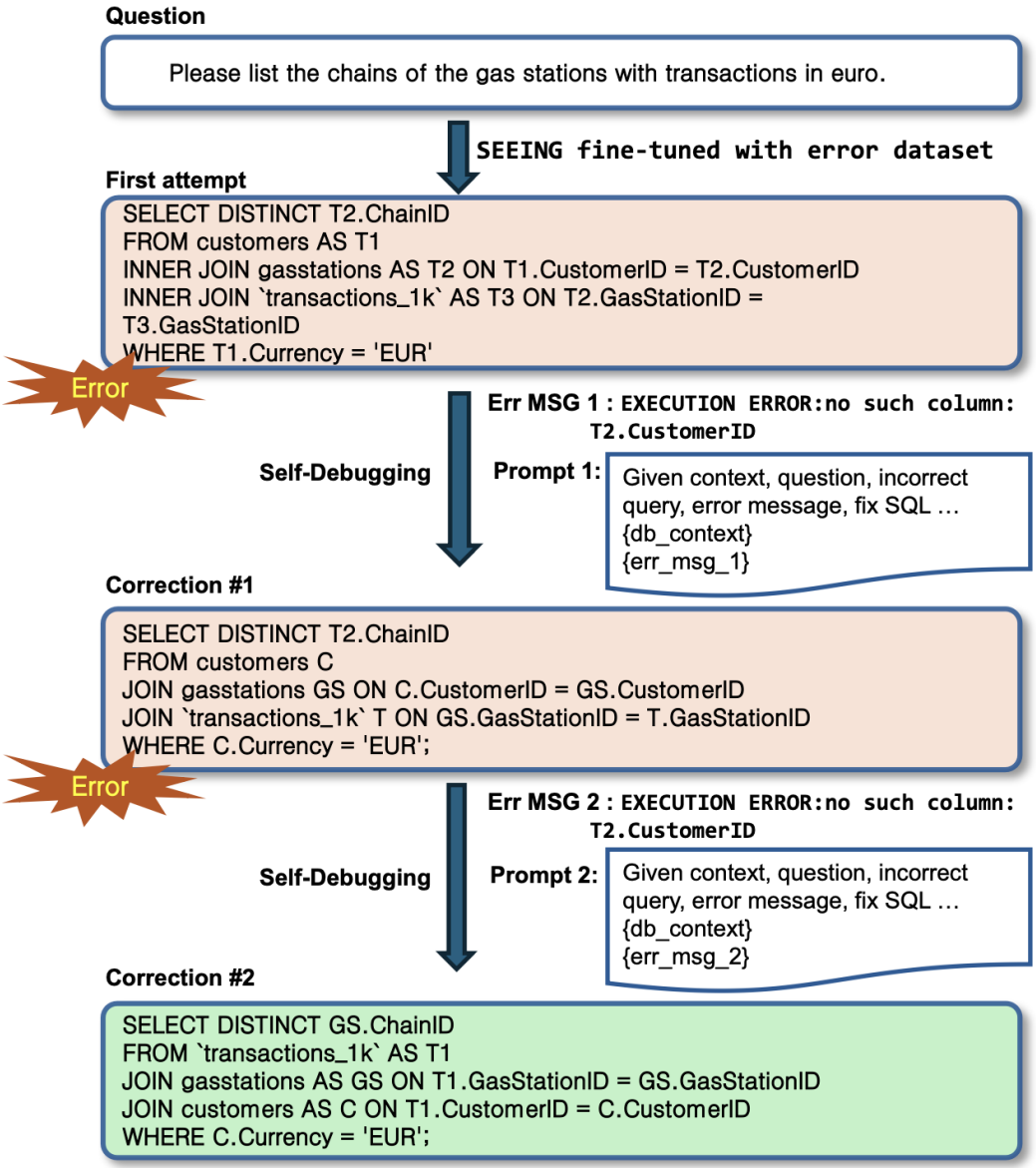

Figure 13: An example of SEEING in action, demonstrating how the model addresses execution errors in SQL queries. In the first attempt, the query generates an execution error due to the incorrect reference to T2.CustomerID. Through self-debugging, SEEING recognizes the error and attempts a correction. However, the first correction still results in an execution error, as the query continues to reference the wrong table for CustomerID. After a second round of debugging, SEEING correctly identifies the table that contains CustomerID and adjusts the query accordingly, successfully resolving the execution error. This example highlights the iterative debugging process within SEEING, fine-tuned with an error dataset, to achieve correct SQL query execution.