

Highlights

G²SQL: Guided & Guarded Text-to-SQL Generation with Two-Stage Verification

Xiang Li, Jinguo You, Heng Li, Jun Peng, Xi Chen, Ziheng Guo, Kun Li, Tianyi Xu

- Research highlight 1: This paper proposes a learning-based SQL-Plan Feedback Loop, a novel SQL generation approach designed to effectively address the problem of error propagation during the SQL generation process. The method first converts an initial SQL query into a structured SQL generation plan, consisting of multiple incremental steps. Each step includes a natural language description and its corresponding SQL segment. These steps are sequential and interdependent. The SQL in each step is executed on the database to obtain intermediate results, which are then combined with the step’s description and passed to an LLM for verification and refinement. If an error is identified in a particular step, that step and all subsequent steps are revised accordingly. The LLM is then instructed to regenerate the SQL based on the updated plan. This process repeats—transforming the regenerated SQL back into a plan and rechecking it—until the SQL passes all validations, ensuring both correctness and robustness in the final output.
- Research highlight 2: We propose a novel SQL verification framework, Reviewer-Observer, which enhances the effectiveness of SQL correctness checking while ensuring that the entire process remains safe and free from over-verification. In this framework, an LLM acting as the Reviewer is responsible for constructing test units to evaluate the correctness of a given SQL query. The test cases created by the Reviewer include two types: (1) natural language questions, which are answered using the LLM, and (2) SQL-based test queries, whose results are obtained by executing them on the database. To ensure that the SQL queries generated by the Reviewer do not harm the database or introduce redundant checks, another LLM serves as the Observer, overseeing the entire process. If the Observer detects any potentially harmful or duplicative test cases, it immediately halts their execution and provides feedback to the Reviewer to revise and regenerate safe SQL test

cases. The final decision on SQL correctness is jointly determined by the Reviewer and the Observer, thereby achieving reliable, secure, and efficient SQL validation.

- Research Highlight 3: We propose G²SQL, a highly efficient SQL generation framework that achieves performance comparable to state-of-the-art methods on both the BIRD and Spider datasets—while generating only a single candidate SQL per query. Extensive experiments across various types of LLMs demonstrate consistent performance improvements, and G²SQL shows strong competitiveness when compared with several advanced baselines. Notably, a token-level cost analysis reveals that G²SQL consumes the fewest tokens among all methods, highlighting its economic efficiency in addition to its effectiveness.

G²SQL: Guided & Guarded Text-to-SQL Generation with Two-Stage Verification

Xiang Li^a, Jinguo You^{a,*}, Heng Li^b, Jun Peng^a, Xi Chen^a, Ziheng Guo^a,
Kun Li^c, Tianyi Xu^c

^a*Faculty of Information Engineering and Automation, Kunming University of Science
and Technology, Kunming, 650500, Yunnan, China*

^b*School of Electronic Information, Central South
University, Changsha, 410083, Hunan, China*

^c*Beijing Huawei Digital Technologies Co., Ltd., Beijing, 100085, Beijing, China*

Abstract

Despite the remarkable progress of large language models (LLMs) in the Text-to-SQL domain, issues such as model hallucination remain a challenge. During SQL generation, an error at any stage may inevitably influence subsequent outputs, resulting in suboptimal or incorrect SQL queries. Moreover, when using LLMs to verify and revise generated SQL without human supervision, the models may sometimes introduce operations that tamper with or damage the database content. To address these challenges, we propose G²SQL, a supervised SQL generation framework enhanced by two-stage verification. In G²SQL, we adopt a learning-based SQL-Plan feedback loop to inspect and optimize the SQL generation process, aiming to minimize error propagation. After SQL is generated, a Reviewer-Observer mechanism is employed to further validate and revise the queries while ensuring database safety. Extensive experiments on both proprietary and open-source LLMs demonstrate the effectiveness and robustness of G²SQL, achieving a maximum execution accuracy of 73.16% on the BIRD development set and 89.97% on the Spider test set, verifying the efficiency and advancement of the proposed framework.

*Corresponding author

Email addresses: li_xiang@stu.kust.edu.cn (Xiang Li), jgyou@kust.edu.cn (Jinguo You), liheng@csu.edu.cn (Heng Li), junpeng@stu.kust.edu.cn (Jun Peng), chenxi328@stu.kust.edu.cn (Xi Chen), wzskqjh@stu.kust.edu.cn (Ziheng Guo), likun75@huawei.com (Kun Li), xutianyi5@huawei.com (Tianyi Xu)

Keywords: Text-to-SQL, Large Language Models, SQL-Plan Feedback Loop, Reviewer-Observer

1. Introduction

Traditional database query methods require users to possess a certain level of expertise, which hinders non-expert users from interacting with databases and thus restricts the broader development of database applications. To address these challenges, the technique of natural language to SQL (Text-to-SQL) has been proposed, and especially with the rapid advancement of large language models (LLMs), Text-to-SQL has seen wider adoption and significant progress.

LLMs are mainly applied in the Text-to-SQL field through two approaches: prompt engineering and supervised fine-tuning (SFT). In supervised fine-tuning methods, pre-annotated datasets are used to fine-tune open-source LLMs through techniques such as full-parameter fine-tuning [1], LoRA [2], and QLoRA [3], enabling LLMs to acquire domain-specific knowledge and generalization capabilities, with methods like CodeS [4] and OmniSQL [5] as representatives.

In prompt engineering methods, approaches such as DAIL-SQL [6] and DIN-SQL [7] fully leverage the capabilities of LLMs by carefully designing context structures that LLMs can better understand, thereby improving the quality of generated SQL. Methods like PET-SQL [8], RSL-SQL [9], ISESL-SQL [10], and Graphix-T5 [11] filter out irrelevant database schema information through schema linking before feeding the input into the LLM, thus reducing interference from unrelated information and enhancing SQL generation quality.

Additionally, some studies [12, 13, 14, 15] improve LLM performance by applying techniques like chain-of-thought prompting and few-shot prompting. Recently, researchers have also observed that LLMs are highly sensitive to prompts and that different LLMs behave differently when tackling specific problems. This has led to the development of multi-candidate generation-based Text-to-SQL methods, such as MCS-SQL [16], XiYan-SQL [17], CHESS [18], and CHASE [19].

Hallucination remains an intrinsic challenge for large language models (LLMs) [20], referring to the phenomenon where models generate content that deviates from user intent or factual correctness. This issue introduces





Case 1: Two examples of error propagation. Question: Which orders have at least 2 products on it? List the order id and date. Gold SQL: <code>SELECT T1.order_id , T2.date_order_placed FROM Orders AS T1 JOIN Order_items AS T2 ON T1.order_id = T2.order_id GROUP BY T1.order_id HAVING count(*) >= 2</code>	
(a) Database schema link error.	(b) Chain of thought generation error.
 Schema linking results: <code>[Orders.order_id , Orders.date_order_placed]</code>	 CoT: Identify the Tables and Columns: Start by focusing on the Orders table. You are interested in two specific columns: order_id and date_order_placed
Predict SQL: <code>SELECT Orders.order_id , Orders.date_order_placed FROM Orders GROUP BY Orders.order_id HAVING count(*) >= 2</code>	
Case 2: Two examples of errors in SQL accuracy checking process Question: List the address, town and county information of the customers who live in the USA. SQL: <code>SELECT address_line_1 , town_city , county FROM Customers WHERE Country = 'USA'</code>	
(a) Errors involving tampering with database information.	(b) Issues of repeated or redundant verification steps.
 Test unit: <code>INSERT INTO Customers (Country) VALUES ('USA')</code>	 Test unit 1: <code>SELECT address_line_1 , town_city , county FROM Customers LIMIT 1;</code> Test unit 2: <code>Does the `Customers` table contain the columns `address_line_1`, `town_city`, and `county`?</code>

Fig. 1. Two error cases in the Text-to-SQL domain, where Case 1 illustrates issues caused by error propagation, and Case 2 presents potential problems arising during SQL accuracy checking.

new challenges for LLM-based Text-to-SQL approaches, making it critical to explore strategies that minimize the impact of hallucination on SQL generation. Existing research has proposed two main directions to address this problem. One line of work mitigates hallucination by generating intermediate SQL representations or chain-of-thought reasoning steps and then producing the final SQL based on these intermediates. Another line of work relies on developing a large set of SQL candidates, applying self-consistency correction mechanisms, and selecting the optimal candidate to counteract hallucination effects. However, approaches based on intermediate representations or reasoning steps are prone to error propagation: errors introduced at any stage can cascade throughout the entire SQL generation process, as illustrated in Case 1 of Figure 1, where (a) demonstrates error propagation caused by in-

correct schema linking, and (b) shows error propagation stemming from mistakes within the chain-of-thought reasoning. In contrast, candidate-based selection methods, while achieving promising performance in recent state-of-the-art models, require generating a vast number of SQL candidates and conducting extensive correction operations. Such methods incur significant computational and resource costs, rendering them impractical for real-world deployment.

Moreover, it is important to note that, due to the aforementioned hallucination issues, LLMs may generate SQL manipulation statements during the processes of SQL generation or accuracy testing, which could alter database contents or corrupt database structures. If executed, such SQL statements could severely compromise database security, as illustrated in Case 2(a) of Figure 1, to validate the correctness of a WHERE clause, the LLM attempts to insert specific values into the database to satisfy the filter condition. However, this approach inevitably alters and potentially damages the original content of the database, raising concerns about data integrity.. Furthermore, our experiments reveal that during the generation of test units for SQL validation, LLMs sometimes produce redundant tests or generate test cases whose objectives have already been addressed in previous validations. Repeated occurrences of such redundancy can lead to unnecessary resource consumption and introduce excessive noise into the validation process, ultimately complicating the final decision-making, as shown in Case 2 (b) of Figure 1.

To address the aforementioned issues, we propose G²SQL, a Text-to-SQL framework that incorporates a two-stage verification and correction mechanism before and after SQL generation. Moreover, an observer LLM is introduced to monitor the SQL verification process in real time, ensuring that the entire procedure is both safe and effective. Specifically, to mitigate the impact of LLM hallucinations during SQL generation, we refine the generation process into two stages. In the first stage, SQL generation is decomposed into a step-by-step procedure, where each step is examined and corrected to prevent early-stage errors from propagating and affecting subsequent steps. In the second stage, after the SQL statement is produced, another LLM is tasked with generating diverse test cases—including natural language questions and SQL test queries—to validate the correctness of the generated SQL. These test cases are executed using dedicated tools to collect feedback for further correction. To prevent the generation of harmful test cases—such as those that attempt to manipulate the database content—or redundant tests,

we introduce an observer-role LLM to oversee the entire process, interrupt any unsafe or unproductive actions, and provide appropriate guidance.

We conducted extensive experiments on the BIRD and Spider datasets and validated the effectiveness of our proposed G²SQL using both open-source and closed-source LLMs. The experimental results demonstrate the effectiveness of G²SQL. Our main contributions are summarized as follows:

- We propose a Learning-based SQL-Plan Feedback Loop method that validates and corrects each step of SQL generation before the SQL is finalized, effectively reducing the issues caused by error propagation.
- We introduce a Reviewer-Observer-based SQL accuracy checking method, which carefully inspects and validates SQL queries under supervision. This method improves the effectiveness of SQL verification while ensuring that the process does not compromise database security.
- Our method achieves performance close to state-of-the-art levels, with an execution accuracy of 73.16% on the development set of the BIRD dataset and 89.97% on the test set of the Spider dataset, even when generating only a single SQL query.

2. Related Work

2.1. Schema Link

Schema linking refers to extracting relevant information, such as tables and columns, from the database schema before SQL generation to simplify the schema and avoid interference from redundant information. TA-SQL [21] and PET-SQL [8] use prompt engineering first to have LLMs generate an initial SQL query, then extract relevant tables and columns from it to achieve schema linking. C3 [22] and CHESS [18] employ a two-stage schema linking approach, where LLMs are first instructed to select the most relevant tables for the query and then, based on the chosen tables, to select the corresponding columns. Additionally, methods like [23] and Graphix-T5 [11] use pre-trained models, such as T5, to perform schema linking before SQL generation, selecting out the database schemas required for the SQL query.

Although the previous method has made some progress in the database schema linking task, there is currently no approach that can identify all relevant database schemas. If a schema is missing or an incorrect schema is generated, it will impact the accuracy of subsequent SQL generation.

2.2. Prompt Engineering Techniques

Prompt engineering plays a crucial role in guiding LLMs to perform specific tasks, and the quality of the prompt directly impacts the model’s performance. In recent years, various prompting techniques have emerged to improve LLM performance in text-to-SQL applications. Based on the selection of text-to-SQL examples, prompting methods are generally categorized into zero-shot and few-shot approaches. Studies such as [24, 15, 25] adopt zero-shot prompting, where no examples are provided to the LLM, with the focus placed on other aspects of the task. On the other hand, [7, 26] uses few-shot prompting, retrieving relevant examples to help the model learn the solution patterns from those examples and apply them to new tasks. Additionally, in the text-to-SQL field, methods that guide LLMs to perform multi-step reasoning have been explored to progressively refine the generated SQL towards the correct answer. This includes techniques such as Chain-of-Thought (CoT) prompting [27], Least-to-Most prompting [28], and self-consistency methods [29].

2.3. SQL Check and Modification

In the field of text-to-SQL, the evaluation of SQL query correctness and their correction is an important research topic. Currently, some studies [30] assess the accuracy of SQL queries by comparing database execution results with the self-consistency of LLM and then input erroneous SQL queries into the LLM for correction. Other works [31] leverage database execution error messages, inputting both the error messages and the erroneous SQL into the LLM for modification, and finally use a voting mechanism across multiple models to select the optimal SQL query. However, these existing studies generally rely on the self-consistency of LLMs to judge the correctness of SQL queries. Moreover, during the modification process, only SQL queries that produce execution errors are adjusted, while semantic errors are not effectively identified or corrected. Typically, errors are handled by simply regenerating SQL queries, lacking targeted optimization strategies.

3. Methods

3.1. Overall Framework

We proposed G²SQL framework is composed of three core components: (1) Formulate an SQL generation plan, (2) SQL Correctness Verification, and

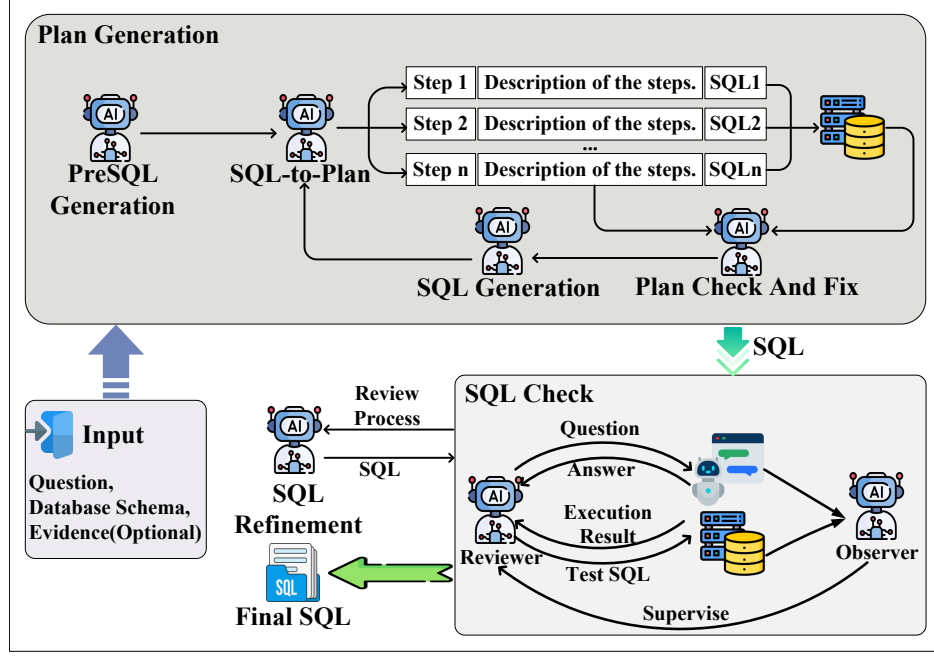


Fig. 2. G²SQL Overall Architecture. G²SQL consists of three main modules: (1) SQL Generation Planning Module, which generates an optimal SQL generation plan through step-by-step checking and refinement; (2) SQL Accuracy Verification Module, where a Reviewer LLM constructs test cases and verifies them using appropriate tools, while an Observer LLM monitors the entire verification process in real time; (3) SQL Refinement Module, which modifies the SQL statements based on the outputs from the verification module.

(3) SQL Refinement. G²SQL begins by employing a large language model (LLM) to generate an initial SQL query. Next, the SQL Generation Planning Module constructs a detailed generation plan based on this initial query. This plan is then provided to the LLM to generate a candidate SQL query for verification. The SQL Correctness Verification Module carefully examines the candidate query by constructing safe and effective test cases to identify potential issues. If any errors are detected, both the problematic query and the identified problems are passed to the SQL Refinement Module for correction. Through iterative interaction between the Verification and Refinement Modules, G²SQL produces a final, correct SQL query. The overall architecture of G²SQL is shown in Figure 2. The overall algorithm flow is shown in Algorithm 1. To provide a clearer understanding of the complete SQL generation process of G²SQL, we present a detailed case study in [Appendix](#)

B. The prompt templates used in this work are provided in [Appendix C](#).

Algorithm 1 The algorithm of G²SQL

Require: Question q , database schema sc , knowledge k (Optional)

Ensure: SQL query sql

```

1:  $sql \leftarrow \text{LLM}(q, sc, k)$ 
2:  $plan \leftarrow \text{SQLtoPlan}(sql)$ 
3: while  $\neg \text{planCheck}(plan)$  do
4:    $plan \leftarrow \text{planFix}(plan)$ 
5:    $sql \leftarrow \text{PlantoSQL}(plan)$ 
6:    $plan \leftarrow \text{SQLtoPlan}(sql)$ 
7: end while
8:  $isTrue \leftarrow \text{False}$ 
9: while  $\neg isTrue$  do
10:   $isTrue, checkProcess, fixSuggest \leftarrow \text{sqlCheck}(sql)$ 
11:  if  $\neg isTrue$  then
12:     $sql \leftarrow \text{sqlModification}(sql, checkProcess, fixSuggest)$ 
13:  end if
14: end while

```

3.2. Formulate an SQL Generation Plan

In this stage, we first input the initial SQL query and the database schema into the LLM, guiding it to decompose the query into a detailed generation plan. Each step in the plan includes a textual description explaining the purpose and function of the step and a corresponding SQL statement. We then execute each step’s SQL query on the target database to obtain feedback. For queries that return lengthy results, we retain only the first five rows as the database feedback.

To minimize errors in the SQL generation plan, we adopt a learning-based SQL-Plan Feedback Loop to iteratively check and refine the plan. Specifically, we first input the SQL generation plan along with the execution results of each step into the LLM, prompting it to analyze the feasibility and correctness of each step. The LLM then identifies and corrects any erroneous steps in the plan. Next, the revised SQL generation plan is provided to the LLM to generate a corresponding SQL query. This query is then transformed back into an SQL generation plan and re-evaluated using the same process.

Algorithm 2 The algorithm of Generate Plan

Require: Question q , database schema sc , knowledge k (Optional)

Ensure: SQL query sql

```
1:  $exeResult \leftarrow []$ 
2:  $sql \leftarrow \text{LLM}(q, sc, k)$ 
3:  $plan \leftarrow \text{LLM}(q, sc, k, sql)$ 
4:  $isFinish \leftarrow \text{False}$ 
5: while  $\neg isTrue$  do
6:   for  $step$  in  $plan$  do
7:      $exeResult.append(\text{DB}(step["SQL"]))$ 
8:   end for
9:    $isTrue, errorPart \leftarrow \text{LLM}(q, sc, k, plan, exeResult)$ 
10:  if  $\neg isTrue$  then
11:     $plan \leftarrow \text{LLM}(q, sc, k, plan, errorPart)$ 
12:     $sql \leftarrow \text{LLM}(q, sc, k, plan)$ 
13:  end if
14: end while
```

The loop continues until the SQL generation plan is deemed correct, at which point the corresponding SQL query is finalized and output.

During the process of checking and refining the SQL generation plan, in order to prevent the LLM from repeatedly making the same mistakes due to hallucinations or other issues, we record the process and results of each modification. These historical records are then provided to the LLM in the subsequent rounds of checking and refinement, encouraging it to learn from past revisions and avoid generating the same errors repeatedly.

Through the above steps, potential issues in the SQL generation process are significantly mitigated, reducing the likelihood of generating incorrect SQL statements caused by the propagation of errors. Moreover, the entire SQL generation process becomes transparent and traceable, which further enhances the interpretability of the generated SQL queries. The algorithm flow of this module is shown in Algorithm 2.

3.3. SQL Check

To maximize the accuracy of generated SQL queries, we propose a supervised heuristic verification method named Reviewer-Observer, which performs SQL correctness checking. In this framework, the SQL query to be

Algorithm 3 The algorithm of SQL Check

Require: Question q , database schema sc , knowledge k (Optional), SQL query sql

Ensure: Boolean $isTrue$, list $history$

```
1:  $isTrue \leftarrow \text{False}$ 
2:  $history \leftarrow []$ 
3:  $feedback \leftarrow ''$ 
4: while  $\neg isTrue$  do
5:    $testExample, isFinish, isTrue1 \leftarrow \text{LLM1}(q, sc, k, sql, feedback, history)$ 
6:    $isEfficient, feedBack, isTrue2 \leftarrow \text{LLM2}(testExample, history)$ 
7:   if  $isEfficient$  and  $\neg isFinish$  then
8:     if  $testExample$  is  $sql$  then
9:        $exeResult \leftarrow \text{DB}(sql)$ 
10:       $history.append([testExample, exeResult])$ 
11:    else
12:       $answer \leftarrow \text{LLM3}(testExample)$ 
13:       $history.append([testExample, answer])$ 
14:    end if
15:  end if
16:   $isTrue \leftarrow isTrue1 \text{ and } isTrue2$ 
17: end while
```

verified is first passed into the module, where an LLM acting as the Reviewer is responsible for generating corresponding test cases. These test cases include two types: (1) natural language questions related to the logic of the SQL query, and (2) SQL statements used for testing. For the natural language questions, another LLM focused on question answering and reasoning is invoked to provide both answers and explanatory justifications. For the SQL-based test cases, the system executes them against the corresponding database to collect feedback.

To ensure the validity and safety of the generated test cases, an additional LLM serves as an Observer to monitor each step of the verification process in real time. Due to potential hallucinations or contextual misunderstandings, the Reviewer may generate redundant or overlapping natural language questions, which can lead to unnecessary resource consumption and introduce noise that complicates final decision-making. Furthermore, the Reviewer may produce CREATE or INSERT statements in an attempt to verify

the correctness of the original SQL. While such statements may provide useful test signals, they also pose serious risks by modifying or corrupting the original database contents. When such issues arise, the Observer intervenes promptly to halt the action and provide feedback to the Reviewer, thereby mitigating both resource waste and the risk of data corruption.

After several rounds of testing, the final decision regarding SQL correctness is jointly made by the Reviewer and Observer. To adopt a conservative strategy that prioritizes reliability, we apply a logical AND operation on their judgments—that is, a query is marked as correct only if both models independently deem it correct; otherwise, it is considered erroneous. For queries judged to be incorrect, the system records the entire verification process and generates suggestions for modification, which serve as input to the downstream correction module. To avoid infinite loops and excessive resource usage, we limit the entire verification process to at most 3 iterations, with each iteration consisting of no more than 10 rounds of testing. The algorithm flow of this module is shown in Algorithm 3.

3.4. SQL Refinement

This module is designed to handle SQL queries that contain errors or require optimization. For erroneous SQL queries, we first retrieve their corresponding testing processes from the SQL checking module. To minimize the potential impact of excessive intermediate information on model performance, we extract only the relevant test cases that exposed the errors, along with the correction or optimization suggestions provided by the checking module, as the basis for refinement.

To prevent the model from repeatedly making the same mistakes during multiple rounds of refinement, we incorporate a history of previous modifications before each new editing step, allowing the LLM to learn from past revisions. Finally, through iterative interaction with the checking module, the refined and correct SQL query is generated.

4. Experiments

4.1. Datasets

We evaluate the effectiveness of the proposed G²SQL on two major Text-to-SQL benchmarks: Spider [32] and BIRD [33].

Spider is a widely used benchmark for evaluating Text-to-SQL systems. It consists of a training set with 8,659 samples, a development set with 1,034

samples, and a test set with 2,147 samples. The dataset spans 200 different databases across 138 distinct domains, requiring Text-to-SQL models to demonstrate strong generalization and adaptability to unseen domains and unfamiliar database schemas.

The BIRD dataset, released by Alibaba DAMO Academy, is a large-scale benchmark specifically designed for the Text-to-SQL task. It contains 95 large-scale databases and a collection of high-quality natural language-SQL pairs, with a total data volume of up to 33.4 GB, spanning 37 professional domains. Compared to Spider, BIRD focuses more on integrating external knowledge, logical reasoning, and arithmetic operations, strengthening the connection between natural language questions and database content. It presents new challenges for Text-to-SQL systems in handling large databases and performing complex reasoning.

4.2. Evaluation Metrics

Following the evaluation protocols of the BIRD and Spider test suites [34], we adopt Execution Accuracy (EX) as the primary metric to assess the effectiveness of our proposed framework. Execution Accuracy is the proportion of predicted SQL queries whose execution results exactly match those of the corresponding gold SQL queries, relative to the total number of test samples.

4.3. Baseline Model

To evaluate the generalization ability of G²SQL across different types of models, we conducted experiments on both open-source and proprietary commercial LLMs. Among the open-source models, we selected DeepSeek-V3 [35] and Qwen-2.5-coder-32B [36], as they are widely adopted and represent different parameter scales, allowing us to assess performance across varying model capacities. For proprietary models, we evaluated G²SQL on Gemini-2.0-flash, Gemini-2.0-flash-exp, GPT-4.1-mini [37], and GPT-4o [38]. Specifically, Gemini-2.0-flash and Gemini-2.0-flash-exp are two variants of the same model family, differing in capabilities, and we include both to examine performance variance under varying model strengths. GPT-4.1-mini represents a newer generation of OpenAI’s models, while GPT-4o is a widely adopted, high-performance model. We include experiments on GPT-4.1-mini to highlight the effectiveness of our method on cutting-edge LLMs, and use GPT-4o to maintain consistency with baseline comparisons used in prior work.

It is important to note that, unless otherwise specified, Gemini-2.0-flash serves as the default backbone model throughout our experiments. This choice is motivated by its favourable balance of performance and cost, making it a practical and effective foundation for evaluating our approach.

Table 1: Execution accuracy of G²SQL on the BIRD development set across different models and difficulty levels.

Model	Simple	Moderate	Challenging	Total
Count	925	464	145	1534
Qwen-coder-32b	60.43	45.26	35.86	53.52
G²SQL + Qwen2.5-coder-32b (ours)	72.03	58.84	59.61	66.87(↑13.35)
DeepSeek-V3	67.78	48.92	42.76	59.71
G²SQL + DeepSeek-V3 (ours)	73.56	62.67	65.15	69.47(↑9.76)
GPT-4.1-mini	62.27	41.59	40.69	53.97
G²SQL + GPT-4.1-mini (ours)	72.71	60.27	63.25	68.05(↑14.08)
Gemini-2.0-flash	70.27	53.45	46.90	62.97
G²SQL + Gemini-2.0-flash (ours)	74.92	64.66	68.97	71.25(↑8.28)
Gemini-2.0-flash-exp	72.16	55.23	47.33	64.69
G²SQL + Gemini-2.0-flash-exp (ours)	75.12	67.10	69.33	72.15(↑7.45)
GPT-4o	72.3	55.12	47.1	64.72
G²SQL + GPT-4o (ours)	76.43	67.52	70.4	73.16(↑8.44)

Table 2: Execution accuracy of G²SQL on the Spider test set across different models and difficulty levels.

Model	Easy	Medium	Hard	Extra	Total
Count	470	857	463	357	2147
Qwen2.5-coder-32b	90.40	85.60	78.00	67.20	82.00
G²SQL + Qwen-coder-32b (ours)	93.60	90.40	83.50	79.60	87.82 (↑5.82)
Gemini-2.0-flash	94.00	90.00	83.60	75.90	87.10
G²SQL + Gemini-2.0-flash (ours)	94.20	92.50	86.70	81.20	89.74 (↑2.64)
GPT-4o	94.30	91.40	85.40	79.80	88.81
G²SQL + GPT-4o (ours)	94.80	92.10	86.40	83.10	89.97 (↑1.16)

4.4. Result

BIRD Result. We evaluated the execution accuracy of G²SQL on the development set of the BIRD dataset using different baseline models for the

Table 3: Execution accuracy (EX) comparison of different methods on the BIRD development set.

Method	EX (%)
CHASE-SQL + Gemini-1.5 pro [19]	74.90
CHESS [18]	68.31
E-SQL + GPT-4o [39]	65.58
MCS-SQL + GPT-4 [16]	63.36
OpenSearch-SQL-v2 + GPT-4o [40]	69.30
RSL-SQL + GPT-4o [9]	67.21
XiYan-SQL [17]	73.34
OmniSQL-32B [5]	69.23
Distillery + GPT-4o [41]	67.21
G²SQL + Gemini-2.0-flash (ours)	71.25
G²SQL + Gemini-2.0-flash-exp (ours)	72.15
G²SQL + GPT-4o (ours)	73.16

Table 4: Execution accuracy (EX) comparison of different methods on the Spider test set.

Method	EX (%)
MCS-SQL + GPT-4 [16]	89.60
CHASE-SQL + Gemini-1.5 [19]	87.60
DAIL-SQL + GPT-4 [6]	86.60
XiYan-SQL [17]	89.65
DIN-SQL + GPT-4 [7]	85.30
PET-SQL [8]	87.60
C3 + ChatGPT + Zero-Shot [22]	82.30
Tool-SQL + GPT-4 [42]	85.60
G²SQL + Gemini-2.0-flash (ours)	89.74
G²SQL + GPT-4o (ours)	89.97

Text-to-SQL task, as shown in Table 1. Compared to the open-source baseline model Qwen-coder-32B, our approach achieves an overall improvement of 13.35% in execution accuracy. Notably, G²SQL consistently outperforms the baseline across all difficulty levels, with improvements of at least 11%, demon-

strating its strong adaptability and effectiveness when built upon open-source models.

Furthermore, G²SQL exhibits even stronger performance when applied to proprietary or large-parameter LLMs. Specifically, it outperforms DeepSeek-V3 by 9.76%, GPT-4.1-mini by 14.08%, and GPT-4o by 8.44%. Compared to Gemini-2.0-flash and Gemini-2.0-flash-exp, G²SQL achieves improvements of 8.28% and 7.45%, respectively. These results indicate that G²SQL offers excellent generalization ability and robustness across a wide range of LLMs.

In addition, Table 3 presents a comparison between our method and several state-of-the-art approaches on the BIRD dataset in terms of execution accuracy. As shown, our method outperforms most existing methods. Notably, compared to advanced techniques such as XiYan-SQL and CHASE, which rely on generating a large number of SQL candidates and making multiple LLM calls to boost accuracy, G²SQL achieves comparable performance while generating only a single SQL query. This highlights the efficiency and competitiveness of our approach, demonstrating that G²SQL can deliver strong results with significantly reduced computational overhead.

Spider Result. To further evaluate the performance of G²SQL across different datasets, we conducted experiments on the test set of the Spider dataset. As shown in Table 2, evaluations were carried out using one open-source LLM and two proprietary LLMs. When built upon the open-source model Qwen-coder-32B, our approach achieved an execution accuracy of 87.82%, representing a 5.82% improvement over the baseline. For proprietary models, G²SQL reached 89.74% accuracy with Gemini-2.0-flash and 89.97% with GPT-4o, surpassing the corresponding baselines by 2.64% and 1.16%, respectively.

It is worth noting that Table 4 compares G²SQL with several state-of-the-art methods on the Spider dataset, showing that our approach performs on par with the best-performing systems. These results further confirm the robustness and effectiveness of G²SQL across different scenarios and benchmarks.

4.5. Effectiveness of Two-Stage Validation

Our proposed G²SQL primarily employs a two-stage verification approach to check and refine the generated SQL statements. To quantitatively assess the effectiveness of the two core components—the SQL-Plan Feedback Loop and the Reviewer-Observer-based SQL accuracy verification method—in identifying and correcting erroneous SQL statements, we conduct experiments on

Table 5: SQL accuracy identification on the BIRD dev set: Reviewer-Observer vs. Fine-tuning and Self-consistency.

		Syntactic Error(%)	Semantic Error(%)	Correct(%)	Overall(%)
Method	Total	73	554	907	1534
SFT(Qwen2.5-coder-7b)		100	68.2	63.92	67.18
Self-Consistency		86.3	16.1	82.4	58.64
Reviewer-Observer		100	77.85	88.67	85.3

Table 6: Effectiveness of different methods in fixing different types of SQL errors on the BIRD dev set.

Method	Syntax Error (%)	Semantic Error (%)
Self-Consistency	65.7	21.8
SQL-Plan Feedback Loop	92.3	63.3
Reviewer-Observer + Refinement	89.2	75.9
Two-Stage Verification	94.6	80.2

the development set of the BIRD benchmark. Table 5 presents a comparison of the SQL accuracy verification performance of our Reviewer-Observer method against two mainstream SQL verification approaches across different SQL correctness categories. Table 6 shows the error correction performance of the two proposed methods individually, as well as the two-stage verification approach that combines both methods. We also compare these results with the model-based Self-Consistency correction approach.

To better illustrate the effectiveness of each method in handling different types of SQL errors, we categorize SQL errors into syntactic errors and semantic errors. Syntactic errors refer to violations of SQL grammar that prevent successful compilation or execution by the database engine. In contrast, semantic errors refer to SQL statements that are syntactically correct and executable but fail to reflect the user’s intended query logic.

Based on this categorization, we evaluate and compare each method. In Table 5, we present the performance of our Reviewer-Observer-based verification method compared with two baselines: a supervised fine-tuned model (based on Qwen2.5-coder-7b, with fine-tuning details provided in Appendix A) and the Self-Consistency approach. As shown in the table, our method significantly outperforms both baselines in detecting syntactic errors, semantic errors, and correctly written SQL, as well as in overall accuracy. These

results validate the effectiveness and superiority of our proposed SQL verification approach.

Table 6 provides a comparison of the correction performance across methods. We show the error correction capabilities of the Self-Consistency approach and each component in G²SQL. The results demonstrate that both of our proposed verification strategies are more effective than the Self-Consistency method in correcting syntactic and semantic errors. Moreover, the combined two-stage verification method achieves the best performance overall. This further confirms the effectiveness of each module in G²SQL and highlights the necessity of integrating both strategies into a unified verification framework.

Table 7: Ablation results of G²SQL with different components on the BIRD development set.

Method	Simple(%)	Moderate(%)	Challenging(%)	Total(%)
G ² SQL + Qwen-coder-32b	72.03	58.84	59.61	66.87
w/o Plan Generation	68.78	53.44	47.58	62.14(↓4.73)
w/o Check & Refinement	70.36	54.20	52.18	63.75(↓3.12)
G ² SQL + Gemini-2.0-flash	74.92	64.66	68.97	71.25
w/o Plan Generation	71.56	62.31	61.03	67.77(↓3.49)
w/o Check & Refinement	72.98	63.22	65.15	69.29(↓1.97)

Table 8: Performance of different components in the Reviewer-Observer framework on the BIRD dev set: preventing harmful SQL generation and reducing test case redundancy.

Method	Harmful SQL Count			Test Case Count		
	Simple	Moderate	Challenging	Simple	Moderate	Challenging
Reviewer-Observer	0	0	0	2.10	4.50	7.90
w/o Observer	2.3	7.1	13.5	6.50	11.70	24.20
w/o Question-Answer	0	0	0	4.30	9.60	9.80
w/o Writing Test SQL	0	0	0	8.50	11.40	13.30

4.6. Ablation Study

To assess the contribution of each component in the G²SQL architecture to the overall performance, we conduct ablation studies on an open-source model (Qwen-coder-32B) and a closed-source model (Gemini-2.0-flash). As shown in Table 7, for the open-source model, removing the SQL plan generation module results in a 4.73% drop in execution accuracy, while removing

Table 9: Performance of different components in the Reviewer-Observer framework on identifying different types of SQL correctness on the BIRD dev set.

		Syntactic Error(%)	Semantic Error(%)	Correct(%)	Overall(%)
Method	Total	73	554	907	1534
Reviewer-Observer		100	77.85	88.67	85.3
w/o Observer		100	62.13(↓15.72)	79.5(↓9.17)	74.2(↓11.1)
w/o Question-Answer		100	74.33(↓3.52)	82.7(↓5.97)	80.5(↓4.8)
w/o Writing Test SQL		97.25(↓2.75)	53.07(↓24.78)	86.89(↓1.78)	75.17(↓10.13)

the SQL checking and revision module leads to a 3.12% decrease. These results indicate that each component in our architecture contributes positively to performance when applied to open-source models. Similarly, on the closed-source model, excluding the SQL plan generation module causes a 3.49% drop in performance, and removing the checking and revision module results in a 1.97% decrease. This further demonstrates that each part of the G²SQL architecture is necessary and effective, regardless of the underlying model type.

The ablation results also reveal that when using smaller-parameter models, the two-stage validation and revision approach can maximize the model’s potential. Relying solely on either validation or revision alone fails to achieve optimal performance. In contrast, larger LLMs, due to their stronger inherent programming and reasoning capabilities, are less affected by the removal of the checking and revision module. However, they show greater sensitivity to changes in the planning and verification stage. Nevertheless, each component of our proposed G²SQL architecture contributes positively to both categories of LLMs, which demonstrates its adaptability and robustness across a wide range of model types and capabilities.

To further validate the advantages of our proposed Reviewer-Observer-based SQL accuracy verification method in database safety protection and reduction of test redundancy, we conducted a detailed ablation study on the development set of the BIRD benchmark.

As shown in Table 8, when the Observer component is removed (i.e., the entire SQL testing process is not monitored), the test cases generated by the Reviewer often include SQL statements that pose a risk to database safety across all SQL difficulty levels. Moreover, the number of harmful SQL statements increases with the complexity of the SQL query. In contrast, when the Observer is included, no harmful SQL statements are produced, indicat-

ing that this component effectively safeguards database security during the verification process.

In addition, we compared the number of test cases required to verify the correctness of a single SQL statement when removing each of the three modules in the Reviewer-Observer framework. As shown in Table 8, the removal of the Observer component leads to a dramatic increase in the number of required test cases, especially for the most complex SQL queries. When the Observer is present, the number of test cases is reduced by at least half, highlighting its effectiveness in avoiding unnecessary test generation and reducing model inference costs. Notably, removing either the Question-Answer module or the Test SQL generation module also results in more test cases being generated compared to the complete Reviewer-Observer system, indicating that both modules contribute to reducing test redundancy.

In Table 9, we present the impact of removing each module in the Reviewer-Observer framework on the ability to identify different types of SQL correctness. The results show that removing the Observer leads to the largest drop in overall performance, suggesting its crucial role in ensuring the stability and effectiveness of the verification process. Moreover, the removal of the Test SQL generation module causes the most significant degradation in identifying both syntactic and semantic errors, confirming that generating targeted test cases is an essential and effective step for accurate SQL verification. Although removing the Question-Answer module results in the smallest performance drop, it still plays a role in identifying correct SQL statements and, as shown in Table 8, contributes to reducing test redundancy.

4.7. Fine-grained Case Study

Figure 3 presents one case study illustrating the design of an SQL generation plan and two case studies related to SQL correctness verification.

In Case 1, we illustrate how our SQL generation planning approach helps handle complex queries. The blue text highlights information from the Gold SQL, while the red text shows the incorrect SQL generated directly by Gemini-2.0-flash, along with its erroneous parts. The green text displays the correct SQL produced after applying our step-by-step SQL generation planning strategy. This example demonstrates that when a query involves complex calculations or requires value-level reasoning to determine filter conditions, direct generation by the LLM becomes unreliable. In contrast, our method incrementally constructs the SQL with verification and refinement at each step, significantly improving its ability to handle such complex cases.

Case 1: The process of formulating an SQL statement generation plan.
<p>Question: Name schools in Riverside which the average of average math score for SAT is grater than 400, what is the funding type of these schools?</p> <p>Gold SQL: SELECT T1.sname, T2.'Charter Funding Type' FROM satscores AS T1 INNER JOIN frpm AS T2 ON T1.cds = T2.CDSCode WHERE T2.'District Name' LIKE "Riverside%" GROUP BY T1.sname, T2.'Charter Funding Type' HAVING CAST(SUM(T1.AvgScrMath) AS REAL) / COUNT(T1.cds) > 400;</p>
Gemini-2.0-flash
<p>Predicted SQL: SELECT s.School, s.FundingType FROM schools s JOIN satscores ss ON s.CDSCode = ss.cds WHERE s.County = "Riverside" AND ss.AvgScrMath > 400;</p>
G²SQL + Gemini-2.0-flash
<p>Plan: "Step 1: Select all rows from the frpm table where the district name starts with 'Riverside'.": "SELECT * FROM frpm WHERE 'District Name' LIKE 'Riverside%'",</p> <p>"Step 2: Extract only the CDSCode and Charter Funding Type from the filtered Riverside district rows.": "SELECT CDSCode, 'Charter Funding Type' FROM frpm WHERE 'District Name' LIKE 'Riverside%'",</p> <p>...</p> <p>"Step 6: Filter the grouped results to include only those schools with an average math score greater than 400.": "SELECT satscores.sname, frpm.'Charter Funding Type' FROM satscores INNER JOIN frpm ON satscores.cds = frpm.CDSCode WHERE frpm.'District Name' LIKE 'Riverside%' GROUP BY satscores.sname, frpm.'Charter Funding Type' HAVING AVG(satscores.AvgScrMath) > 400".</p> <p>Predicted SQL: SELECT satscores.sname, frpm.'Charter Funding Type' FROM satscores INNER JOIN frpm ON satscores.cds = frpm.CDSCode WHERE frpm.'District Name' LIKE 'Riverside%' GROUP BY satscores.sname, frpm.'Charter Funding Type' HAVING AVG(satscores.AvgScrMath) > 400</p>
Case 2: Examples of corrupted databases were generated during SQL validation.
<p>Question: What is the free rate for students between the ages of 5 and 17 at the school run by Kacey Gibson?</p> <p>Predicted SQL: SELECT CAST(T2.'Free Meal Count (Ages 5-17)' AS REAL) / T2.'Enrollment (Ages 5-17)' FROM schools AS T1 INNER JOIN frpm AS T2 ON T1.CDSCode = T2.CDSCode WHERE T1.AdmFName1 = 'Kacey' AND T1.AdmLName1 = 'Gibson';</p> <p>Reviewer:</p> <p>① "Question": "Insert the corresponding values into the 'schools' table and check the connection operation of the SQL statement.";</p> <p>② "SQL": "INSERT INTO frpm (AdmFName1) VALUES ('Kacey'), ('Gibson')";</p> <p>Observer: This test case verifies the correctness of SQL by inserting data into the database, which may damage the security of database data and does not allow the use of database operation statements for verification.</p>
Case 3: An example of duplicate validation occurred during SQL checking.
<p>Question: What is the free rate for students between the ages of 5 and 17 at the school run by Kacey Gibson?</p> <p>Predicted SQL: SELECT CAST(T2.'Free Meal Count (Ages 5-17)' AS REAL) / T2.'Enrollment (Ages 5-17)' FROM schools AS T1 INNER JOIN frpm AS T2 ON T1.CDSCode = T2.CDSCode WHERE T1.AdmFName1 = 'Kacey' AND T1.AdmLName1 = 'Gibson';</p> <p>Reviewer:</p> <p>① "Question": "Check that the 'Free Meal Count (Ages 5-17)' / 'Enrollment (Ages 5-17)' calculations are performed correctly.";</p> <p>② "SQL": "SELECT CAST(T2.'Free Meal Count (Ages 5-17)' AS REAL) / T2.'Enrollment (Ages 5-17)' FROM schools AS T1 INNER JOIN frpm AS T2 ON T1.CDSCode = T2.CDSCode LIMIT 1";</p> <p>Action (DB) : [0.516822429906542]</p> <p>③ "Question": "Did the SQL statement successfully perform the relevant calculation? Is the calculation necessary?".</p> <p>Observer : This test unit has already been successfully validated and does not need to be validated again.</p>

Fig. 3. Several case studies of G²SQL in SQL generation and SQL accuracy verification.

In Case 2, we show a problematic example from the SQL verification stage, where the LLM attempts to verify correctness by executing data-modifying SQL statements. The red text highlights such a test SQL statement, which was intended to insert specific values into the database in order to validate filter conditions. However, executing such SQL statements can corrupt the database by modifying its original content, thereby posing a serious security threat. Our method introduces an Observer—an auxiliary LLM that supervises the SQL verification process. The blue text marks the Observer’s intervention, which successfully prevents the harmful operation and provides feedback for a safer verification process, thus helping preserve database integrity.

In Case 3, we provide an example of redundant or excessive verification during the SQL checking process. The purple text highlights unnecessary steps where the model redundantly validates arithmetic operations and multi-table joins that were already covered in the previous check. This not only adds overhead but also wastes computational resources. The blue text again shows the Observer’s intervention, which effectively identifies and halts the redundant checks, thereby improving verification efficiency and reducing resource usage.

4.8. Analysis and Discussion of Hallucinations and Errors

Hallucination remains a persistent challenge for large language models (LLMs). It refers to the generation of fictitious or misleading content that deviates from user intent, which can significantly hinder task performance. This issue is particularly pronounced in the Text-to-SQL domain, where hallucinations may indirectly result in incorrect SQL generation or the production of harmful and redundant test cases during verification.

It is important to distinguish between hallucinations and errors. Errors lead to immediate task failures—for example, generating SQL statements with syntactic mistakes. Hallucinations, on the other hand, may not cause overt failures: the generated SQL may be syntactically correct and return results aligned with the user’s intent, but it may involve unnecessary table joins or irrelevant operations that reflect over-generation rather than direct errors.

To quantitatively assess the capability of G²SQL in mitigating hallucinations and correcting errors, we adopt the definitions provided by TA-SQL [21] and categorize the generated SQL statements accordingly.

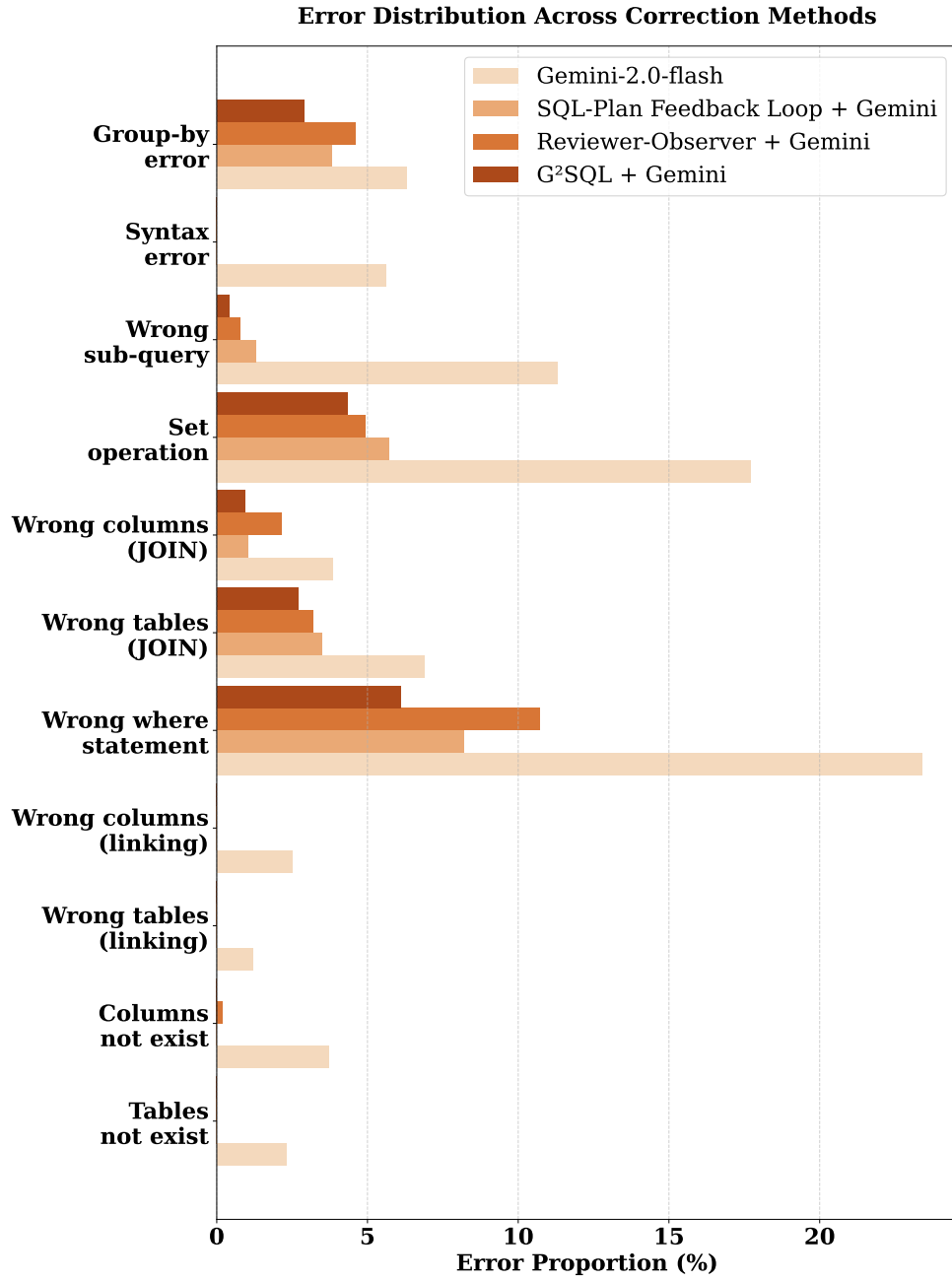


Fig. 4. Error Distribution Across Correction Methods

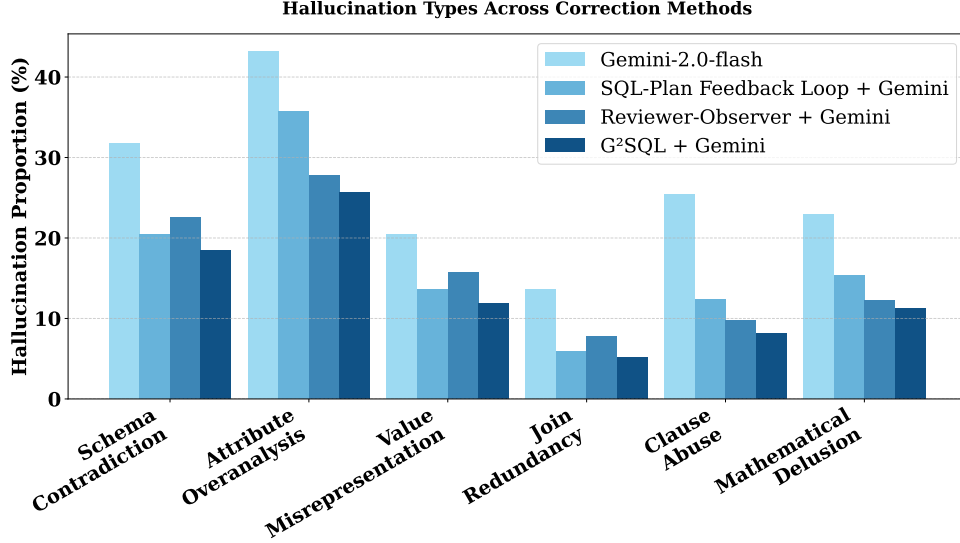


Fig. 5. Hallucination Types Across Correction Methods

As shown in Figure 4, we analyze the effectiveness of G²SQL’s two core components—SQL-Plan Feedback Loop and Reviewer-Observer—in reducing SQL errors. Both components contribute substantially to error reduction across various types, and the full G²SQL framework, which integrates both, achieves the best overall performance. However, challenges remain in handling highly complex queries, particularly those involving intricate filtering conditions.

Figure 5 presents a comparison of G²SQL and its components in mitigating hallucinations. The results indicate that G²SQL effectively reduces various forms of hallucination. Nonetheless, over-analysis, where the SQL includes redundant tables and columns, emerges as the most persistent and difficult-to-eliminate issue. This form of hallucination remains a major challenge for G²SQL and other LLM-based approaches alike.

4.9. Token Efficiency

Given that LLM APIs typically charge based on the number of tokens used, token consumption directly reflects the economic cost of SQL generation. To quantitatively evaluate the cost-effectiveness of G²SQL in terms of model invocation, we present a comparison in Figure 6, showing the average number of tokens consumed to generate a single SQL statement on the BIRD

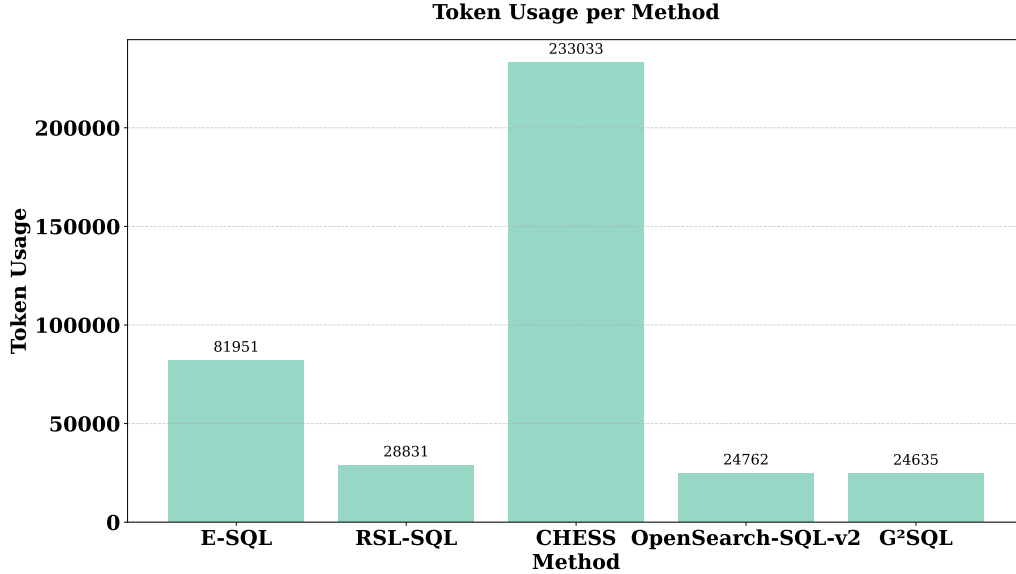


Fig. 6. Token Usage per Method

development set.

As illustrated in the figure, G²SQL achieves the lowest token usage among all evaluated methods. In particular, it reduces token consumption by nearly 10 \times compared to CHESS, and achieves higher execution accuracy than OpenSearch-SQL-v2, despite having a similar token usage. These results demonstrate that G²SQL maintains strong performance while being more economically efficient, highlighting its advantage in both accuracy and cost.

5. Conclusion and Future Work

In this paper, we propose G²SQL, a novel Text-to-SQL framework that performs validation and revision both before and after SQL generation. This two-stage approach effectively mitigates issues caused by the propagation of errors during the SQL generation process and addresses the challenges associated with SQL verification and optimization. For SQL accuracy checking, our method operates under a supervised setting, ensuring that the validation process is both safe and effective. This design avoids potential database corruption caused by unsafe SQL operations generated by the model and prevents resource waste and ineffective evaluation due to redundant or overlapping test cases.

Although our proposed method has achieved promising results, the SQL correction process has yet to reach its full potential. In future work, we plan to further explore strategies for targeted SQL revision, aiming to better leverage the information from the checking module. For example, we will investigate differentiated correction strategies for syntactic and semantic errors—adjusting syntactic errors to align more closely with the correct tables and columns, and guiding semantic errors toward the intended query logic. Additionally, we will explore whether a divide-and-conquer strategy is necessary for handling SQL queries of varying difficulty levels, which remains an open question for further investigation.

Acknowledgments

This research work is supported by the National Natural Science Foundation of China (62062046, 62462042) and CCF-Huawei Populus Grove Fund.

Data availability

The dataset used in this paper is available for download at the following link: BIRD: <https://bird-bench.github.io/>, Spider: <https://yale-lily.github.io/spider>.

References

- [1] K. Lv, Y. Yang, T. Liu, Q. Gao, Q. Guo, X. Qiu, Full parameter finetuning for large language models with limited resources, arXiv preprint arXiv:2306.09782 (2023).
- [2] S. Devalal, A. Karthikeyan, Lora technology-an overview, in: 2018 second international conference on electronics, communication and aerospace technology (ICECA), IEEE, 2018, pp. 284–290.
- [3] T. Dettmers, A. Pagnoni, A. Holtzman, L. Zettlemoyer, Qlora: Efficient finetuning of quantized llms, Advances in neural information processing systems 36 (2023) 10088–10115.
- [4] H. Li, J. Zhang, H. Liu, J. Fan, X. Zhang, J. Zhu, R. Wei, H. Pan, C. Li, H. Chen, Codes: Towards building open-source language models for text-to-sql, Proceedings of the ACM on Management of Data 2 (3) (2024) 1–28.

- [5] H. Li, S. Wu, X. Zhang, X. Huang, J. Zhang, F. Jiang, S. Wang, T. Zhang, J. Chen, R. Shi, et al., Omnisql: Synthesizing high-quality text-to-sql data at scale, arXiv preprint arXiv:2503.02240 (2025).
- [6] D. Gao, H. Wang, Y. Li, X. Sun, Y. Qian, B. Ding, J. Zhou, Text-to-sql empowered by large language models: A benchmark evaluation, *Proc. VLDB Endow.* 17 (5) (2024) 1132–1145.
- [7] M. Pourreza, D. Rafiei, Din-sql: Decomposed in-context learning of text-to-sql with self-correction, in: *Advances in Neural Information Processing Systems*, 2023, pp. 36339–36348.
- [8] Z. Li, X. Wang, J. Zhao, S. Yang, G. Du, X. Hu, B. Zhang, Y. Ye, Z. Li, R. Zhao, et al., Pet-sql: A prompt-enhanced two-stage text-to-sql framework with cross-consistency, arXiv e-prints (2024) arXiv-2403.
- [9] Z. Cao, Y. Zheng, Z. Fan, X. Zhang, W. Chen, X. Bai, Rsl-sql: Robust schema linking in text-to-sql generation, arXiv preprint arXiv:2411.00073 (2024).
- [10] A. Liu, X. Hu, L. Lin, L. Wen, Semantic enhanced text-to-sql parsing via iteratively learning schema linking graph, in: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 1021–1030.
- [11] J. Li, B. Hui, R. Cheng, B. Qin, C. Ma, N. Huo, F. Huang, W. Du, L. Si, Y. Li, Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing, in: *Proceedings of the AAAI conference on artificial intelligence*, 2023, pp. 13076–13084.
- [12] C. Hu, J. Fu, C. Du, S. Luo, J. Zhao, H. Zhao, Chatdb: Augmenting llms with databases as their symbolic memory, arXiv preprint arXiv:2306.03901 (2023).
- [13] J. Li, B. Hui, G. Qu, J. Yang, B. Li, B. Li, B. Wang, B. Qin, R. Geng, N. Huo, et al., 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 (2023) 42330–42357.
- [14] X. Liu, Z. Tan, Divide and prompt: Chain of thought prompting for text-to-sql, arXiv preprint arXiv:2304.11556 (2023).

- [15] T. Wang, H. Lin, X. Han, L. Sun, X. Chen, H. Wang, Z. Zeng, Dbcpilot: Scaling natural language querying to massive databases, arXiv preprint arXiv:2312.03463 (2023).
- [16] D. Lee, C. Park, J. Kim, H. Park, Mcs-sql: Leveraging multiple prompts and multiple-choice selection for text-to-sql generation, arXiv preprint arXiv:2405.07467 (2024).
- [17] Y. Gao, Y. Liu, X. Li, X. Shi, Y. Zhu, Y. Wang, S. Li, W. Li, Y. Hong, Z. Luo, et al., Xiyao-sql: A multi-generator ensemble framework for text-to-sql, arXiv preprint arXiv:2411.08599 (2024).
- [18] S. Talaei, M. Pourreza, Y.-C. Chang, A. Mirhoseini, A. Saberi, Chess: Contextual harnessing for efficient sql synthesis, arXiv preprint arXiv:2405.16755 (2024).
- [19] M. Pourreza, H. Li, R. Sun, Y. Chung, S. Talaei, G. T. Kakkar, Y. Gan, A. Saberi, F. Ozcan, S. O. Arik, Chase-sql: Multi-path reasoning and preference optimized candidate selection in text-to-sql, arXiv preprint arXiv:2410.01943 (2024).
- [20] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin, et al., A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions, ACM Transactions on Information Systems 43 (2) (2025) 1–55.
- [21] G. Qu, J. Li, B. Li, B. Qin, N. Huo, C. Ma, R. Cheng, Before generation, align it! A novel and effective strategy for mitigating hallucinations in text-to-sql generation, in: Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024, 2024, pp. 5456–5471.
- [22] X. Dong, C. Zhang, Y. Ge, Y. Mao, Y. Gao, J. Lin, D. Lou, et al., C3: Zero-shot text-to-sql with chatgpt, arXiv preprint arXiv:2307.07306 (2023).
- [23] Z. Gu, J. Fan, N. Tang, S. Zhang, Y. Zhang, Z. Chen, L. Cao, G. Li, S. Madden, X. Du, Interleaving pre-trained language models and large language models for zero-shot nl2sql generation, arXiv preprint arXiv:2306.08891 (2023).

- [24] S. Chang, E. Fosler-Lussier, How to prompt LLMs for text-to-SQL: A study in zero-shot, single-domain, and cross-domain settings, in: NeurIPS 2023 Second Table Representation Learning Workshop, 2023.
- [25] N. Rajkumar, R. Li, D. Bahdanau, Evaluating the text-to-sql capabilities of large language models, arXiv preprint arXiv:2204.00498 (2022).
- [26] R. Sun, S. Ö. Arik, A. Muzio, L. Miculicich, S. Gundabathula, P. Yin, H. Dai, H. Nakhost, R. Sinha, Z. Wang, et al., Sql-palm: Improved large language model adaptation for text-to-sql (extended), arXiv preprint arXiv:2306.00739 (2023).
- [27] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, D. Zhou, Chain-of-thought prompting elicits reasoning in large language models, in: Proceedings of the 36th International Conference on Neural Information Processing Systems, 2022, pp. 24824–24837.
- [28] D. Zhou, N. Schärli, L. Hou, J. Wei, N. Scales, X. Wang, D. Schuurmans, C. Cui, O. Bousquet, Q. V. Le, E. H. Chi, Least-to-most prompting enables complex reasoning in large language models, in: The Eleventh International Conference on Learning Representations, 2023.
- [29] X. Wang, J. Wei, D. Schuurmans, Q. V. Le, E. H. Chi, S. Narang, A. Chowdhery, D. Zhou, Self-consistency improves chain of thought reasoning in language models, in: The Eleventh International Conference on Learning Representations, 2023.
- [30] B. Wang, C. Ren, J. Yang, X. Liang, J. Bai, L. Chai, Z. Yan, Q. Zhang, D. Yin, X. Sun, Z. Li, MAC-SQL: A multi-agent collaborative framework for text-to-sql, in: Proceedings of the 31st International Conference on Computational Linguistics, COLING 2025, Abu Dhabi, UAE, January 19-24, 2025, 2025, pp. 540–557.
- [31] B. Zhang, Y. Ye, G. Du, X. Hu, Z. Li, S. Yang, C. H. Liu, R. Zhao, Z. Li, H. Mao, Benchmarking the text-to-sql capability of large language models: A comprehensive evaluation, arXiv preprint arXiv:2403.02951 (2024).
- [32] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, et al., Spider: A large-scale human-labeled dataset

for complex and cross-domain semantic parsing and text-to-sql task, in: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, 2018, pp. 3911–3921.

- [33] J. Li, B. Hui, G. Qu, J. Yang, B. Li, B. Li, B. Wang, B. Qin, R. Geng, N. Huo, X. Zhou, M. Chenhao, G. Li, K. Chang, F. Huang, R. Cheng, Y. Li, Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls, in: Advances in Neural Information Processing Systems, 2023, pp. 42330–42357.
- [34] R. Zhong, T. Yu, D. Klein, Semantic evaluation for text-to-sql with distilled test suites, in: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2020, pp. 396–411.
- [35] A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, et al., Deepseek-v3 technical report, arXiv preprint arXiv:2412.19437 (2024).
- [36] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Lu, et al., Qwen2.5-coder technical report, arXiv preprint arXiv:2409.12186 (2024).
- [37] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al., Gpt-4 technical report, arXiv preprint arXiv:2303.08774 (2023).
- [38] A. Hurst, A. Lerer, A. P. Goucher, A. Perelman, A. Ramesh, A. Clark, A. Ostrow, A. Welihinda, A. Hayes, A. Radford, et al., Gpt-4o system card, arXiv preprint arXiv:2410.21276 (2024).
- [39] H. A. Caferoğlu, Ö. Ulusoy, E-sql: Direct schema linking via question enrichment in text-to-sql, arXiv preprint arXiv:2409.16751 (2024).
- [40] X. Xie, G. Xu, L. Zhao, R. Guo, Opensearch-sql: Enhancing text-to-sql with dynamic few-shot and consistency alignment, arXiv preprint arXiv:2502.14913 (2025).
- [41] K. Maamari, F. Abubaker, D. Jaroslawicz, A. Mhedhbi, The death of schema linking? text-to-SQL in the age of well-reasoned language models, in: NeurIPS 2024 Third Table Representation Learning Workshop, 2024.

- [42] Z. Wang, R. Zhang, Z. Nie, J. Kim, Tool-assisted agent on sql inspection and refinement in real-world scenarios, arXiv preprint arXiv:2408.16991 (2024).

Appendix A. Details of Model Fine-tuning

During the fine-tuning data preparation stage, we constructed a dataset consisting of both correct and incorrect SQL queries by leveraging the training sets of BIRD and Spider. Specifically, we employed Gemini-2.0-flash in a zero-shot prompting setting to generate SQL queries for each natural language question in the training data. The BIRD training set contains 9,428 natural language question–SQL pairs, while Spider includes 8,659 such pairs. After applying Gemini-2.0-flash to generate SQL queries for all questions in both datasets, we obtained 9,628 correctly generated SQL queries and 8,459 incorrectly generated ones.

We labelled these data accordingly and used them to fine-tune the Qwen2.5-coder-7b model using the LoRA (Low-Rank Adaptation) method. The key hyperparameters were set as follows: `lora_r` = 8 (rank of the low-rank matrices), `lora_alpha` = 16 (scaling factor), and `lora_dropout` = 0.05 (dropout rate). We used the AdamW optimizer with a learning rate of 5e-5, trained the model for 5 epochs with a total of 1000 steps, and enabled fp16 mixed-precision training to improve efficiency. The learning rate scheduler was set to a linear decay schedule (`lr_scheduler_type`="linear").

Appendix B. Case

Input

Question: Please list the lowest three eligible free rates for students aged 5-17 in continuation schools.

Evidence: Eligible free rates for students aged 5-17 = ‘Free Meal Count (Ages 5-17)’ ‘Enrollment (Ages 5-17)’

DataBase Schema:

Database_Name california_schools

Table: frpm

[('CDSCode:TEXT', 'Primary Key', "Value_example:[('01100170109835'), ('01100170112607'), ('01100170118489',)]"), ('Academic Year:TEXT', "Value_example:[('2014-2015',)]"), ('County Code:TEXT', "Value_example:[('01'), ('02'), ('03',)]"), ('District Code:INTEGER', "Value_example:[(10017,), (31609,), (31617,)]"), ('School Code:TEXT', "Value_example:[('0109835'), ('0112607'), ('0118489',)]"), ('County Name:TEXT', "Value_example:[('Alameda'), ('Alpine'),

('Amador',))), ('District Name:TEXT', "Value_example:[('Alameda County Office of Education',), ('California School for the Blind (State Special Schl)',), ('California School for the Deaf-Fremont (State Special Schl)',)]"), ('School Name:TEXT', "Value_example:[('FAME Public Charter',), ('Envision Academy for Arts & Technology',), ('Aspire California College Preparatory Academy',)]"), ('District Type:TEXT', "Value_example:[('County Office of Education (COE)',), ('State Special Schools',), ('Unified School District',)]"), ('School Type:TEXT', "Value_example:[('K-12 Schools (Public)',), ('High Schools (Public)',), ('Elementary Schools (Public)',)]"), ('Educational Option Type:TEXT', "Value_example:[('Traditional',), ('Juvenile Court School',), ('County Community School',)]"), ('NSLP Provision Status:TEXT', "Value_example:[('Breakfast Provision 2',), ('Provision 2',), ('CEP',)]"), ('Charter School (Y/N):INTEGER', 'Value_example:[(1,), (0,)]'), ('Charter School Number:TEXT', "Value_example:[('0728',), ('0811',), ('1049',)]"), ('Charter Funding Type:TEXT', "Value_example:[('Directly funded',), ('Locally funded',), ('Not in CS funding model',)]"), ('IRC:INTEGER', 'Value_example:[(1,), (0,)]'), ('Low Grade:TEXT', "Value_example:[('K',), ('9',), ('1',)]"), ('High Grade:TEXT', "Value_example:[('12',), ('8',), ('5',)]"), ('Enrollment (K-12):REAL', 'Value_example:[(1087.0,), (395.0,), (244.0,)]'), ('Free Meal Count (K-12):REAL', 'Value_example:[(565.0,), (186.0,), (134.0,)]'), ('Percent (%) Eligible Free (K-12):REAL', 'Value_example:[(0.519779208831647,), (0.470886075949367,), (0.549180327868853,)]'), ('FRPM Count (K-12):REAL', 'Value_example:[(715.0,), (186.0,), (175.0,)]'), ('Percent (%) Eligible FRPM (K-12):REAL', 'Value_example:[(0.657773689052438,), (0.470886075949367,), (0.717213114754098,)]'), ('Enrollment (Ages 5-17):REAL', 'Value_example:[(1070.0,), (376.0,), (230.0,)]'), ('Free Meal Count (Ages 5-17):REAL', 'Value_example:[(553.0,), (182.0,), (128.0,)]'), ('Percent (%) Eligible Free (Ages 5-17):REAL', 'Value_example:[(0.516822429906542,), (0.484042553191489,), (0.556521739130435,)]'), ('FRPM Count (Ages 5-17):REAL', 'Value_example:[(702.0,), (182.0,), (168.0,)]'), ('Percent (%) Eligible FRPM (Ages 5-17):REAL', 'Value_example:[(0.65607476635514,),

```
(0.484042553191489,),(0.730434782608696,)]'), ('2013-14 CALPADS
Fall 1 Certification Status:INTEGER', 'Value_example:[(1,)]']]
```

frpm Column Description

```
[County Name(description: County Code ), FRPM Count
(K-12)(description: Free or Reduced Price Meal Count
(K-12))] # Table: satscores [('cds:TEXT', 'Primary Key',
"Value_example:[('10101080000000',), ('10101080109991',),
('10101080111682',)]"), ('rtype:TEXT', "Value_example:[('D',),
('S',)]"), ('sname:TEXT', "Value_example:[('FAME Public Char-
ter',), ('Envision Academy for Arts & Technology',), ('Aspire
California College Preparatory Academy',)]"), ('dname:TEXT',
"Value_example:[('Alameda County Office of Education',),
('Alameda Unified',), ('Albany City Unified',)]"), ('cname:TEXT',
"Value_example:[('Alameda',), ('Amador',), ('Butte',)]"), ('en-
roll12:INTEGER', 'Value_example:[(398,),(62,),(75,)]'), ('NumTst-
Tskr:INTEGER', 'Value_example:[(88,),(17,),(71,)]'), ('AvgScr-
Read:INTEGER', 'Value_example:[(418,),(503,),(397,)]'),
('AvgScrMath:INTEGER', 'Value_example:[(418,),(546,),(387,)]'),
('AvgScrWrite:INTEGER', 'Value_example:[(417,),(505,),(395,)]'),
('NumGE1500:INTEGER', 'Value_example:[(14,),(9,),(5,)]']]
```

satscores Column Description

```
[cds(description: California Department Schools), sname(description:
school name), dname(description: district segment),
cname(description: county name), enroll12(description: enroll-
ment (1st-12nd grade)), NumTstTskr(description: Number of
Test Takers in this school), AvgScrRead(description: average
scores in Reading), AvgScrMath(description: average scores
in Math), AvgScrWrite(description: average scores in writing),
NumGE1500(description: Number of Test Takers Whose Total SAT
Scores Are Greater or Equal to 1500)]
```

Table: schools

```
[('CDSCode:TEXT', 'Primary Key', "Value_example:[('01100170000000',),
('01100170109835',), ('01100170112607',)]"), ('NCESDist:TEXT',
"Value_example:[('0691051',), ('0600002',), ('0600003',)]"), ('NCES-
school:TEXT', "Value_example:[('10546',), ('10947',), ('12283',)]"),
('StatusType:TEXT', "Value_example:[('Active',), ('Closed',),
```

```

('Merged',)]"), ('County:TEXT', "Value_example:[('Alameda'),
('Alpine'), ('Amador'),]"), ('District:TEXT',
"Value_example:[('Alameda County Office of Education'), ('Cal-
ifornia School for the Blind (State Special Schl)'), ('California School
for the Deaf-Fremont (State Special Schl)'),]"), ('School:TEXT',
"Value_example:[('FAME Public Charter'), ('Envision Academy
for Arts & Technology'), ('Aspire California College Preparatory
Academy'),]"), ('Street:TEXT', "Value_example:[('313 West Winton
Avenue'), ('39899 Balentine Drive, Suite 335'), ('1515 Webster
Street'),]"), ('StreetAbr:TEXT', "Value_example:[('313 West Winton
Ave.'), ('39899 Balentine Dr., Ste. 335'), ('1515 Webster St.')]"),
('City:TEXT', "Value_example:[('Hayward'), ('Newark'), ('Oak-
land'),]"), ('Zip:TEXT', "Value_example:[('94544-1136'), ('94560-
5359'), ('94612-3355'),]"), ('State:TEXT', "Value_example:[('CA'),]"),
('MailStreet:TEXT', "Value_example:[('313 West Winton Avenue'),
('39899 Balentine Drive, Suite 335'), ('1515 Webster Street'),]"),
('MailStrAbr:TEXT', "Value_example:[('313 West Winton Ave.'),
('39899 Balentine Dr., Ste. 335'), ('1515 Webster St.')]"),
('MailCity:TEXT', "Value_example:[('Hayward'), ('Newark'),
('Oakland'),]"), ('MailZip:TEXT', "Value_example:[('94544-
1136'), ('94560-5359'), ('94612'),]"), ('MailState:TEXT',
"Value_example:[('CA'),]"), ('Phone:TEXT', "Value_example:[('510)
887-0152'), ('510) 596-8901'), ('510) 686-4131'),]"), ('Ext:TEXT',
"Value_example:[('130'), ('1240'), ('1200'),]"), ('Website:TEXT',
"Value_example:[('www.acoe.org'), ('www.envisionacademy.org/'),
('www.aspirepublicschools.org'),]"), ('OpenDate:DATE',
"Value_example:[('2005-08-29'), ('2006-08-28'), ('2008-08-21'),]"),
('ClosedDate:DATE', "Value_example:[('2015-07-31'), ('2015-06-
30'), ('1989-06-30'),]"), ('Charter:INTEGER', 'Value_example:[(1,
0,)]'), ('CharterNum:TEXT', "Value_example:[('0728'), ('0811'),
('1049'),]"), ('FundingType:TEXT', "Value_example:[('Directly
funded'), ('Locally funded'), ('Not in CS funding model'),]"),
('DOC:TEXT', "Value_example:[('00'), ('31'), ('34'),]"), ('DOC-
Type:TEXT', "Value_example:[('County Office of Education
(COE)'), ('State Special Schools'), ('Non-School Locations'),]"),
('SOC:TEXT', "Value_example:[('65'), ('66'), ('60'),]"), ('SOC-

```

```

Type:TEXT', "Value_example:[('K-12 Schools (Public)'), ('High
Schools (Public)'), ('Elementary Schools (Public)',)]"), ('EdOps
sCode:TEXT', "Value_example:[('TRAD'), ('JUV'), ('COMM',)]"),
('EdOpsName:TEXT', "Value_example:[('Traditional'), ('Ju-
venile Court School'), ('County Community School',)]"),
('EILCode:TEXT', "Value_example:[('ELEMHIGH'), ('HS'),
('ELEM',)]"), ('EILName:TEXT', "Value_example:[('Elementary-
High Combination'), ('High School'), ('Elementary',)]"), ('GSof-
fered:TEXT', "Value_example:[('K-12'), ('9-12'), ('K-8',)]"),
('GSServed:TEXT', "Value_example:[('K-12'), ('9-12'), ('K-7',)]"),
('Virtual:TEXT', "Value_example:[('P'), ('N'), ('F',)]"), ('Mag-
net:INTEGER', 'Value_example:[(0), (1,)]'), ('Latitude:REAL',
'Value_example:[(37.658212,), (37.521436,), (37.80452,)]'), ('Lon-
gitude:REAL', 'Value_example:[(-122.09713,), (-121.99391,),
(-122.26815,)]'), ('AdmFName1:TEXT', "Value_example:[('L
Karen'), ('Laura'), ('Clifford',)]"), ('AdmLName1:TEXT',
"Value_example:[('Monroe'), ('Robell'), ('Thompson',)]"),
('AdmEmail1:TEXT', "Value_example:[('lkmonroe@acoe.org'),
('laura@envisionacademy.org'), ('AdmFName2:TEXT',
"Value_example:[('Sau-Lim (Lance)'), ('Jennifer'), ('Annal-
isa'), ('cliffordt@communityschoolforcreativeeducation.org',)]")"),
('AdmLName2:TEXT', "Value_example:[('Tsang'),
('Koelling'), ('Moore',)]"), ('AdmEmail2:TEXT',
"Value_example:[('stsang@unityhigh.org'), ('jkoelling@efcps.net'),
('annalisa.moore@neacalc.org',)]"), ('AdmFName3:TEXT',
"Value_example:[('Drew'), ('Irma'), ('Vickie',)]"),
('AdmLName3:TEXT', "Value_example:[('Sarratore'),
('Munoz'), ('Chang',)]"), ('AdmEmail3:TEXT',
"Value_example:[('dsarratore@vincentacademy.org'),
('gmunoz@piedmont.k12.ca.us'), ('vickiechang@acoe.org',)]"), ('Las-
tUpdate:DATE', "Value_example:[('2015-06-23'), ('2015-09-01'),
('2015-06-18',)]")]]

```

schools Column Description

[NCESDist(description: This field represents the 7-digit National Center for Educational Statistics (NCES) school district identification number. The first 2 digits identify the state and the last 5 digits

identify the school district. Combined, they make a unique 7-digit ID for each school district.), NCESSchool(description: This field represents the 5-digit NCES school identification number. The NCESSchool combined with the NCESDist form a unique 12-digit ID for each school.), StatusType(description: This field identifies the status of the district.), County(description: County name), StreetAbr(description: The abbreviated street address of the school, district, or administrative authority's physical location.), Ext(description: The phone number extension of the school, district, or administrative authority.), Website(description: The website address of the school, district, or administrative authority.), OpenDate(description: The date the school opened.), ClosedDate(description: The date the school closed.), Charter(description: This field identifies a charter school.), CharterNum(description: The charter school number.), FundingType(description: Indicates the charter school funding type), DOC(description: District Ownership Code), DOCType(description: The District Ownership Code Type is the text description of the DOC category.), SOC(description: The School Ownership Code is a numeric code used to identify the type of school.), SOCType(description: The School Ownership Code Type is the text description of the type of school.), EdOpsCode(description: The Education Option Code is a short text description of the type of education offered.), EdOpsName(description: Educational Option Name), EILCode(description: The Educational Instruction Level Code is a short text description of the institution's type relative to the grade range served.), EILName(description: The Educational Instruction Level Name is the long text description of the institution's type relative to the grade range served.), GSoffered(description: The grade span offered is the lowest grade and the highest grade offered or supported by the school, district, or administrative authority. This field might differ from the grade span served as reported in the most recent certified California Longitudinal Pupil Achievement (CALPADS) Fall 1 data collection.), GSServed(description: It is the lowest grade and the highest grade of student enrollment as reported in the most recent certified CALPADS Fall 1 data collection. Only K–12 enrollment is reported through CALPADS. This field may differ from the grade span offered.), Vir-

tual(description: This field identifies the type of virtual instruction offered by the school. Virtual instruction is instruction in which students and teachers are separated by time and/or location, and interaction occurs via computers and/or telecommunications technologies.), Magnet(description: This field identifies whether a school is a magnet school and/or provides a magnet program.), Latitude(description: The angular distance (expressed in degrees) between the location of the school, district, or administrative authority and the equator measured north to south.), Longitude(description: The angular distance (expressed in degrees) between the location of the school, district, or administrative authority and the prime meridian (Greenwich, England) measured from west to east.), AdmFName1(description: administrator's first name), AdmLName1(description: administrator's last name), AdmEmail1(description: administrator's email address)]

[foreign keys]
 ['schools.CDSCode = frpm.CDSCode', 'schools.CDSCode = satscores.cds']

SQL-Plan Feedback Loop

PreSQL

```
SELECT s.School AS school_name, f.[County Name] AS county,
f.[Percent (%) Eligible Free (Ages 5-17)] AS eligible_free_rate FROM
schools s JOIN frpm f ON s.CDSCode = f.CDSCode WHERE
s.EdOpsName = 'Continuation' AND f.[Percent (%) Eligible Free
(Ages 5-17)] IS NOT NULL ORDER BY f.[Percent (%) Eligible Free
(Ages 5-17)] ASC LIMIT 3;
```

SQL-to-Plan

Step 1: "Join the 'schools' and 'frpm' tables on the CDSCode field to combine school metadata with meal program information.": "SELECT * FROM schools s JOIN frpm f ON s.CDSCode = f.CDSCode",

Step 2: "Filter the joined result to only include continuation schools by selecting rows where the 'EdOpsName' is 'Continuation'.": "SELECT * FROM schools s JOIN frpm f ON s.CDSCode = f.CDSCode WHERE s.EdOpsName = 'Continuation'",

Step 3: "Further filter the result to only include rows where the percentage of students aged 5–17 eligible for free meals is not NULL.": "SELECT * FROM schools s JOIN frpm f ON s.CDSCode = f.CDSCode WHERE s.EdOpsName = 'Continuation' AND f.[Percent (%) Eligible Free (Ages 5-17)] IS NOT NULL",

Step 4: "Select only the school name, county name, and eligible free meal percentage for students aged 5–17.": "SELECT s.School AS school_name, f.[County Name] AS county, f.[Percent (%) Eligible Free (Ages 5-17)] AS eligible_free_rate FROM schools s JOIN frpm f ON s.CDSCode = f.CDSCode WHERE s.EdOpsName = 'Continuation' AND f.[Percent (%) Eligible Free (Ages 5-17)] IS NOT NULL",

Step 5: "Sort the results in ascending order by the eligible free rate for students aged 5–17 and limit the output to the lowest three entries.": "SELECT s.School AS school_name, f.[County Name] AS county, f.[Percent (%) Eligible Free (Ages 5-17)] AS eligible_free_rate FROM schools s JOIN frpm f ON s.CDSCode = f.CDSCode WHERE s.EdOpsName = 'Continuation' AND f.[Percent (%) Eligible Free (Ages 5-17)] IS NOT NULL ORDER BY f.[Percent (%) Eligible Free (Ages 5-17)] ASC LIMIT 3"

DB_EXE

SQL 1: [CDSCode (01100170109835), NCESDist (0691051), NCESSchool (10546) ...]

SQL 2: []

SQL 3: []

SQL 4: []

SQL 5: []

Plan Check

step 1: Join the 'schools' and 'frpm' tables on the CDSCode field to combine school metadata with meal program information.

Evaluation: × Unnecessary join. The correct SQL only queries the frpm table.

step 2: Filter the joined result to only include continuation schools by selecting rows where the 'EdOpsName' is 'Continuation'.

Evaluation: × Incorrect filtering field. The relevant column is Educa-

tional Option Type in the frpm table, not EdOpsName.

step 3: Further filter the result to only include rows where the percentage of students aged 5–17 eligible for free meals is not NULL.

Evaluation: Filtering out NULL values is correct, but the previous step already invalidates this one due to incorrect filtering logic.

step 4: Select only the school name, county name, and eligible free meal percentage for students aged 5–17.

Evaluation: × The original SQL only calculates the ratio; it doesn't involve school name or county.

step 5: Sort the results in ascending order by the eligible free rate for students aged 5–17 and limit the output to the lowest three entries.

Evaluation: Correct sorting and limiting, but applied on incorrectly selected and filtered data.

Plan Fix

Step 1: "Select the free meal count and enrollment count for students aged 5–17 from the 'frpm' table to calculate eligibility rates.": "SELECT Free Meal Count (Ages 5-17), Enrollment (Ages 5-17) FROM frpm",

Step 2: "Filter the rows to only include continuation schools by checking that the 'Educational Option Type' is 'Continuation School'.": "SELECT Free Meal Count (Ages 5-17), Enrollment (Ages 5-17) FROM frpm WHERE Educational Option Type = 'Continuation School'",

Step 3: "Compute the eligible free rate by dividing free meal count by enrollment, and exclude rows where this value is NULL.": "SELECT Free Meal Count (Ages 5-17) / Enrollment (Ages 5-17) AS eligible_free_rate FROM frpm WHERE Educational Option Type = 'Continuation School' AND Free Meal Count (Ages 5-17) / Enrollment (Ages 5-17) IS NOT NULL",

Step 4: "Sort the result by eligible free rate in ascending order and return only the lowest three rates.": "SELECT Free Meal Count (Ages 5-17) / Enrollment (Ages 5-17) AS eligible_free_rate FROM frpm WHERE Educational Option Type = 'Continuation School' AND Free Meal Count (Ages 5-17) / Enrollment (Ages 5-17) IS NOT NULL ORDER BY Free Meal Count (Ages 5-17) / Enrollment (Ages

5-17) ASC LIMIT 3”

SQL Generation

SELECT Free Meal Count (Ages 5-17) / Enrollment (Ages 5-17) AS eligible_free_rate FROM frpm WHERE Educational Option Type = 'Continuation School' AND Free Meal Count (Ages 5-17) / Enrollment (Ages 5-17) IS NOT NULL ORDER BY Free Meal Count (Ages 5-17) / Enrollment (Ages 5-17) ASC LIMIT 3

SQL-to-Plan

Step 1: "Access the 'frpm' table to retrieve relevant school meal and enrollment data.": "SELECT * FROM frpm",

Step 2: "Extract only the columns related to free meal count and enrollment for students aged 5–17.": "SELECT 'Free Meal Count (Ages 5-17)', 'Enrollment (Ages 5-17)', 'Educational Option Type' FROM frpm",

Step 3: "Filter the rows to include only those where the school type is 'Continuation School'.": "SELECT 'Free Meal Count (Ages 5-17)', 'Enrollment (Ages 5-17)' FROM frpm WHERE 'Educational Option Type' = 'Continuation School'",

Step 4: "Calculate the free meal eligibility rate by dividing free meal count by enrollment, and remove rows where this rate is NULL.": "SELECT 'Free Meal Count (Ages 5-17)' / 'Enrollment (Ages 5-17)' AS eligible_free_rate FROM frpm WHERE 'Educational Option Type' = 'Continuation School' AND 'Free Meal Count (Ages 5-17)' / 'Enrollment (Ages 5-17)' IS NOT NULL",

Step 5: "Sort the results in ascending order by the eligibility rate and limit the output to the bottom three schools.": "SELECT 'Free Meal Count (Ages 5-17)' / 'Enrollment (Ages 5-17)' AS eligible_free_rate FROM frpm WHERE 'Educational Option Type' = 'Continuation School' AND 'Free Meal Count (Ages 5-17)' / 'Enrollment (Ages 5-17)' IS NOT NULL ORDER BY eligible_free_rate ASC LIMIT 3"

Plan Check

OK

Output

```
SELECT 'Free Meal Count (Ages 5-17)' / 'Enrollment (Ages 5-17)' AS  
eligible_free_rate FROM frpm WHERE 'Educational Option Type' =  
'Continuation School' AND 'Free Meal Count (Ages 5-17)' / 'Enroll-  
ment (Ages 5-17)' IS NOT NULL ORDER BY eligible_free_rate ASC  
LIMIT 3
```

Reviewer-Observer

Reviewer(Question): Are Eligible free rates for students aged 5-17 calculated correctly?

Answer: According to the provided SQL statement and evidence, the SQL statement correctly uses the calculation formula in the evidence.

Reviewer(SQL): SELECT 'Free Meal Count (Ages 5-17)' / 'Enrollment (Ages 5-17)' FROM frpm LIMIT 1;

DB: [0.516822429906542]

Reviewer(SQL): INSERT INTO frpm (Educational Option Type) VALUES ('Continuation School')

Observer: This SQL statement inserts values into the database. Executing this statement will destroy the original data. You cannot write insert statements, only query statements.

Reviewer(SQL): SELECT 'Free Meal Count (Ages 5-17)' / 'Enrollment (Ages 5-17)' FROM frpm WHERE 'Educational Option Type' = 'Continuation School' LIMIT 1;

DB: [0.482142857142857]

Reviewer(Question): The case of null values should be filtered out during calculation.

Answer: "IS NOT NULL" is used in the SQL statement to filter out null values.

Reviewer(SQL): SELECT 'Free Meal Count (Ages 5-17)' / 'Enrollment (Ages 5-17)' AS eligible_free_rate FROM frpm WHERE 'Educational Option Type' = 'Continuation School' AND 'Free Meal Count (Ages 5-17)' / 'Enrollment (Ages 5-17)' IS NOT NULL ORDER BY eligible_free_rate ASC LIMIT 3;

DB: [0.0434782608695652, 0.0704225352112676, 0.113636363636364]

Reviewer: The test was completed and no problems were found in the SQL statement.

Observer: No errors were found during the entire test process, and the SQL statement was correct.

Output

```
SELECT 'Free Meal Count (Ages 5-17)' / 'Enrollment (Ages 5-17)' AS  
eligible_free_rate FROM frpm WHERE 'Educational Option Type' =  
'Continuation School' AND 'Free Meal Count (Ages 5-17)' / 'Enroll-  
ment (Ages 5-17)' IS NOT NULL ORDER BY eligible_free_rate ASC  
LIMIT 3;
```

Output

```
SELECT 'Free Meal Count (Ages 5-17)' / 'Enrollment (Ages 5-17)' AS  
eligible_free_rate FROM frpm WHERE 'Educational Option Type' =  
'Continuation School' AND 'Free Meal Count (Ages 5-17)' / 'Enroll-  
ment (Ages 5-17)' IS NOT NULL ORDER BY eligible_free_rate ASC  
LIMIT 3;
```

Appendix C. Prompt Template

SQL generated prompt template.

You are an expert in SQLite databases.

Generate a SQL statement that matches the question based on the information provided.

The generated SQL statements must be executable on the SQLite database.

{schema}

Question:

{question}

Evidence:

{evidence}

No explanation or expansion is required, just give the SQL statement directly.

You must ensure that the output result can be parsed by python's json.loads.

Output in the following format:

```
{{"SQL": "SELECT ..."}}
```

Generate SQL prompt templates according to the plan.

You are an expert in SQLite databases.

Refer to the provided SQL generation plan and provide the SQL statement that best fits the question.

The generated SQL statements must be executable on the SQLite database.

{schema}

Question:

{question}

Evidence:

{evidence}

SQL generation plan:

{plan}

No explanation or expansion is required, just give the SQL statement directly.

You must ensure that the output result can be parsed by python's json.loads.

Output in the following format:

```
{{ "SQL": "SELECT ..." }}
```

Convert SQL statements into prompt templates for generating plans.

Based on the information given, analyze the generation process of the given SQL statement, and finally give the generation steps of the SQL.

Database Schema:

{schema}

Question:

{question}

Evidence:

{evidence}

SQL:

{sql}

Requirement:

1. Each step in the SQL generation plan can only achieve one goal.
2. Provide a SQL statement to complete each step.
3. No need to explain and expand, think step by step.

You must ensure that the output result can be parsed by python's json.loads.

The output must be strictly in the following format, and each step must be preceded by Step:

```
{ "step": { "Step 1 Text description.": "The SQL statement corresponding to step 1.", "Step 2 Text description.": "The SQL statement corresponding to step 2.", "Step 3 Text description.": "The SQL statement corresponding to step 3.", ... "Step n Text description.": "The SQL statement corresponding to step n." } }
```

A prompt template for checking and modifying SQL generation plans.

Based on the information provided, determine whether a step in a SQL generation plan meets the requirements of the question.

Database Schema:

{schema}

Question:

{question}

###The following are the relevant knowledge that must be involved when generating SQL:

{evidence}

The following are the steps that have passed the inspection:

{plans}

The following steps are steps that have not yet been judged:

{other_plans}

Here are the steps you need to make a judgment:

{step}

The execution result of the SQL statement corresponding to this step:

{db_exe}

Requirement:

1. Based on the relevant information, analyze the correctness of the

step step by step.

2. Combining all the steps, you only need to determine whether the step has completed the task that it is supposed to complete, without considering the overall situation.

3. If you think the step is correct, output 1 and give the reason;

4. If you think the step is wrong, output 0 and give the reason for the error;

5. You only need to focus on whether the expected results of this step have been achieved.

Provide your output below:

{{ "result": "0" or "1", "reason": "" }}

Generate prompt templates for SQL test cases.

You are an expert in SQLite databases.

You need to perform a validity check on a given SQL statement.

Database Schema:

{schema}

Question:

{question}

Evidence:

{evidence}

SQL statements to be checked:

{sql}

You can use the following two methods to check the accuracy of the SQL statement:

###Solution 1: Write a test SQL statement to verify the correctness of the SQL to be checked.###

###Solution 2: Ask questions about the doubts in the SQL statement and analyze the correctness of the SQL by answering the questions.###

Best Practices:

1. For parts involving calculation, filtering and nested queries, SQL statements should be written to verify their correctness;

2. For the final SQL statement execution result, questions should be raised to verify whether it meets the requirements of natural language

questions.

3. The verification process should be consistent and progressive, and all parts should be verified as carefully as possible.
4. There should be no containment relationship between steps in a validation scenario.
5. Only SQL query statements can be used for verification, and no operations that modify database contents are allowed.

Requirements:

1. If you use solution 1 for verification, the output is in the following format:

```
{{ "Solution": 1, "Question": "The SQL statement to be verified."
"SQL" : "SQL statement that needs to be executed through the
database to obtain the execution result." }}
```

2. If you use solution 2 for verification, the output is in the following format:

```
{{ "Solution": 2, "Question": "Questions that need to be answered for
doubts about the SQL statement." }}
```

3. The output contains at least 10 steps that need to be verified.
4. No explanation or expansion is required.
5. You must ensure that the output result can be parsed by python's json.loads.

Output in the following format:

```
{{ "solution":[ {{ "Solution": 1 or 2, "SQL" or "Question": "SQL
statement" or "question" }}, {{ "Solution": 1 or 2, "SQL" or "Ques-
tion": "SQL statement" or "question" }} ... ], "Thinking pro-
cess:": "Give the entire thought process of SQL validation.", "Reflection
process:": "Reflect constructively on the process." }}
```

A prompt template for answering natural language questions about SQL.

You are an expert in SQLite databases.

You need to answer all the questions in the question list by combining relevant information.

Database Schema:

{schema}

Question:
 {question}
 Evidence:
 {evidence}
 SQL statements to be checked:
 {sql}
 The SQL execution result to be checked:
 {exe_result}
 List of questions:
 {question_list}
 No explanation or expansion is required.
 You must ensure that the output result can be parsed by python's
 json.loads.
 Output in the following format:
 {{ "answer":[{{ "Question":"Answered questions.", "An-
 swer":"Detailed answers to questions." }}, {{ "Question":"Answered
 questions.", "Answer":"Detailed answers to questions." }}, ...
], "Thinking process":"The thought process that goes into an-
 swering each question in your list of questions.", "Reflection
 process":"Constructive reflection on the process of answering the
 questions." }}

The prompt template for the supervision module.

You are an expert in SQLite databases. Please analyze the given SQL
 check case to see if it is valid and not make any modifications to the
 database.
 DataBase Schema:
 {schema}
 Question:
 {question}
 Evidence:
 {evidence}
 Target SQL:
 {sql}
 Tests already conducted:

{test_history}

Tests already conducted::

{plan}

Test cases that need to be checked:

{test_case}

Best Practices:

1. The SQL statements used for testing cannot modify the content and structure of any database after execution on the database;
2. The test cases used cannot be duplicated with previously tested cases;
3. Carefully analyze the purpose of the test case to ensure that it can achieve the purpose of checking SQL correctness;
4. If problems are found, suggestions need to be given.

Requirement:

If a problem is found, output 0 and provide the reason and suggestion.

If there is no problem, output 1 and the reason

You must ensure that the output result can be parsed by python's json.loads.

Output in the following format:

{{ "result": "0" or "1", "reason": " ", "suggest": " " }}