CM-SQL: A Cross-Model Consistency Framework for Text-to-SQL

Anonymous ACL submission

Abstract

In recent years, large language models (LLMs) have been widely applied to the task of Text-to-SQL. Currently, most LLM-based Text-to-SQL methods primarily adopt the following approaches to improve the accuracy of generated SQL: (1) schema linking; and (2) leveraging the model's self-consistency to check, modify, and select the generated SQL. However, due to issues such as hallucinations in LLMs, the database schema generated during the schema linking phase may contain errors or omissions. On the other hand, LLMs often exhibit overconfidence when evaluating the correctness of their outputs. To address these issues, we propose a cross-model consistency SQL generation framework (CM-SQL) , which generates SQL outputs from different perspectives by feeding two database schemas into two LLMs. The framework combines the stability of fine-tuned models with the powerful reasoning capabilities of LLMs to evaluate the generated SQL. Additionally, we propose a local modification strategy to correct erroneous SQL. Finally, the outputs of the evaluation module and the LLM are used to select candidate SQLs, yielding the final SQL. We evaluated the proposed framework using GPT-4o-mini and DeepSeek-V2.5 on the BIRD dev dataset, achieving an execution accuracy of 65.65%, significantly outperforming most methods based on the same LLMs, and achieving performance comparable to many approaches relying on more expensive models such as GPT-4.

1 Introduction

The goal of the Text-to-SQL task is to convert user-provided natural language queries into SQL statements that align with the user's intent. With the development of large language models (LLMs) , such as the GPT series, these models have demonstrated remarkable potential in areas like text translation and code generation (Gao et al., 2022, 2023a,b; Jiao et al., 2023) . In the Text-to-SQL task, LLMs are widely used in the following stages.

During the candidate SQL generation phase, relevant database schemas and external knowledge are input into the LLM to generate SQL queries that meet user requirements. Some studies (Lee et al., 2025; Pourreza et al., 2024) adjust the LLM's temperature parameter or change the order of tables and columns in the database schema to obtain different SQL queries. Other research (Gao et al., 2024b; Sarker et al., 2024) uses multiple LLM models to generate SQL, thereby enriching the candidate set. Additionally, some studies (Li et al., 2024a) apply supervised fine-tuning techniques, improving SQL generation performance by fine-tuning smaller models.

In the SQL accuracy verification process, most methods (Cao et al., 2024; Cen et al., 2024; Talaei et al., 2024) rely on various prompt engineering techniques to leverage the LLM's self-consistency checks on the generated SQL. For erroneous SQL, existing methods (Pourreza et al., 2024; Zhang et al., 2024; Talaei et al., 2024) typically input both the incorrect SQL and the database's error messages into the LLM for correction.

In the final SQL selection phase, most methods (Wang et al., 2025) adopt a self-consistency approach to select SQL queries from the candidate set, assuming that the SQL with the same execution result is the final SQL. However, this approach fails when none of the candidate SQL queries have the same execution result. Moreover, SQL queries with the same execution result are not necessarily correct. Some studies (Gao et al., 2024b; Pourreza et al., 2024) suggest using fine-tuned models for SQL selection, but due to factors such as the model size and the quality of fine-tuning data, a fine-tuned model may struggle to accurately identify the correct SQL.

The SQL generation process described above faces several challenges: (1) Model Hallucination and Parameter Sensitivity: Due to issues like hallucination in LLMs, altering model parameters or changing the order of database schemas to enrich the SQL candidate set may lead to the generation of incorrect SQL statements or the repetition of identical SQL queries. (2) Overconfidence in Self-Consistency Checking: LLMs tend to exhibit "overconfidence" when evaluating their answers (Lin et al., 2024) . Using self-consistency checks to verify SQL statements generated by the LLM may fail to identify semantically incorrect SQL queries that are syntactically valid. (3) Difficulty in Error Correction: During the error correction process for SQL statements, when the SQL is syntactically correct, the lack of explicit error location information makes it challenging for the LLM to accurately modify the incorrect parts of the query, and it may unintentionally alter previously correct sections of the query.

To address the challenges mentioned above, we propose a cross-model consistency SQL generation framework (CM-SQL) . The CM-SQL architecture first inputs the complete database schema in V-Schema format (Full-Schema) and the schema-linked database schema (Simplified-Schema) into two different LLMs to generate SQL queries from different perspectives, enriching the SQL candidate set. To maximize the accuracy of the generated SQL, we first decompose the SQL query into different parts, then evaluate the accuracy of each part and assign scores. The overall accuracy judgment and score are obtained by integrating the accuracy scores of each part. To mitigate the "overconfidence" issue in LLMs during SQL checking, we use two LLMs and a fine-tuned 7B model to assess SQL accuracy through cross-consistency. For erroneous SQL queries, we propose a local modification method that uses the output from the checking module to modify only the erroneous parts. Finally, based on the output from the checking module, we rank the SQL candidate set according to accuracy scores and input them into the LLM to select the final SQL.

Our main contributions and results are summarized as follows:

- CM-SQL proposes the V-Schema database schema organization and expands the range of candidate SQL queries by inputting both the full database schema and the schema-linked database schema into different LLMs, generating SQL candidates from various perspectives.

- CM-SQL proposes a method that first splits the SQL query and then uses the reasoning ability of LLMs and the stability of fine-tuned models to assess the SQL query and its parts. This approach not only improves the accuracy of SQL accuracy judgment but also locates the erroneous parts of the SQL.

- CM-SQL proposes a SQL local modification strategy, which only modifies the erroneous parts of the SQL query. This approach effectively reduces the likelihood of modifying correct parts incorrectly, ensuring that each modification brings the query closer to the correct answer.

- CM-SQL uses two low-cost models, DeepSeek-V2.5 and GPT-4o-mini, achieving an execution accuracy of 65.65%. This surpasses most methods based on these two models and achieves results comparable to those obtained by many methods using more expensive closed-source models.

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 (Qu et al., 2024) and PET-SQL (Li et al., 2024b) 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 (Dong et al., 2023) and CHESS (Talaei et al., 2024) 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 (Gu et al., 2023) and Graphix-T5 (Li et al., 2023a) 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 Organization of the Schema Structure

In studies (Gao et al., 2024a; Nan et al., 2023; Rajkumar et al., 2022; Arora et al., 2023) , the database schema is organized using the CREATE TABLE SQL statement, with (Arora et al., 2023; Chang and Fosler-Lussier, 2023) further enhancing this structure by adding randomly selected values or ranges of data following the table creation statements. In contrast, (Chen et al., 2024) simplifies the database schema into a structure of the form "table(column(value))" with added primary and foreign key information, thus streamlining the schema representation. Other works (Wang et al., 2025; Gao et al., 2024b) have proposed representing the database structure in a semi-structured format. For instance, (Gao et al., 2024b) introduces M-Schema, which employs a two-stage retrieval strategy based on Locality Sensitive Hashing (LSH) and semantic similarity to identify similar values within the database, and adds descriptive information for each column.

The organizational structure of the database schema in the method above has not effectively improved the selection of example values. This is because not all SQL statement generations require value operations, and SQL statements involving value operations are only applied to a small subset of relevant columns. Additionally, multiple columns in the database may contain identical values, a scenario that is not distinguished in existing database schema representations. To address this, we have designed the V-Schema database schema organization framework, which further enhances the value selection process.

2.3 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 (Chang and Fosler-Lussier, 2023; Wang et al., 2023a; Rajkumar et al., 2022) 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, (Pourreza and Rafiei, 2023; Sun et al., 2023) 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 (Wei et al., 2022) , Least-to-Most prompting (Zhou et al., 2023) , and self-consistency methods (Wang et al., 2023b) .

2.4 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 (Wang et al., 2025) 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 (Zhang et al., 2024) 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 Methodology

3.1 Overall Framework

This section provides an overview of the CM-SQL architecture proposed in this paper, which consists of the following modules: (1) Database Schema Generation Module, (2) Initial SQL Generation Module, (3) SQL Check Module, (4) SQL Modification Module and (5) SQL Selection Module. The Schema Generation Module is responsible for constructing two types of database schemas: the Full-Schema and the Simplify-Schema, which are organized into the V-Schema format. These schemas are then fed into the Initial SQL Generation Module, which generates a set of potential SQL candidates. Each SQL query is input into the SQL Check module for accuracy validation. Next, the SQL Modification Module uses the output of the SQL Check Module to modify problematic SQL

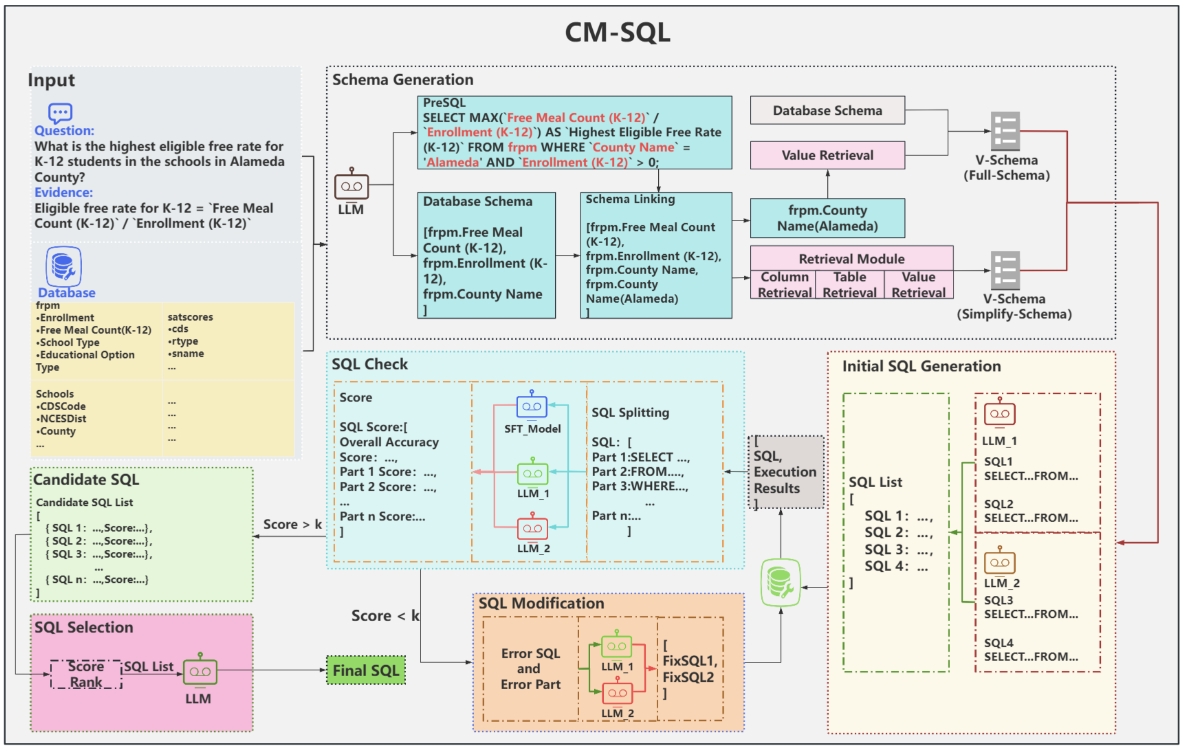

CM-SQL
Input
Question:
What is the highest eligible free rate for K-12 students in the schools in Alameda County?
Evidence:
Eligible free rate for K-12 = `Free Meal Count (K-12)` / `Enrollment (K-12)`
Database
frpm
•Enrollment
•Free Meal Count(K-12)
•School Type
•Educational Option Type
satscores
•cds
•rtype
•sname
Schools
•CDSCode
•NCESDist
•County
Schema Generation
LLM
PreSQL
SELECT MAX(`Free Meal Count (K-12)` / `Enrollment (K-12)`) AS `Highest Eligible Free Rate (K-12)` FROM frpm WHERE `County Name` = 'Alameda' AND `Enrollment (K-12)` > 0;
Database Schema
[frpm.Free Meal Count (K-12), frpm.Enrollment (K-12), frpm.County Name]
Schema Linking
[frpm.Free Meal Count (K-12), frpm.Enrollment (K-12), frpm.County Name, frpm.County Name(Alameda)]
Database Schema
Value Retrieval
frpm.County Name(Alameda)
Retrieval Module
Column Retrieval
Table Retrieval
Value Retrieval
V-Schema (Full-Schema)
V-Schema (Simplify-Schema)
SQL Check
Score
SQL Score:[Overall Accuracy Score: ..., Part 1 Score: ..., Part 2 Score: ..., ... Part n Score:...]
SFT_Model
LLM_1
LLM_2
SQL Splitting
SQL: [Part 1:SELECT ..., Part 2:FROM...., Part 3:WHERE..., ... Part n:...]
[SQL, Execution Results]
Initial SQL Generation
SQL List [SQL 1: ..., SQL 2: ..., SQL 3: ..., SQL 4: ...]
LLM_1
SQL1 SELECT...FROM...
SQL2 SELECT...FROM...
LLM_2
SQL3 SELECT...FROM...
SQL4 SELECT...FROM...
Candidate SQL
Candidate SQL List [{ SQL 1: ...,Score:...}, { SQL 2: ...,Score:...}, { SQL 3: ...,Score:...}, ... { SQL n: ...,Score:...}]
Score > k
Score < k
SQL Selection
Score Rank
SQL List
LLM
Final SQL
SQL Modification
Error SQL and Error Part
LLM_1
LLM_2
[FixSQL1, FixSQL2]

Figure 1: The overall architecture of CM-SQL consists of the following modules:(1) Database Schema Generation Module: This module generates two types of database schemas, which are represented using V-Schema.(2) Initial SQL Generation Module: Using the two database schemas as input, this module employs two different large language models (LLMs) to generate initial SQL queries.(3) SQL Check Module: This module utilizes cross-model consistency checks to assess the accuracy of the generated SQL queries and each of their components.(4) SQL Modification Module: This module employs a localized modification strategy to revise SQL queries with identified issues.(5) SQL Selection Module: Based on the output of the SQL Check Module, this module selects the final SQL query from the pool of candidates.

queries, interacting with the SQL Check Module up to k times to correct errors in the SQL queries. Finally, the candidate SQL queries are passed to the SQL Selection Module, which outputs the final SQL query. The overall architecture of CM-SQL is shown in Figure 1. In Algorithm 1 of Appendix A, we demonstrate the collaborative process of the various components in CM-SQL. The following sections provide detailed descriptions of the implementation of each module. The relevant prompt templates are provided in Appendix C.

3.2 Schema Generation

The database schema includes relevant information such as tables, columns, values, and primary/foreign key relationships within the database. In this module, we first input the entire database schema into the LLMs, instructing them to generate preliminary SQL queries and directly generate the database schema. We then extract the database schema and values used in the initial SQL via rules and take the union of the two obtained schemas as the result of schema linking. To ensure that the schema-linked database schema exists in the database, we use a two-stage verification method to validate and modify the schema after linking. The first stage validates the tables, using simple database queries to check the tables in the schema-linked database. If the query result is empty or an error occurs, we use fuzzy matching to find tables from the database that have a similarity score above a specific threshold. If no matching results are found, we directly remove the table and its associated columns. The second stage validates the columns and values, using the same verification method as in the first stage. We then assemble the results of schema linking into a Simplified-Schema in V-Schema format and combine the database schema that has not undergone schema linking with the schema-linked values to assemble a Full-Schema in V-Schema format. The overall algorithmic flow of this module is described in Algorithm 2 in Appendix A, with the specific details of V-Schema and related examples provided in Appendix B.

3.3 Candidate SQL Generation

In the initial generation of candidate SQL queries, to enhance the diversity of SQL in the candidate set, we adopt a two-pronged approach for SQL generation. First, we input both Full-Schema and Simplified-Schema database schemas into the LLM. This method alters the input to the LLM, thereby obtaining SQL queries generated under different database schemas. Second, given that different LLMs have varying training strategies and generative capabilities, we use two LLMs to generate SQL, thereby obtaining SQL queries from different model perspectives.

This approach enriches the SQL candidate set with diverse SQL queries, effectively reducing issues caused by methods such as altering model parameters. To ensure the correctness of the SQL queries in the candidate set as much as possible, each SQL query is input into the SQL check module (Section 3.4) for verification. If an error is found, the SQL modification module (Section 3.5) is used to make the necessary corrections.

3.4 SQL Check

In the preparation phase, to obtain fine-tuning data and examples for prompt engineering, we first use simple prompts on the training set to generate SQL queries with the LLM, collecting data that includes both erroneous and correct SQL queries. These SQL queries are then executed on the corresponding databases to obtain the execution results. We use the Python tool sqlparse to parse the SQL queries, and based on the parsed results, apply rules to split the SQL according to the positions of keywords. The same process is applied to handle the ground truth in the training data.

Next, we use the LLM as an assistant to compare the generated SQL queries with the ground truth and annotate each part of every SQL query in the training data, with the annotations consisting of correctness scores and the reasoning behind those scores. These annotated data are then used to fine-tune an open-source 7B model with instruction-based fine-tuning. This enables the model to understand the SQL checking process and produce stable outputs.

In the SQL checking process, we first execute the SQL query to be evaluated on the corresponding database to obtain the execution results. For excessively long database results, we select a portion of the results to serve as input for the subsequent steps. Then, we decompose each SQL query using the same splitting method as described in the preparation of training data.

Using Few-shot learning and chain-of-thought prompts, we have two LLMs assess the accuracy of the SQL. During the example selection process, we use Retrieval-Augmented Generation (RAG) to retrieve the three most relevant examples from the training data based on query similarity. This helps the LLM learn the corresponding checking process and expected output. Finally, we combine the output from these two LLMs with the output from a fine-tuned model to obtain the final SQL checking result. The process flow and an example of SQL checking are shown in Figure 4 of Appendix.

3.5 SQL Modification

For correcting erroneous SQL queries, we propose a localized SQL modification strategy. Specifically, based on the output of the checking module, we incorporate the original SQL query (PreSQL), along with its overall error causes and specific erroneous parts, into the Chain-of-Thought (CoT) prompt for the LLM. This guides the LLM to focus on correcting the erroneous parts while minimizing changes to the correct parts, gradually refining the SQL towards the correct solution.

To mitigate the potential impact of issues like LLM hallucinations on the accuracy of SQL modifications, we use two different LLMs for the modification process to generate two distinct corrected SQL queries. Drawing from relevant research (Zhang et al., 2024; Gao et al., 2024a) , we construct the prompts such that the LLMs perform step-by-step reasoning and output both the modified SQL and the reasoning process in JSON format. This not only enhances the model's ability to modify the SQL query but also makes the modification process more transparent and traceable. The entire SQL modification process is shown in Figure 5 of Appendix.

After the modification, we pass the corrected SQL query into the SQL accuracy checking module to evaluate the accuracy of the modified query. If the modified SQL is still incorrect, the modification process continues until it approaches the correct solution. To avoid repeated errors during multiple rounds of modification, we also pass the historical modification information and error details to the LLM, prompting it to avoid making the same mistakes. This helps maximize the likelihood of successful modifications.

To prevent the modification process from entering a loop of redundant corrections, we limit the maximum number of modification attempts to 5. If the SQL query still fails to be corrected after these attempts, the modification process is terminated. This ensures that resources are used efficiently while still striving to improve the SQL generation and correction accuracy.

3.6 SQL Selection

In this study, for the final selection of SQL, we first execute each SQL query in the SQL candidate set on the corresponding database to obtain the execution results. For SQL queries with identical execution results, we retain the one with the highest score based on the output from the checking module. If the SQL candidate set contains only two queries and they produce the same execution result, the SQL query with the higher score is directly selected as the final SQL.

Considering that LLMs tend to choose the first option in multiple-choice tasks (Wang et al., 2023c; Zheng et al., 2023) , when there are more than two SQL queries in the candidate set, we first sort the SQL queries based on the scores output by the checking module. We then input the sorted SQL queries along with their partial execution results into the LLM, allowing it to select the final SQL query. The specific algorithmic flow of this module is described in Algorithm 3 in Appendix A.

4 Experiments

4.1 Experimental Setup

4.1.1 Datasets

BIRD (Benchmark for Intelligent Retrieval and Database) (Li et al., 2023b) is a large-scale, real-world database text-to-SQL evaluation benchmark released by Alibaba DAMO Academy. BIRD is designed to advance the development of the Text-to-SQL task, particularly in evaluating performance in real-world database environments. The benchmark includes high-quality Text-to-SQL pairs from 95 large-scale databases, with a total database size of 33.4GB, spanning 37 professional fields, such as finance, e-commerce, healthcare, and more. As such, BIRD is one of the most challenging benchmarks for Text-to-SQL, providing diverse database types and rich business scenarios, and enabling comprehensive evaluation of models' capabilities in handling complex queries.

4.1.2 Base Model

This paper conducts relevant experiments using two large language models and a fine-tuned 7B open-source model. Specifically, GPT-4o-mini (OpenAI, 2024) and DeepSeek-V2.5 (Liu et al., 2024) are employed as base models for SQL generation, SQL accuracy checking, and SQL candidate selection. The Qwen2.5-7B-Instruct (Yang et al., 2024) model, fine-tuned for this task, is used as an auxiliary model to enhance the functionality of SQL accuracy checking and SQL candidate selection.

4.2 Evaluation Metrics

Following previous related work (Wang et al., 2024; Ren et al., 2024) , this paper adopts Execution Accuracy (EX) to evaluate the effectiveness of the generated SQL. EX measures the proportion of questions in the evaluation set where the execution results of both the predicted and ground-truth inquiries are identical, relative to the total number of queries.

4.3 Comparison with Existing Methods

Table 1 presents the execution accuracy of our proposed method compared to the competitive approaches on the BIRD dataset. Compared to methods such as RSL-SQL and TA-SQL, which use schema linking techniques to enhance SQL generation quality, our method outperforms them by 2.29% and 9.46%, respectively, in execution accuracy. Compared to methods like MAC-SQL and SQLFixAgent, which assess SQL accuracy through model self-consistency, our method achieves an improvement of 8.09% and 5.39% in execution accuracy, respectively. Compared to MCS-SQL, which enriches SQL candidates and selects SQL queries by constructing different prompts, our method improves execution accuracy by 2.09%. Additionally, it is noteworthy that our method uses only two low-cost LLMs, achieving performance comparable to that of more expensive models such as GPT-4 or GPT-4o, demonstrating both the effectiveness and cost-efficiency of the proposed CM-SQL framework for SQL generation.

4.4 Effectiveness of the SQL Check Module

To verify the effectiveness of the SQL accuracy checking method we proposed, we collected the

Method	EX(%)
MAC-SQL(Wang et al., 2025) + GPT-4	57.56
RSL-SQL(Cao et al., 2024) + DeepSeek	63.36
TA-SQL(Qu et al., 2024) + GPT-4	56.19
SQLFixAgent(Cen et al., 2024) + GPT-3.5 Turbo	60.17
MCS-SQL(Lee et al., 2025) + GPT-4	63.56
E-SQL(Caferoğlu and Ulusoy, 2024) + GPT-4o-mini	61.60
CM-SQL + DeepSeek + GPT-4o-mini(our)	**65.65**

Table 1: The pass rate of SQL accuracy assessment on the BIRD Dev set using self-consistency methods by DeepSeek and GPT-4o-mini.

Method	Simple	Moderate	Challenging	Total
DDL	69.41	54.19	40.28	62.06
MAC-Schema	72.23	56.28	46.86	65.01
V-Schema	72.54	57.63	47.22	**65.65**

Table 2: Execution accuracies with different difficulties on the Dev set of BIRD in the CM-SQL framework using three database schema representations.

results from the first four rounds of the SQL generation process as a test dataset. We then evaluated the following SQL accuracy assessment methods: (1) LLM Self-consistency evaluation: The SQL query and partial execution results are input into the LLM, which performs accuracy checking on the SQL query. (2) Cross-consistency evaluation with two LLMs: The SQL query and partial execution results are input into two LLMs and the combined output from both models is used to assess the SQL accuracy. (3) Evaluation using a fine-tuned model: A 7B model is fine-tuned using instructions to evaluate the accuracy of the SQL query. (4) Our method: The SQL query is first executed and split. The split results and partial execution results are then input into two LLMs and a fine-tuned 7B model. The combined output of these three models is used to assess SQL accuracy.

The experimental results, shown in Figure 6 of the Appendix, demonstrate that our method performs best in identifying SQL queries that are syntactically correct but contain semantic errors. Additionally, the overall accuracy of SQL accuracy judgment reaches the optimal level. However, it is worth noting that as the number of modifications increases, the number of erroneous SQL queries in the candidate set gradually decreases. This is especially true for SQL queries with syntax errors, which are almost all corrected into syntactically correct SQL queries. Due to factors such as the limitations of model capabilities, even if all erroneous SQL queries were judged as correct, the overall performance would still show little difference.

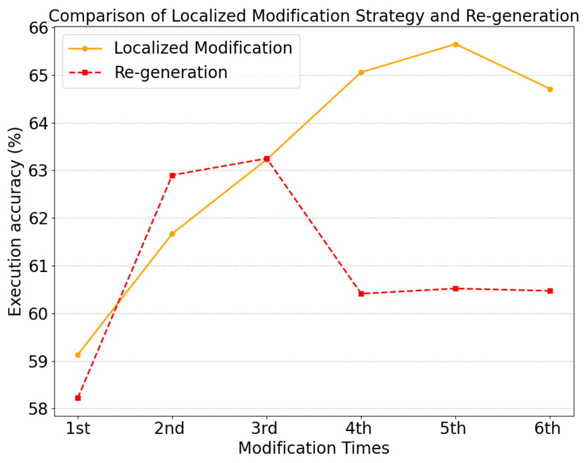


Figure 2: Execution accuracy per modification on BIRD's Dev set.

4.5 Results of Multiple Rounds of Modifications

To validate the effectiveness of our proposed local modification strategy, we conducted a comparative experiment with the traditional regeneration approach, analyzing their performance over six rounds of modification. As shown in Figure 2, during the process using the regeneration method, although some erroneous SQL queries could be corrected in the first three rounds, the performance gradually declined as the number of modification rounds increased. Notably, after the fourth round, the regeneration method even transformed most of the originally correct SQL queries into incorrect ones. In contrast, our proposed local modification strategy exhibited positive effects in the first five rounds of modification. Despite a performance decline in the sixth round due to factors such as model limitations, its overall performance still outperformed the traditional approach. It is worth mentioning that, due to the inability of the check module to completely filter out all correct SQL queries, some correct queries were mistakenly identified as erroneous and underwent unnecessary modifications. The experimental results indicate that our proposed local modification strategy significantly outperforms the traditional regeneration approach in terms of modification capability, while better preserving the correctness of the original SQL queries.

4.6 V-Schema

To demonstrate the effectiveness of the V-Schema we proposed, we compared the performance of us-

Method	Simple	Moderate	Challenging	Total
Full pipeline	72.54	57.63	47.22	65.65
w/o Full-Schema	70.38	55.05	43.75	63.23(↓2.42)
w/o Simplify-Schema	70.16	53.33	45.14	62.71(↓2.94)
w/o Multi-Model SQL Generation	70.49	55.27	42.36	63.23(↓2.42)
w/o SQL Check & Modification	68.65	54.19	40.97	61.67(↓3.98)
w/o SQL Selection	70.27	54.41	42.36	62.84(↓2.81)

Table 3: Execution accuracy results for ablation of different parts of CM-SQL on the Dev set of BIRD. ↓ Indicates a decrease in the accuracy rate.

ing Database Definition Language (DDL) , MAC-Schema, and our V-Schema on the DEV set of BIRD. As shown in Table 2, we evaluated the performance of the three database schemas across different levels of difficulty. From the experimental results, it is evident that the V-Schema method we employed outperformed the DDL-based schema expression by 3.13%, 3.44%, and 6.94% on Simple, Moderate, and Challenging difficulties, respectively, with an overall improvement of 3.59%. Compared to the MAC-Schema, V-Schema showed improvements of 0.31%, 1.35%, and 0.36%, with an overall increase of 0.64%. This further validates the effectiveness of V-Schema and also highlights that excessive irrelevant values in the input can negatively impact the SQL generation performance of LLMs.

4.7 Ablation Study

To demonstrate the effectiveness of each component in our proposed CM-SQL framework, we conducted an ablation study on the DEV set of the BIRD dataset, evaluating the impact of each component on overall performance through execution accuracy. As shown in Table 3, the absence of the SQL Check module resulted in a 3.98% decrease in overall performance, proving that the cross-consistency check between the fine-tuned model and LLMs plays a crucial role in ensuring the correctness of the generated SQL and significantly enhances the overall architecture's performance.

The removal of the Full-Schema and Simplify-Schema components led to performance drops of 2.42% and 2.94%, respectively, highlighting the importance of providing the complete database schema and the impact of simplifying the schema through schema linking on the model's accuracy. These findings suggest that different schema representations have varied influences on the model's ability to generate accurate SQL queries.

Furthermore, the absence of the Multi-Model SQL Generation and SQL Selection modules resulted in performance drops of 2.42% and 2.81%, respectively. This demonstrates the importance of model collaboration, where the interaction between different models helps mitigate their individual weaknesses, ultimately improving the accuracy of the generated SQL queries.

In summary, the ablation study conducted on the BIRD DEV set confirms the effectiveness and adaptability of each component in our proposed CM-SQL framework for challenging text-to-SQL tasks, such as those presented by the BIRD dataset. Each component contributes significantly to the overall performance, underlining the value of our integrated approach in tackling complex SQL generation tasks.

5 CONCLUSION

In this paper, we propose the CM-SQL architecture, which enriches the SQL candidate set by altering the input database schema for LLMs and leveraging the generative capabilities of different LLMs. For SQL accuracy assessment, we employ a method that first splits the SQL query and then uses cross-consistency checks with LLMs and fine-tuned models to evaluate each part of the query. Experimental results demonstrate the effectiveness of this checking method. For error correction in SQL queries, we propose a local modification strategy to adjust the erroneous parts of the SQL, and experiments show that this approach is more stable and effective than existing methods that regenerate SQL queries. Through comparative experiments and ablation studies, the effectiveness of the CM-SQL architecture and the capabilities of its modules in handling related tasks are demonstrated.

6 Limitations

This paper has two main limitations, which highlight areas for future work:

- We proposed the use of cross-consistency between multiple LLMs and fine-tuned models for SQL accuracy checking, and experimental results demonstrate its effectiveness. However, in this study, only two LLMs and a fine-tuned 7B model were used for the related experiments. More LLMs and models with larger fine-tuning parameters were not explored, so the upper limit of this method remains undetermined.

- The local modification strategy we proposed has been proven effective in experiments, but its capability is limited by the SQL checking method. Therefore, further improvements and innovations are needed in future work to minimize the reliance on the SQL checking method.

References

Aseem Arora, Shabbirhussain Bhaisaheb, Harshit Nigam, Manasi Patwardhan, Lovekesh Vig, and Gautam Shroff. 2023. Adapt and decompose: Efficient generalization of text-to-sql via domain adapted least-to-most prompting. In *Proceedings of the 1st GenBench Workshop on (Benchmarking) Generalisation in NLP*, pages 25–47.

Hasan Alp Caferoğlu and Özgür Ulusoy. 2024. E-sql: Direct schema linking via question enrichment in text-to-sql. *arXiv preprint arXiv:2409.16751*.

Zhenbiao Cao, Yuanlei Zheng, Zhihao Fan, Xiaojin Zhang, and Wei Chen. 2024. Rsl-sql: Robust schema linking in text-to-sql generation. *arXiv preprint arXiv:2411.00073*.

Jipeng Cen, Jiaxin Liu, Zhixu Li, and Jingjing Wang. 2024. Sqlfixagent: Towards semantic-accurate sql generation via multi-agent collaboration. *arXiv preprint arXiv:2406.13408*.

Shuaichen Chang and Eric Fosler-Lussier. 2023. 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*.

Xiaojun Chen, Tianle Wang, Tianhao Qiu, Jianbin Qin, and Min Yang. 2024. Open-sql framework: Enhancing text-to-sql on open-source large language models. *arXiv preprint arXiv:2405.06674*.

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

Catherine A Gao, Frederick M Howard, Nikolay S Markov, Emma C Dyer, Siddhi Ramesh, Yuan Luo, and Alexander T Pearson. 2022. Comparing scientific abstracts generated by chatgpt to original abstracts using an artificial intelligence output detector, plagiarism detector, and blinded human reviewers. *BioRxiv*, pages 2022–12.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024a. Text-to-sql empowered by large language models: A benchmark evaluation. *Proc. VLDB Endow.*, 17(5):1132–1145.

Shuzheng Gao, Xin-Cheng Wen, Cuiyun Gao, Wenxuan Wang, and Michael R Lyu. 2023a. Constructing effective in-context demonstration for code intelligence tasks: An empirical study. *CoRR*.

Shuzheng Gao, Xin-Cheng Wen, Cuiyun Gao, Wenxuan Wang, Hongyu Zhang, and Michael R Lyu. 2023b. What makes good in-context demonstrations for code intelligence tasks with llms? In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 761–773. IEEE.

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

Zihui Gu, Ju Fan, Nan Tang, Songyue Zhang, Yuxin Zhang, Zui Chen, Lei Cao, Guoliang Li, Sam Madden, and Xiaoyong Du. 2023. Interleaving pre-trained language models and large language models for zero-shot nl2sql generation. *arXiv preprint arXiv:2306.08891*.

Wenxiang Jiao, Wenxuan Wang, Jen-tse Huang, Xing Wang, and Zhaopeng Tu. 2023. Is chatgpt a good translator? a preliminary study. *arXiv preprint arXiv:2301.08745*, 1(10).

Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. 2025. MCS-SQL: leveraging multiple prompts and multiple-choice selection for text-to-sql generation. In *Proceedings of the 31st International Conference on Computational Linguistics, COLING 2025, Abu Dhabi, UAE, January 19-24, 2025*, pages 337–353. Association for Computational Linguistics.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024a. Codes: Towards building open-source language models for text-to-sql. *Proceedings of the ACM on Management of Data*, 2(3):127.

Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo

Si, and Yongbin Li. 2023a. Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, pages 13076–13084. AAAI Press.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chen-Chuan Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023b. 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 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Zhishuai Li, Xiang Wang, Jingjing Zhao, Sun Yang, Guoqing Du, Xiaoru Hu, Bin Zhang, Yuxiao Ye, Ziyue Li, Rui Zhao, et al. 2024b. Pet-sql: A prompt-enhanced two-stage text-to-sql framework with cross-consistency. *arXiv preprint arXiv:2403.09732*.

Zhen Lin, Shubhendu Trivedi, and Jimeng Sun. 2024. Generating with confidence: Uncertainty quantification for black-box large language models. *Trans. Mach. Learn. Res.*, 2024.

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. 2024. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*.

Linyong Nan, Yilun Zhao, Weijin Zou, Narutatsu Ri, Jaesung Tae, Ellen Zhang, Arman Cohan, and Dragomir Radev. 2023. Enhancing text-to-sql capabilities of large language models: A study on prompt design strategies. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 14935–14956. Association for Computational Linguistics.

Gpt OpenAI. 2024. 4o mini: Advancing cost-efficient intelligence, 2024. *URL: https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence*.

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

Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Ge Qu, Jinyang Li, Bowen Li, Bowen Qin, Nan Huo, Chenhao Ma, and Reynold Cheng. 2024. 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*, pages 5456–5471.

Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. 2022. Evaluating the text-to-sql capabilities of large language models. *arXiv preprint arXiv:2204.00498*.

Tonghui Ren, Yuankai Fan, Zhenying He, Ren Huang, Jiaqi Dai, Can Huang, Yinan Jing, Kai Zhang, Yifan Yang, and X. Sean Wang. 2024. PURPLE: making a large language model a better SQL writer. In *40th IEEE International Conference on Data Engineering, ICDE 2024, Utrecht, The Netherlands, May 13-16, 2024*, pages 15–28. IEEE.

Shouvon Sarker, Xishuang Dong, Xiangfang Li, and Lijun Qian. 2024. Enhancing llm fine-tuning for text-to-sqls by sql quality measurement. *arXiv preprint arXiv:2410.01869*.

Ruoxi Sun, Sercan Ö Arik, Alex Muzio, Lesly Miculicich, Satya Gundabathula, Pengcheng Yin, Hanjun Dai, Hootan Nakhost, Rajarishi Sinha, Zifeng Wang, et al. 2023. Sql-palm: Improved large language model adaptation for text-to-sql (extended). *arXiv preprint arXiv:2306.00739*.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. Chess: Contextual harnessing for efficient sql synthesis. *arXiv preprint arXiv:2405.16755*.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2025. 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*, pages 540–557. Association for Computational Linguistics.

Tianshu Wang, Hongyu Lin, Xianpei Han, Le Sun, Xiaoyang Chen, Hao Wang, and Zhenyu Zeng. 2023a. Dbcopilot: Scaling natural language querying to massive databases. *arXiv preprint arXiv:2312.03463*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023b. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Yiwei Wang, Yujun Cai, Muhao Chen, Yuxuan Liang, and Bryan Hooi. 2023c. Primacy effect of chatgpt. *arXiv preprint arXiv:2310.13206*.

Zhongyuan Wang, Richong Zhang, Zhijie Nie, and Jaein Kim. 2024. Tool-assisted agent on sql inspection and refinement in real-world scenarios. *arXiv preprint arXiv:2408.16991*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, pages 24824–24837.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, et al. 2024. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*.

Bin Zhang, Yuxiao Ye, Guoqing Du, Xiaoru Hu, Zhishuai Li, Sun Yang, Chi Harold Liu, Rui Zhao, Ziyue Li, and Hangyu Mao. 2024. Benchmarking the text-to-sql capability of large language models: A comprehensive evaluation. *arXiv preprint arXiv:2403.02951*.

Chujie Zheng, Hao Zhou, Fandong Meng, Jie Zhou, and Minlie Huang. 2023. Large language models are not robust multiple choice selectors. In *The Twelfth International Conference on Learning Representations*.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, and Ed H. Chi. 2023. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*.

A Algorithm Workflow

Algorithm 1 The algorithm of CM-SQL

```
1: Input: question $q$, database schema $sc$, knowledge $k$
2: Output: sql
3: Full_Schema, Simplify_Schema = schemaGeneration($q$, $sc$, $k$)
4: preSqlList = []
5: preSqlList.append(InitialSQLGeneration(Full_Schema))
6: preSqlList.append(InitialSQLGeneration(Simplify_Schema))
7: candidateSql = []
8: for sql in preSqlList do
9:     modifyHistory = []
10:    while len(modifyHistory) $<$ maxTryTimes do
11:        exeResult = databaseExe(sql)
12:        success, error = SqlCheck(sql, exeResult)
13:        if success then
14:            candidateSql.append(sql)
15:            break
16:        else
17:            modifyHistory.append([sql, error])
18:            sql = sqlModification(sql, error, modifyHistory)
19:        end if
20:    end while
21: end for
22: return sqlSelection(candidateSql)
```

Algorithm 2 The algorithm of Schema Generation

```
1: Input: question $q$, database schema $sc$, knowledge $k$
2: Output: V-Schema
3: Table, Column, Value = [], [], []
4: preSql = LLM($q$, $sc$, $k$)
5: t, c = LLM($q$, $sc$, $k$)
6: table, column, value = sqlParse(preSql)
7: T, C, V = t + table, c + column, v
8: for each $t$, $c$, $v$ in T, C, V do
9:     if databaseExe(SELECT * FROM t) then
10:        Table.append(t)
11:    end if
12:    if databaseExe(SELECT t FROM c) then
13:        Column.append(c)
14:    end if
15:    if databaseExe(SELECT * FROM t WHERE c == v) then
16:        Value.append(LLM($sc$, v))
17:    end if
18: end for
19: return VSchema(Table, Column, Value)
```

Algorithm 3 The algorithm of SQL Selection

```
1: Input: SQL Candidate Set $sqlList$, V-Schema $sc$, question $q$, knowledge $k$
2: Output: final_sql
3: sort($sqlList$)                 ▷ Sort by the SQL check scores
4: sqlList = uniqueSQL($sqlList$)   ▷ Remove duplicates, keep the highest scored SQL
5: if len($sqlList$) = 1 then
6:     return $sqlList[0]$
7: else
8:     return LLM($q$, $sc$, $k$, $sqlList$)
9: end if
```

B V-Schema Database Schema Organization

We refer to methods from MAC-SQL(Wang et al., 2025) and XIYAN (Gao et al., 2024b) to design a schema representation format called "V-Schema". In V-Schema, a semi-structured schema representation method is employed to express the logical relationships between databases, tables, and columns, with retrieved values and their locations appended at the end. Specifically, we use "[DB_ID]" to represent the database name, "[Schema]" as a delimiter for the database schema, and "#Table:" to specify the table name and its structure. The foreign key relationships within the database are denoted by "[Foreign keys]".

Unlike MAC-SQL and XIYAN-SQL, in order to reduce the interference of unnecessary information on the model, we do not provide example values for every column. Instead, we only include values retrieved through the retrieval modules and use LLMs to generate a description for these values, distinguishing the differences between the retrieved values and clarifying their meaning within the database. Figure 3 shows an example of Schema for MAC-SQL and M-Schema for XIYAN-SQL with our proposed VSchema.

C Prompt Template

Table 4 presents the prompt templates for generating SQL, Table 5 provides the prompt templates for SQL accuracy checking, and Tables 6 and 7 show the prompt templates for modifying SQL and selecting SQL from the SQL candidate set, respectively.

D Supplementary Figures

Figure 3 shows four types of database schema formats, illustrating the expression of the database schemas under the query "What is the amount of Mike's order on 16 September 2024?". The parts highlighted in red represent the main differences between our proposed V-Schema and the other three schema formats. The remaining three schemas do not apply any special handling to the selection of example values. Additionally, the database organization in V-Schema is the most concise, removing a large number of redundant example values while ensuring sufficient information is provided.

Figures 4 and 5, respectively, illustrate the verification process and the modification process for the syntactically correct but semantically incorrect SQL query "SELECT MAX(Percent (%) Eligible Free (K-12)) FROM frpm WHERE County Name = 'Alameda';" generated under the query "What is the highest eligible free rate for K-12 students in the schools in Alameda County?".

```
###You are an expert in databases, Based on the provided information, provide the best SQL statement.
###No explanation or extension is needed.
##Database schema:
{Full-Schema/Simplify-Schema}
##Evidence:
{evidence}
##Question:
{question}
###Requirement:
1. The generated SQL statements need to be executable on SQLite databases.
2. Give the thought process when generating SQL and reflect on it.
3. You must ensure that the output result can be parsed by python's json.loads.
4. Let's think step by step.
##Output:
{{
"SQL":"SELECT...",
"Reason":"...",
"Reflection:":"..."
}}
```

Table 4: SQL prompt template for initial generation.

```
###You are an expert in SQLite database. Based on the information provided, judge the accuracy
of the given SQL statement and each part of the split SQL statement, and give scores and reasons.
###No explanation or extension is needed.
##Database schema:
{Full-Schema/Simplify-Schema}
##Evidence:
{evidence}
##Question:
{question}
##SQL:
{SQL}
##Split SQL statement::
{Split-SQL}
##The result of SQL execution:
{ex-result}
##Example:
{Few-shot}
###Requirement:
1. Carefully analyze whether the given SQL statement and each part after splitting meet the
requirements of the question.
2. If you think this part is correct, then give it a score higher than 80.
Otherwise, give a reasonable score lower than 80 based on your analysis.
The score ranges from 0 to 100.
3. You must ensure that the output result can be parsed by python's json.loads.
4. Let's think step by step.
##Output:
{{
{"SQL":"SELECT...",
"Score":"...",
"Reason:":"..."},
{"Part 1":"...",
"Score":"...",
"Reason:":"..."},
...
}}
```

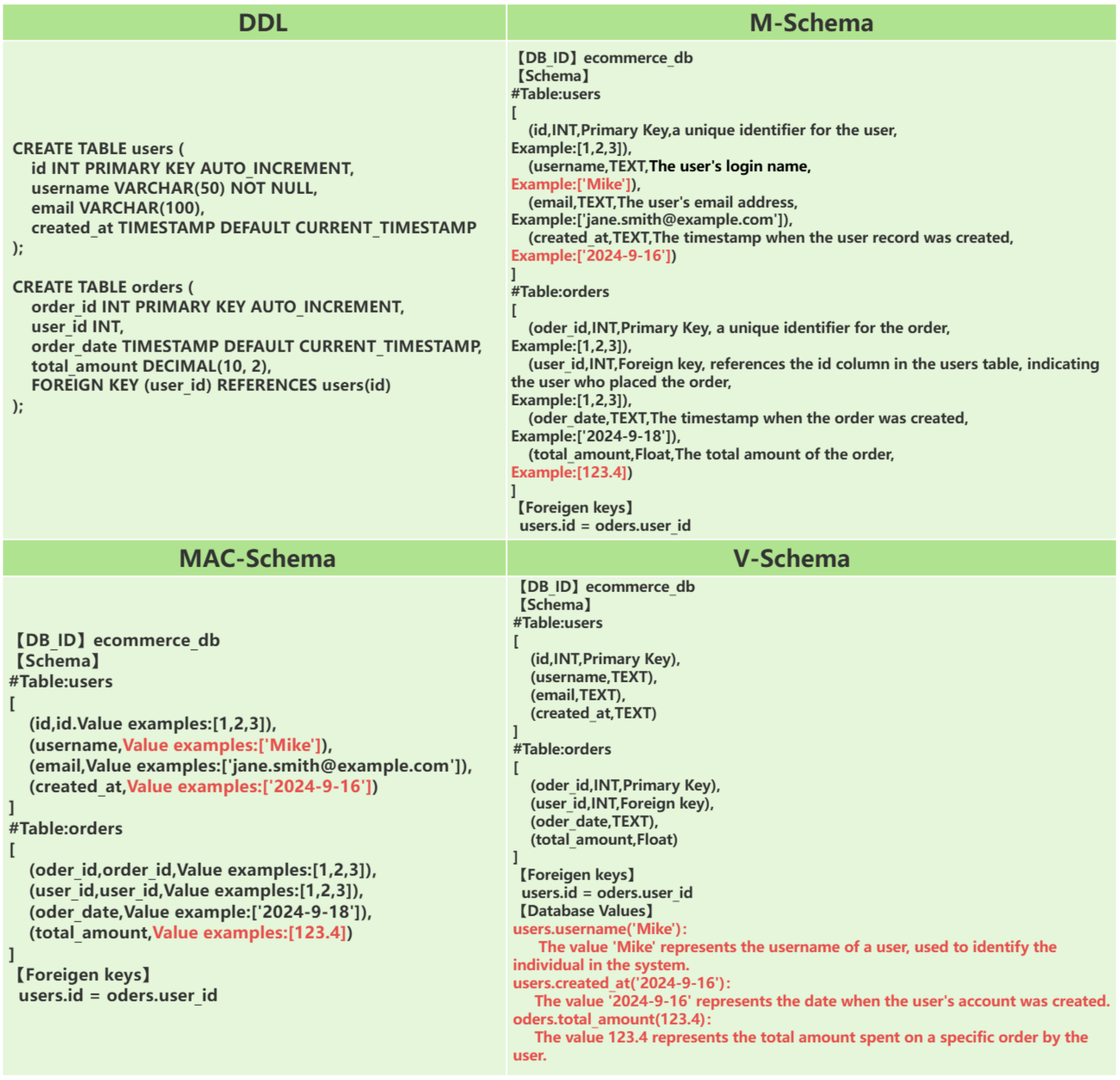
Table 5: SQL accuracy verification prompt template.

```
###You are an expert in databases. Modify the SQL statement according to the given information.
###No explanation or extension is needed.
##Database schema:
{Full-Schema/Simplify-Schema}
##Evidence:
{evidence}
##Question:
{question}
##SQL:
{SQL}
##This is the part of the SQL you need to modify:
{error-part}
##SQL modification history:
{history}
###Requirement:
1. During the modification process, only the given error parts need to be modified.
2. Give the reason for the modification and reflect on the entire SQL modification process.
3. You must ensure that the output result can be parsed by python's json.loads.
4. Let's think step by step.
##Output:
{{
"SQL":SELECT...,
"Reason":...,
"Reflection:":...
}}
```

Table 6: SQL modification prompt template.

```
###You are an expert in databases, Based on the information given,
select the SQL statement that best matches the question from the SQL list.
###No explanation or extension is needed.
##Database schema:
{Full-Schema/Simplify-Schema}
##Evidence:
{evidence}
##Question:
{question}
##SQL List:
{Sort-SQL-list}
###Requirement:
1. Give reasons for choosing SQL and reflect on the process.
2. You must ensure that the output result can be parsed by python's json.loads.
3. Let's think step by step.
##Output:
{{
"SQL":"SELECT...",
"Reason":"...",
"Reflection:":"..."
}}
```

Table 7: SQL selection prompt template.



DDL

```
CREATE TABLE users (
    id INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(50) NOT NULL,
    email VARCHAR(100),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE orders (
    order_id INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT,
    order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    total_amount DECIMAL(10, 2),
    FOREIGN KEY (user_id) REFERENCES users(id)
);
```

M-Schema

```
【DB_ID】ecommerce_db
【Schema】
#Table:users
[
    (id,INT,Primary Key,a unique identifier for the user,
Example:[1,2,3]),
    (username,TEXT,The user's login name,
Example:['Mike']),
    (email,TEXT,The user's email address,
Example:['jane.smith@example.com']),
    (created_at,TEXT,The timestamp when the user record was created,
Example:['2024-9-16'])
]
#Table:orders
[
    (oder_id,INT,Primary Key, a unique identifier for the order,
Example:[1,2,3]),
    (user_id,INT,Foreign key, references the id column in the users table, indicating
the user who placed the order,
Example:[1,2,3]),
    (oder_date,TEXT,The timestamp when the order was created,
Example:['2024-9-18']),
    (total_amount,Float,The total amount of the order,
Example:[123.4])
]
【Foreigen keys】
 users.id = oders.user_id
```

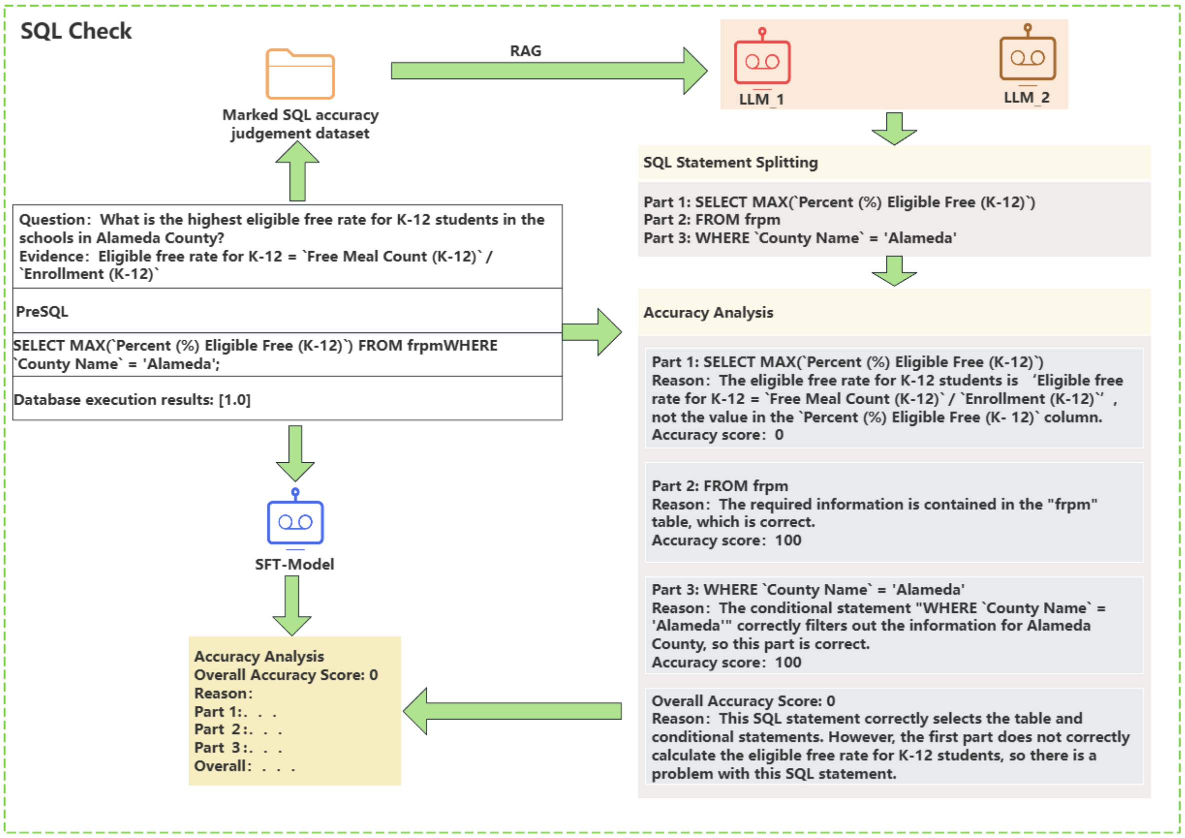
MAC-Schema

```
【DB_ID】ecommerce_db
【Schema】
#Table:users
[
    (id,id.Value examples:[1,2,3]),
    (username,Value examples:['Mike']),
    (email,Value examples:['jane.smith@example.com']),
    (created_at,Value examples:['2024-9-16'])
]
#Table:orders
[
    (oder_id,order_id,Value examples:[1,2,3]),
    (user_id,user_id,Value examples:[1,2,3]),
    (oder_date,Value example:['2024-9-18']),
    (total_amount,Value examples:[123.4])
]
【Foreigen keys】
 users.id = oders.user_id
```

V-Schema

```
【DB_ID】ecommerce_db
【Schema】
#Table:users
[
    (id,INT,Primary Key),
    (username,TEXT),
    (email,TEXT),
    (created_at,TEXT)
]
#Table:orders
[
    (oder_id,INT,Primary Key),
    (user_id,INT,Foreign key),
    (oder_date,TEXT),
    (total_amount,Float)
]
【Foreigen keys】
 users.id = oders.user_id
【Database Values】
users.username('Mike'):
    The value 'Mike' represents the username of a user, used to identify the
individual in the system.
users.created_at('2024-9-16'):
    The value '2024-9-16' represents the date when the user's account was created.
oders.total_amount(123.4):
    The value 123.4 represents the total amount spent on a specific order by the
user.
```

Figure 3: Example of the 4 database schemas when the user question is ‘What is the amount of Mike’s order on 16 September 2024?’ The red text highlights the main differences between the V-Schema and the other three database schema representations.



Question: What is the highest eligible free rate for K-12 students in the schools in Alameda County?
Evidence: Eligible free rate for K-12 = `Free Meal Count (K-12)` / `Enrollment (K-12)`

PreSQL

SELECT MAX(`Percent (%) Eligible Free (K-12)`) FROM frpmWHERE `County Name` = 'Alameda';

Database execution results: [1.0]

Accuracy Analysis
Overall Accuracy Score: 0
Reason:
Part 1:. . .
Part 2:. . .
Part 3:. . .
Overall: . . .

SQL Statement Splitting

Part 1: SELECT MAX(`Percent (%) Eligible Free (K-12)`)
Part 2: FROM frpm
Part 3: WHERE `County Name` = 'Alameda'

Accuracy Analysis

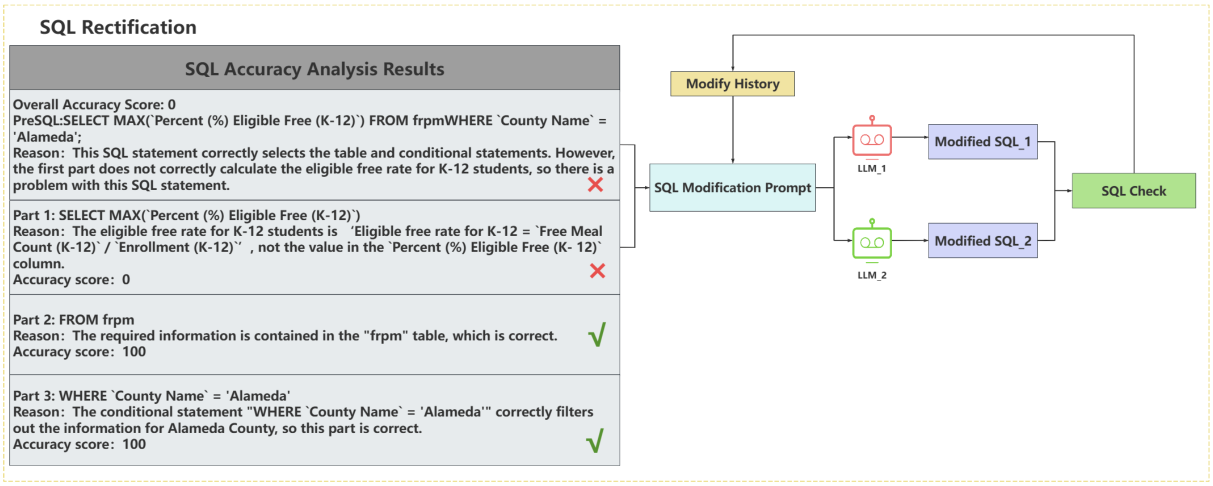
Part 1: SELECT MAX(`Percent (%) Eligible Free (K-12)`)
Reason: The eligible free rate for K-12 students is 'Eligible free rate for K-12 = `Free Meal Count (K-12)` / `Enrollment (K-12)`' , not the value in the `Percent (%) Eligible Free (K- 12)` column.
Accuracy score: 0

Part 2: FROM frpm
Reason: The required information is contained in the "frpm" table, which is correct.
Accuracy score: 100

Part 3: WHERE `County Name` = 'Alameda'
Reason: The conditional statement "WHERE `County Name` = 'Alameda'" correctly filters out the information for Alameda County, so this part is correct.
Accuracy score: 100

Overall Accuracy Score: 0
Reason: This SQL statement correctly selects the table and conditional statements. However, the first part does not correctly calculate the eligible free rate for K-12 students, so there is a problem with this SQL statement.

Figure 4: SQL Accuracy Check Diagram.



SQL Accuracy Analysis Results

Overall Accuracy Score: 0
PreSQL:SELECT MAX(`Percent (%) Eligible Free (K-12)`) FROM frpmWHERE `County Name` = 'Alameda';
Reason: This SQL statement correctly selects the table and conditional statements. However, the first part does not correctly calculate the eligible free rate for K-12 students, so there is a problem with this SQL statement. ✗

Part 1: SELECT MAX(`Percent (%) Eligible Free (K-12)`)
Reason: The eligible free rate for K-12 students is 'Eligible free rate for K-12 = `Free Meal Count (K-12)` / `Enrollment (K-12)`' , not the value in the `Percent (%) Eligible Free (K- 12)` column.
Accuracy score: 0 ✗

Part 2: FROM frpm
Reason: The required information is contained in the "frpm" table, which is correct.
Accuracy score: 100 ✓

Part 3: WHERE `County Name` = 'Alameda'
Reason: The conditional statement "WHERE `County Name` = 'Alameda'" correctly filters out the information for Alameda County, so this part is correct.
Accuracy score: 100 ✓

Figure 5: SQL Modification Diagram.

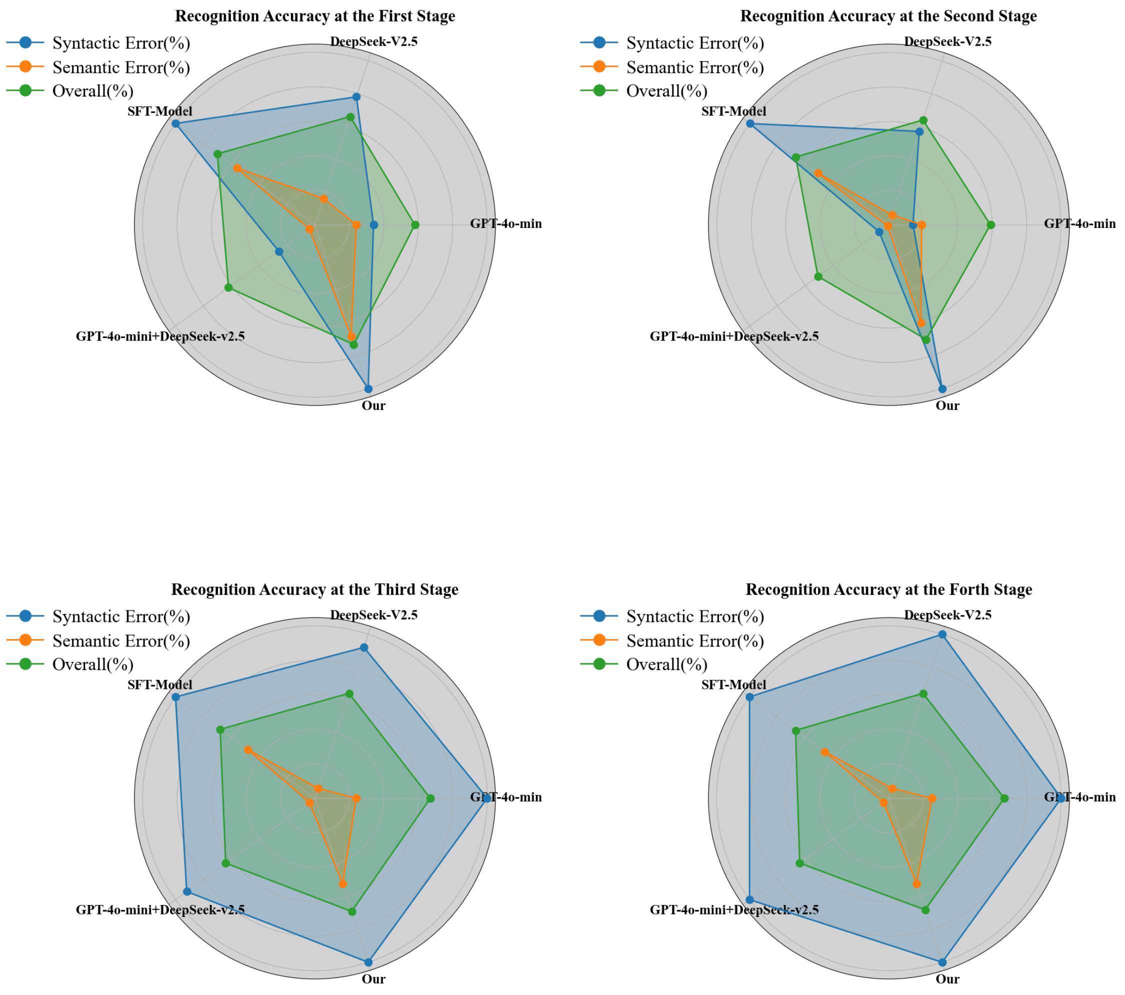


Figure 6: Performance of various judgement methods in the four intermediate outcomes.