GBV-SQL: Guided Generation and SQL2Text Back-Translation Validation for Multi-Agent Text2SQL

Anonymous ACL submission

Abstract

While Large Language Models have significantly advanced Text2SQL generation, a critical semantic gap persists where syntactically valid queries often misinterpret user intent. To mitigate this challenge, we propose GBV-SQL, a novel multi-agent framework that introduces Guided Generation with SQL2Text Back-translation Validation. This mechanism uses a specialized agent to translate the generated SQL back into natural language, which verifies its logical alignment with the original question. Critically, our investigation reveals that current evaluation is undermined by a systemic issue: the poor quality of the benchmarks themselves. We introduce a formal typology for "Gold Errors", which are pervasive flaws in the ground-truth data, and demonstrate how they obscure true model performance. On the challenging BIRD benchmark, GBV-SQL achieves 63.23% execution accuracy, a 5.8% absolute improvement. After removing flawed examples from Spider and repairing flawed examples in BIRD, GBV-SQL achieves 96.5% (dev) and 97.6% (test) execution accuracy on Spider, and 90.42% on the corrected BIRD dataset. Our work offers both a robust framework for semantic validation and a critical perspective on benchmark integrity, highlighting the need for more rigorous dataset curation.

1 Introduction

Text2SQL is a challenging task that involves automatically converting a Natural Language Question (NLQ) into Structured Query Language (SQL) that can be executed on a relational database (Zelle and Mooney, 1996; Qin et al., 2022). The primary goal is to enable non-technical users to interact with complex databases using only natural language, removing the barrier of learning SQL syntax. As a long-standing objective in both natural language processing and database research

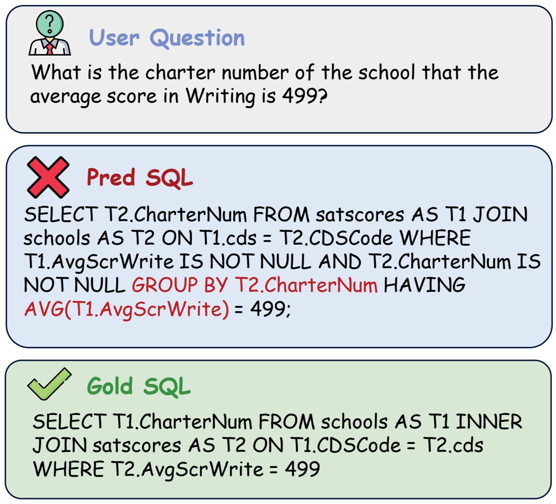


Figure 1: A BIRD dev example where syntactically valid Pred SQL semantically misinterprets the query: it unnecessarily aggregates "average score" by charter groups, while the user explicitly seeks a single school with a score of 499.

(Nguyen et al., 2023; Zhu et al., 2023), this field has been profoundly reshaped by recent advancements in Large Language Models (LLMs). These models, particularly through the paradigm of in-context learning (ICL), now represent the state-of-the-art, often achieving performance that surpasses traditional fine-tuned approaches (Pourreza and Rafiei, 2023; Li et al., 2023a; Liu et al., 2024).

Despite this significant progress, a critical challenge persists: ensuring the generated SQL query is not only syntactically correct but also semantically faithful to the user's original intent. This challenge stems from the inherent semantic gap between an unstructured Natural Language Question (NLQ) and the formal structure of SQL, which remains a primary source of failure. A syntactically valid query can still misinterpret the user's goal, producing plausible but erroneous results that are difficult to detect. For instance, as illustrated in

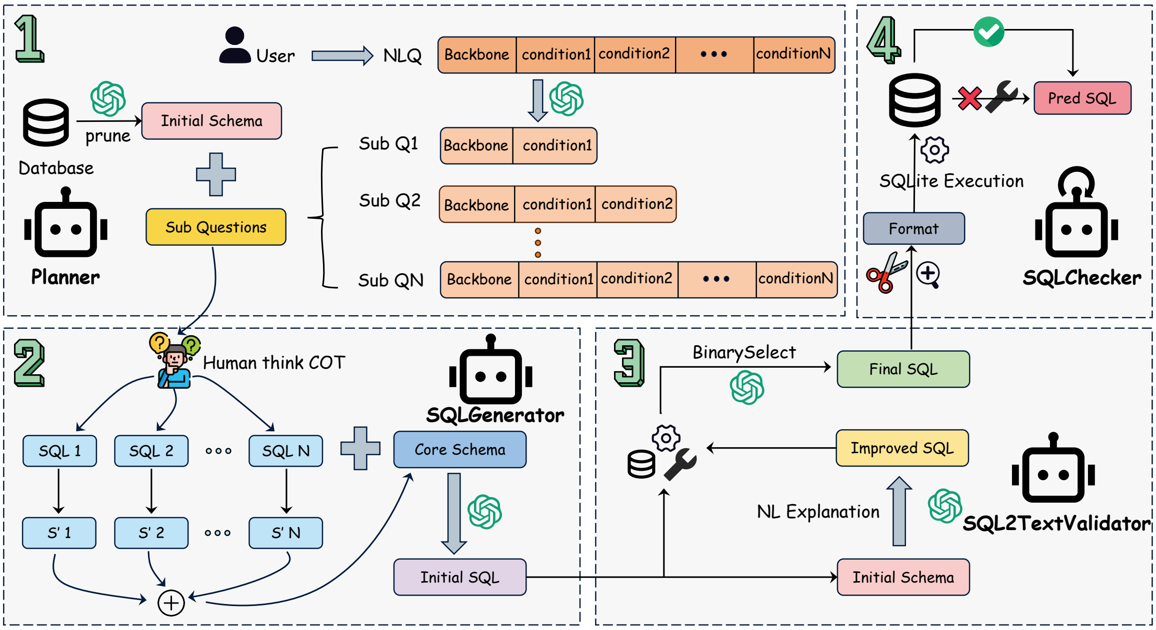


Figure 2: The workflow of GBV-SQL framework, including four agents: (1)*Planner* (2)*SQLGenerator* (3)*SQL2TextValidator* (4)*SQLChecker*.

Figure 1, when asked for a school whose "average score" column has a value of 499, a model might instead generate a query that performs an unnecessary aggregation, calculating the average of the "average score" column across a group of schools and checking if that new result is 499. This error, transforming a direct filtering condition ('WHERE') into a flawed aggregation condition ('HAVING'), highlights the inadequacy of syntax-only verification. While existing state-of-the-art methods employ techniques like query decomposition (Gao et al., 2024a; Xie et al., 2024) and multi-agent collaboration (Wang et al., 2025) to ensure that LLMs generate reasonable SQL, they often lack a dedicated mechanism to robustly validate the final query's logic against the NLQ's semantics. To bridge this semantic gap, we propose GBV-SQL, a multi-agent framework integrating Guided Generation with SQL2Text Back-translation Validation. The workflow is orchestrated by four specialized agents (Figure 2). The process starts with a *Planner* that prunes the database schema and decomposes the NLQ into simpler sub-problems. An *SQLGenerator* then constructs the query using a novel Human-like Chain-of-Thought process. Critically, to ensure semantic fidelity, a dedicated *SQL2TextValidator* performs back-translation, correcting the query to produce an improved version if a logical mismatch with the user's intent is found. Finally, an *SQLChecker* conducts final checks, initiating an iterative repair cycle to resolve any syntactic or execution errors until the query is valid. This entire process is designed to systematically target and minimize inaccuracies.

Our rigorous evaluation on the Spider benchmark (Yu et al., 2018) yields an equally critical finding: a substantial portion of our model's apparent failures stem not from our method, but from inherent errors in the benchmark's ground-truth data. We term these issues "Gold Errors" and propose a formal typology for their systematic classification. This discovery fundamentally challenges the reliability of current evaluation practices and highlights a systemic issue in the field.

Our main contributions are threefold:

- We propose GBV-SQL, a novel multi-agent framework that introduces a SQL2Text back-translation validation mechanism to explicitly bridge the semantic gap in Text2SQL, ensuring the generated query's logic is faithful to the user's original intent.
- We conduct a systematic analysis of errors in the widely-used Spider benchmark, introducing a novel typology for "Gold Errors", which are flaws inherent in the ground-truth data itself.
- By correcting pervasive "Gold Errors", we reveal our model's true performance: execution

accuracy reaches **96.5%** (dev) and **97.6%** (test) on Spider, while on BIRD, it improves from **63.23%** to **90.42%**. This demonstrates the critical impact of benchmark integrity on Text2SQL evaluation.

2 Related Work

2.1 ICL-based Methods for Text2SQL

In-context learning (ICL) with Large Language Models (LLMs) is the current state-of-the-art paradigm for Text2SQL. A dominant strategy for tackling complex queries is decomposition, where methods break a difficult question into simpler sub-problems. For example, DIN-SQL (Pourreza and Rafiei, 2023) first applied CoT prompting for decomposition. DAIL-SQL (Gao et al., 2024a) then explored advanced prompt designs in this framework. (Tai et al., 2023) evaluated multiple prompt strategies and introduced QDecomp to incrementally add schema details to sub-questions. To improve overall robustness, other research has focused on two primary directions. The first is enhancing the quality of the model's input via more accurate schema linking (Cao et al., 2024; Wang and Liu, 2025; Li et al., 2024b). The second is refining the model's output through post-hoc correction of generated SQL (Dong et al., 2023; Ren et al., 2024) or by selecting the optimal query from multiple candidates (Lee et al., 2025; Pourreza et al., 2024). While these diverse strategies have advanced the field, they predominantly operate in a unidirectional workflow. They lack an explicit mechanism to verify if the final, syntactically correct SQL is truly semantically aligned with the user's original intent. Our work directly targets this semantic gap with a novel validation step.

2.2 Multi-Agent Systems for Text2SQL

Recent work has adopted multi-agent systems to coordinate the multi-step reasoning process. MAC-SQL (Wang et al., 2025) and MAG-SQL (Xie et al., 2024) introduced this approach by employing several LLM-based agents to collaboratively manage decomposition, SQL generation, and refinement. These frameworks demonstrated that a division of labor can effectively handle complex queries. However, existing agents only collaborate to generate SQL; none perform an explicit semantic verification of the final query. Other agent-based systems, like SQLFixAgent (Cen et al., 2025), focus specifically on correcting errors from fine-tuned models, which is a different task context. Our GBV-SQL framework introduces a new agent archetype: the SQL2TextValidator. This agent acts as a skeptical verifier whose sole purpose is to perform SQL2Text back-translation. By challenging the semantic fidelity of the generated query from an inverse perspective, it provides a critical feedback loop absent in prior collaborative models.

2.3 Benchmark Quality and Gold Errors

The reliability of Text2SQL evaluation is intrinsically tied to the quality of benchmark datasets. The presence of flaws in the ground-truth labels, termed "Gold Errors", has been noted in the error analyses of prior work (Wang et al., 2025; Lee et al., 2025). Further research has delved into specific quality issues, such as analyzing label accuracy (Renggli et al., 2025), classifying question ambiguity (Dong et al., 2024), or automatically detecting incorrect SQL-NLQ mappings (Yang et al., 2025). These efforts highlight a growing awareness of data quality challenges. Our work builds upon this foundation by proposing a formal, structured typology for Gold Errors. Unlike previous post-hoc analyses, our classification system is more comprehensive, categorizing errors originating from the SQL, the NLQ, and the database itself. This systematic framework not only allows for a more precise and fair re-evaluation of GBV-SQL's true capabilities but also offers a valuable tool for the future curation and maintenance of Text2SQL benchmarks.

3 Methodology

3.1 Overview

In this section, we introduce GBV-SQL, a novel multi-agent framework for Text2SQL leveraging large language models. As shown in Figure 2, our framework comprises four agents: *Planner*, *SQLGenerator*, *SQL2TextValidator* and *SQLChecker*. The Planner initiates the process by invoking an LLM to prune the database schema to its most relevant elements and then decomposes the NLQ into sub-questions. Mimicking human thought processes, the SQLGenerator generates SQL for each sub-question before synthesizing them into the final query. The SQL2TextValidator ensures semantic integrity by prompting an LLM to translate the complete SQL back into natural language for comparison with the original NLQ, yielding a refined query. Finally, the SQLChecker conducts format and syntax checks, correcting any errors to guarantee an executable final SQL.

3.2 Planner

3.2.1 Schema Representation

Prior work (Bogin et al., 2019; Guo et al., 2019; Lei et al., 2020; Gao et al., 2024b) has shown that a well-organized representation of the database schema can improve the accuracy of SQL generation by LLMs. In this paper, we organize our schema description based on the approach in MAC-SQL, but with augmentations. We incorporate the data types of database columns and provide more intuitive foreign key descriptions. Furthermore, we are the first to include descriptions for primary keys within the schema representation. The specific prompt design is illustrated in Appendix C.2.

3.2.2 Schema Pruning

Overly long schemas can cause an LLM to focus on information irrelevant to the NLQ and incur high token costs. Therefore, we perform an initial pruning of the full database schema by prompting the LLM. We prompt the LLM to analyze the NLQ (Q) and the full database schema (D), retaining only the most relevant information.

$$S_{\text{initial}} = \text{Prune}(Q, D)\,, \tag{1}$$

where S_{initial} represents the pruned schema relevant to the question.

3.2.3 NLQ Decomposition

As demonstrated in prior surveys (Pourreza and Rafiei, 2023, 2024), decomposing a question into sub-questions is an effective strategy. We employ a decomposition strategy based on the divide-and-conquer paradigm, wherein question decomposition and SQL generation are separate processes. Specifically, we adopt the Targets-Conditions method proposed in MAG-SQL (Xie et al., 2024) to decompose the NLQ based on its distinct conditional clauses and query targets, as shown in Figure 2.

3.3 SQLGenerator

3.3.1 Human-like CoT

To emulate the human thought process for drafting SQL queries (Yuan et al., 2025), we have designed a novel CoT method, illustrated in Figure 10. This approach first analyzes the intent of the NLQ. It then identifies the relevant tables and columns based on the database schema and determines the specific information to be returned. Subsequently, it systematically analyzes how to construct essential SQL clauses, considering the specific logic for joins, the columns for grouping and ordering, and the potential need for deduplication with DISTINCT. Finally, it emphasizes adherence to proper SQL syntax throughout the generation process.

3.3.2 Sub-SQL Generation and Merging

Our framework employs a two-stage process to generate the SQL. First, for each sub-question decomposed by the Planner, we generate a corresponding sub-SQL. Subsequently, we merge the contextual information from these sub-SQLs to synthesize the final answer. This process begins by applying our Human-like CoT method to generate a sub-SQL for each sub-question (q_i) using the initially pruned schema, S_{initial}. A crucial step follows where we perform a "back-linking" operation on each generated sub-SQL (SQL_i). This operation is an LLM inference step, where the model is prompted to extract the precise set of table-column pairs (S_i) that were essential for the sub-query's creation.

$$SQL_i = \text{HumanCoT}(S_{\text{initial}}, q_i) \tag{2}$$

$$S_i = \text{BackLink}(SQL_i) \tag{3}$$

These extracted schema subsets are then merged to create a more refined and highly relevant schema, S_{core}. This new schema is more accurate than S_{initial} because it is derived directly from the SQL logic required to answer each sub-question. The merging operation is defined as the union of all table-column pairs from the individual schema subsets:

$$S_{\text{core}} = \bigoplus_{i=1}^{n} S_i \tag{4}$$

where the operator is defined for any table t_k in the schema as:

$$\left(\bigoplus_{i=1}^{n} S_i\right)(t_k) = \bigcup_{i=1}^{n} S_i(t_k) \tag{5}$$

Finally, this consolidated schema S_{core}, along with the complete set of sub-question and sub-SQL pairs, provides a rich and focused context for the LLM to generate the final comprehensive query, referred to as Initial SQL in Figure 2.

3.4 SQL2TextValidator

As one of the core modules in GBV-SQL, the SQL2TextValidator performs semantic validation by prompting the LLM to translate the initial SQL query from the SQLGenerator into a detailed natural language explanation. This explanation describes the query's overall function and the specific mechanics of its components, with a particular focus on the query's execution process. The LLM is then tasked with comparing this explanation against the original NLQ to identify any semantic discrepancies. If the logic is consistent, the query is accepted; otherwise, the LLM is prompted to correct the SQL to align with the user's intent, yielding a validated improved SQL. To maximize the validation's efficacy, we incorporate a binary selector inspired by RSL-SQL (Cao et al., 2024). This selector uses the LLM to evaluate the semantic consistency of both the initial and the validated SQL versions against the original NLQ, taking their execution results into account to select the Final SQL. This back-translation validation is a cornerstone of our framework, playing a critical role in ensuring the final query's logic is faithful to the user's original intent. As our experimental results will demonstrate, this validation step is pivotal for correcting semantic deviations and improving overall accuracy.

3.5 SQLChecker

Due to the hallucinatory nature of large models, the generated SQL often contain formatting errors (Shen et al., 2025). Therefore, this module introduces an SQL format trimmer. This component optimizes the query according to two key principles: a minimization principle, which eliminates superfluous fields and redundant operations, and a minimal usability principle, which avoids intrusive formatting (e.g., rounding decimals) that could alter the result's meaning unless explicitly required. Subsequently, we employ a multi-round iterative checking process to analyze the query's executability. If the SQL execution yields an empty result, a value of 0, or contains NULL values, the query is repaired. This repair cycle continues until the query executes successfully or the maximum number of iterations is exceeded. For the repair process, we retrieve the top-k most relevant values from the database based on the SQL and combine them with the partial execution results as context for prompting the LLM to make corrections. Algorithm 1 details this process.

Algorithm 1 The SQLChecker Algorithm

Input: Question q, database db, schema S', initial SQL sql_{pre}
Output: final SQL sql

$sql \leftarrow$ `ReduceFormat`$(q,\ sql_{\text{pre}},\ S')$
$count \leftarrow 0$
while $count < \text{maxTryTime}$ **do**
    $(pass,\ err,\ res) \leftarrow$ `execTool`$(sql,\ db)$
    if $pass$ **then**
        break
    else
        $vals \leftarrow$ `valueRetrieve`$(q,\ sql,\ S',\ db)$
        $sql \leftarrow$ `Refiner`$(q,\ sql,\ S',\ err,\ res,\ vals)$
    end if
    $count \leftarrow count + 1$
end while
return sql

4 Experiments

4.1 Experimental Setup

4.1.1 Datasets

We evaluate GBV-SQL on two widely-used Text2SQL benchmarks: Spider (Yu et al., 2018), to assess cross-domain generalization across its over 200 databases (e.g., `flight_2`), and the more challenging BIRD (Li et al., 2023b), to test performance against real-world complexity involving noisy data and external knowledge. While Spider 2.0 (Lei et al., 2024) is an important industry-oriented evolution, its complexity in long-context understanding and multi-tool usage currently hinders a clear evaluation of our semantic validation mechanism. We plan to adapt GBV-SQL for such enterprise scenarios in future work.

4.1.2 Baseline

In our experiments, we compare our proposed method GBV-SQL against a wide range of ICL-based methods, including DIN-SQL (Pourreza and Rafiei, 2023), DAIL-SQL (Gao et al., 2024a), MAC-SQL (Wang et al., 2025), MAG-SQL (Xie et al., 2024), SuperSQL (Li et al., 2024a), and CogSQL (Yuan et al., 2025).

4.1.3 Evaluation Metrics

We evaluate our framework using two metrics: execution accuracy and valid efficiency score. ***Execution accuracy (EX)*** is defined as the proportion of predicted SQL queries whose execution results exactly match those of the ground-truth SQL. The ***valid efficiency score (VES)***, introduced by the BIRD dataset, jointly considers the correctness and execution efficiency of generated SQL. The

Methods	EX				VES
	Simple	Moderate	Challenging	Total	
GPT-4 (zero-shot)	–	–	–	46.35	51.75
Deepseek-v3 (zero-shot)	49.84	34.70	25.52	42.96	53.25
DIN-SQL + GPT-4	–	–	–	50.72	58.79
MAG-SQL + GPT-4	–	–	–	61.08	–
DAIL-SQL + GPT-4	63.02	45.59	43.05	54.76	–
CogSQL + GPT-4	–	–	–	59.58	64.30
CogSQL + Deepseek-v2	64.11	46.67	39.58	56.52	–
MAC-SQL + GPT-4	65.73	52.96	40.28	59.39	66.24
MAC-SQL + Deepseek-v3	62.92	50.75	43.75	57.43	68.40
GBV-SQL + Deepseek-v3 (ours)	**69.51**	**54.62**	**50.69**	**63.23** (↑5.8%)	**69.87**

Table 1: The EX and VES results on BIRD's dev set. "–" indicates that the value was not reported by the corresponding authors.

efficiency score (related to execution time) is calculated only if the predicted and real SQL results match; otherwise, the score is 0.

4.1.4 Implementation Details

Our experiments primarily use Deepseek-v3 as the backbone large language model, accessed through the Deepseek API. For supplementary validation, we also employ GPT-4o via the OpenAI API. The temperature for both models is set to 0. In our SQLChecker module, the maximum number of repair iterations is set to 3. Our multi-agent architecture draws inspiration from MAC-SQL. Our method is capable of handling moderately complex queries, covering scenarios that involve both single and multiple tables.

Methods	Dev	Test
GPT-4 (zero-shot)	74.6	–
DIN-SQL + GPT-4	82.8	85.3
DAIL-SQL + GPT-4	84.4	86.6
CogSQL + GPT4	85.4	86.4
SuperSQL + GPT4	87.0	–
MAG-SQL + GPT-4	85.3	85.6
MAC-SQL + GPT-4	86.8	82.8
MAC-SQL + Deepseek-v3	80.1	77.6
GBV-SQL + Deepseek-v3 (ours)	79.6	82.8(↑5.2%)
GBV-SQL + GPT-4o (ours)	79.7	83.9
GBV-SQL + No Gold Errors	**96.5**	**97.6**

Table 2: The EX results on the Spider dev and test sets. Improvements relative to MAC-SQL+Deepseek-v3 are shown in parentheses. The "No Gold Errors" designation indicates evaluation on a cleaned subset of the benchmark, from which entries with quality issues have been removed.

4.2 Main Result

4.2.1 BIRD Result

The results in Table 1 show that our method significantly enhances the performance of the Deepseek baseline. Using the Deepseek-v3 model, GBV-SQL achieves an EX of 63.23% on the BIRD dev dataset, which is a 5.8% improvement over MAC-SQL that also uses the Deepseek model, and even surpasses numerous methods that use GPT-4. This highlights the effectiveness of GBV-SQL in Text2SQL tasks.

4.2.2 Spider Result

Table 2 shows the performance of our framework and baseline methods on the Spider dataset. GBV-SQL achieves execution accuracies of 79.6% and 82.8% on the Spider dev and test sets, respectively. On the test set, our method outperforms the MAC-SQL baseline by a notable 5.2% when using the same Deepseek-v3 model, achieving performance comparable to methods that rely on GPT-4. However, the performance on the dev set is unexpectedly lower than anticipated. This discrepancy motivates a deeper investigation into the nature of the execution failures, which reveals that the issue stems not from our model's capabilities but from what we identify as "Gold Errors": pervasive quality issues within the benchmark's ground-truth data itself, as detailed in the following section.

4.3 Error Analysis

A rigorous manual review of all items that failed execution on the Spider benchmark reveals the overwhelming majority of failures are not caused by our model but by inherent quality issues in the dataset. Consequently, we develop a novel framework for classifying these gold-standard errors.

4.3.1 Gold Error Identification

To ensure an objective and reproducible analysis, we establish a rigorous protocol to audit the quality of each data item. The process is conducted by three SQL-proficient graduate students. Initially, two students independently inspect and classify potential quality issues in every data item according to our proposed typology (Figure 4), achieving a substantial inter-annotator agreement (Cohen's Kappa = 0.86). A third student then adjudicates all disagreements to make the final classification, ensuring the reliability of our findings.

4.3.2 A New Typology of Gold Errors

While prior research has acknowledged the existence of data quality problems, a clear and systematic classification has been absent. We address this gap by introducing a new standard that categorizes Gold Errors into three distinct types based on a comprehensive analysis of recurring issues. For concrete examples illustrating each category and subcategory, please refer to the case studies in Appendix B.

- **Type A (SQL-Side Errors)**: Covers all errors originating from the gold SQL itself, assuming the NLQ is valid. This includes semantic errors where the query fails to correctly or optimally represent the user's intent, as well as syntactic errors that prevent the query from executing.
- **Type B (NLQ-Side Errors)**: Includes all cases where the NLQ itself is the source of the problem. This covers questions that are ambiguous, underspecified, logically flawed, or unanswerable with the given database.
- **Type C (Database Errors)**: Signifies errors found within the database itself, such as flawed schema design or inconsistent and "dirty" data values.

As detailed in Appendix A.1, each primary category is further partitioned into a granular hierarchy of subcategories.

4.3.3 Analysis of Gold Errors in Spider

Our analysis of the Spider dev set uncovers a significant number of "Gold Errors" (i.e., errors in the ground-truth labels), with 183 instances found in samples our model failed (EX=0) and 62 in those it passed (EX=1). Figure 5 provides a quantitative breakdown of the various Gold Error categories within these failed samples. A prominent example is the data contamination within the `flight_2` database, which is illustrated in Figure 8; here, extraneous spaces in TEXT attributes cause many official Gold SQL queries to fail, whereas our method correctly handles such "dirty data". After manually correcting for these dataset flaws, we find that only 37 were genuine model failures. Consequently, re-evaluating on a "clean" subset improves our model's execution accuracy to 96.5% on the dev set and 97.6% on the test set (Table 2, "No Gold Errors" row). This finding highlights how benchmark quality issues can mask true model performance. Due to space limitations, further details of our analysis on the Spider test set are available in Figure 6.

Metric	Gold Error Subset	Full Dev Set
Number of Samples	577	1534
Original EX	6.93	63.23
Corrected EX	**79.20**	**90.42**

Table 3: Re-evaluation of GBV-SQL on the BIRD dev set after correcting Gold Errors. The "Gold Error Subset" refers to samples identified with ground-truth flaws.

4.3.4 Analysis of Gold Errors in BIRD

Figure 3 illustrates the standard evaluation process for a question-SQL pair in existing benchmarks such as Spider and BIRD. In this process, the model-generated SQL (predicted SQL) is compared against a single ground-truth query (gold SQL). A binary score of 1 or 0 is awarded based on whether their execution results on the database match. The final Execution Accuracy (EX) metric is then calculated by aggregating these scores across all samples. This process is highly sensitive to the quality of the gold standard; any "Gold Error" in the benchmark can lead to a correct model prediction being incorrectly penalized, thus distorting the performance metrics. Motivated by our findings on Spider, we conducted a comprehensive quality review of the entire BIRD dev set.

Unlike our approach with Spider, where problematic samples were excluded for re-evaluation, our analysis of BIRD involved a more intensive effort: we manually repaired the identified "Gold Errors" within the ground-truth data. Our review revealed that 37.6% of the samples in the BIRD dev set contained "Gold Errors", corroborating recent studies (Shen et al., 2025; Yang et al., 2025), span-

Model	Simple	Mod.	Chall.	Total
GBV-SQL + Deepseek-v3	69.51	54.62	50.69	**63.23**
w/o Planner	68.76	53.98	45.83	62.13 (↓)
w/o SQLGenerator	67.78	53.76	48.61	61.73 (↓)
w/o Human-like CoT	66.70	52.90	51.39	61.08 (↓)
w/o SQL2TextValidator	68.54	52.69	47.92	61.80 (↓)
w/o SQLChecker	66.49	49.89	47.22	59.65 (↓)

Table 4: Ablation of GBV-SQL on the BIRD's dev set. Mod. and Chall. stand for Moderate and Challenging, respectively.

ning a range of issues from flawed SQL logic to ambiguous natural language questions. After correcting these flaws, we re-evaluated our model's predictions on this repaired subset. The results showed that on the BIRD dev set, GBV-SQL's true execution accuracy increased to 90.42%. Table 3 provides a detailed breakdown of the evaluation results before and after the correction. This significant improvement not only highlights the pervasive quality issues within the BIRD benchmark but also reveals the true, higher performance of GBV-SQL when evaluated against a reliable ground truth. A detailed breakdown of Gold Error distribution and case studies for the BIRD dataset is provided in Appendix A.2.2. It is worth noting that even after this meticulous repair effort, the overwhelming majority of the remaining 20.8% of inconsistencies were found to be caused by more subtle or complex Gold Errors, while only a small fraction constituted actual model failures.

4.4 Ablation Study

Table 4 presents our ablation study on the BIRD dev set. "w/o Planner/SQLGenerator" indicates replacement with MAC-SQL's modules, while other variants remove the specified component. "w/o Human-like CoT" indicates replacing the chain-of-thought prompting with a standard zero-shot prompt. The results confirm that each component of the GBV-SQL framework is integral, as removing any one of them degrades overall performance. Notably, ablating the SQLChecker, which handles final formatting and executability analysis, causes the most significant performance drop. This underscores the profound impact of query executability and proper formatting on the current execution based (EX) evaluation paradigm.

4.5 Discussion

The standard Text2SQL evaluation compares the execution results of predicted queries against gold queries, a process highly susceptible to dataset quality. Our comprehensive analyses of both the Spider and BIRD datasets confirm that this is a widespread issue. For instance, an ambiguous natural language question can render a single gold SQL query insufficient for reliable judgment. Likewise, a correct question paired with a flawed gold SQL will still yield an inaccurate EX score. More critically, database-level errors can cause substantial deviations in evaluation. Therefore, we strongly urge the community to place greater emphasis on the integrity of Text2SQL benchmarks, adopting a more rigorous approach to their creation and maintenance. Benchmarks should not be treated as static, infallible standards, but as dynamic entities that require continuous validation, correction, and versioning.

5 Conclusion

This paper investigates the problem of semantic inconsistency in Text2SQL, proposing GBV-SQL, a novel multi-agent framework designed to ensure a query's logic is faithful to user intent. The cornerstone of our approach is a SQL2Text back-translation validation mechanism, which effectively bridges the semantic gap between the user's question and the generated SQL. This method demonstrates significant effectiveness, achieving a 5.8% absolute accuracy improvement on the BIRD benchmark. In our comprehensive evaluation, we also uncovered pervasive flaws in the benchmark datasets, which we classify using a novel "Gold Error" typology. After correcting for these benchmark issues, the true performance of GBV-SQL was revealed, with execution accuracy reaching 96.5% and 97.6% on the Spider development and test sets, respectively, and 90.42% on a corrected version of the BIRD development set. Our work, therefore, offers a robust framework for enhancing semantic fidelity and highlights the critical need for benchmark curation, suggesting future directions in automated validation for complex reasoning tasks.

Limitations

Despite the improvements achieved by GBV-SQL, several limitations persist. First, our evaluation relies on existing benchmarks (Spider, BIRD) where questions are paired with definitive SQL queries. In real-world enterprise scenarios, user intent is often ambiguous or underspecified, requiring multiturn clarificationa capability our current framework lacks. Second, while we address semantic errors, our method essentially relies on the capabilities of the underlying LLM; if the model's internal world knowledge is insufficient (e.g., domain-specific acronyms), the back-translation validator may yield false positives. Finally, we have not yet adapted GBV-SQL to handle long-context retrieval scenarios as introduced in Spider 2.0, which remains a direction for future work.

References

Ben Bogin, Jonathan Berant, and Matt Gardner. 2019. Representing schema structure with graph neural networks for text-to-sql parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4560–4565.

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

Jipeng Cen, Jiaxin Liu, Zhixu Li, and Jingjing Wang. 2025. Sqlfixagent: Towards semantic-accurate text-to-sql parsing via consistency-enhanced multi-agent collaboration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 49–57.

Mingwen Dong, Nischal Ashok Kumar, Yiqun Hu, Anuj Chauhan, Chung-Wei Hang, Shuaichen Chang, Lin Pan, Wuwei Lan, Henghui Zhu, Jiarong Jiang, and 1 others. 2024. Practiq: A practical conversational text-to-sql dataset with ambiguous and unanswerable queries. *Preprint*, arxiv:2410.11076.

Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Jinshu Lin, Dongfang Lou, and 1 others. 2023. C3: Zero-shot text-to-sql with chatgpt. *Preprint*, arxiv:2307.07306.

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. *Proceedings of the VLDB Endowment*, 17(5):1132–1145.

Yingqi Gao, Yifu Liu, Xiaoxia Li, Xiaorong Shi, Yin Zhu, Yiming Wang, Shiqi Li, Wei Li, Yuntao Hong, Zhiling Luo, and 1 others. 2024b. Xiyan-sql: A multi-generator ensemble framework for text-to-sql. *Preprint*, arxiv:2411.08599.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-sql in cross-domain database with intermediate representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4524–4535.

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*, pages 337–353.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, and 1 others. 2024. Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows. *Preprint*, arxiv:2411.07763.

Wenqiang Lei, Weixin Wang, Zhixin Ma, Tian Gan, Wei Lu, Min-Yen Kan, and Tat-Seng Chua. 2020. Re-examining the role of schema linking in text-to-sql. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6943–6954.

Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024a. The dawn of natural language to sql: Are we fully ready? *Proceedings of the VLDB Endowment*, 17(11):3318–3331.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023a. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 13067–13075.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, and 1 others. 2023b. 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*, 36:42330–42357.

Zhishuai Li, Xiang Wang, Jingjing Zhao, Sun Yang, Guoqing Du, Xiaoru Hu, Bin Zhang, Yuxiao Ye, Ziyue Li, Rui Zhao, and 1 others. 2024b. Pet-sql: A prompt-enhanced two-round refinement of text-to-sql with cross-consistency. *Preprint*, arxiv:2403.09732.

Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Nan Tang, and Yuyu Luo. 2024. A survey of nl2sql with large language models: Where are we, and where are we going? *arXiv preprint arXiv:2408.05109*.

Minh Thuan Nguyen, Khanh-Tung Tran, Nhu Van Nguyen, and Xuan-Son Vu. 2023. Vigptqa-state-of-the-art llms for vietnamese question answering: system overview, core models training, and evaluations. In *Proceedings of the 2023 conference on empirical methods in natural language processing: industry track*, pages 754–764.

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. *Preprint*, arxiv:2410.01943.

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

Mohammadreza Pourreza and Davood Rafiei. 2024. Dts-sql: Decomposed text-to-sql with small large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 8212–8220.

Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, Binhua Li, Ruiying Geng, Rongyu Cao, Jian Sun, Luo Si, and 1 others. 2022. A survey on text-to-sql parsing: Concepts, methods, and future directions. *arXiv preprint arXiv:2208.13629*.

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 *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 15–28. IEEE.

Cedric Renggli, Ihab F Ilyas, and Theodoros Rekatsinas. 2025. Fundamental challenges in evaluating text2sql solutions and detecting their limitations. *Preprint*, arxiv:2501.18197.

Jiawei Shen, Chengcheng Wan, Ruoyi Qiao, Jiazhen Zou, Hang Xu, Yuchen Shao, Yueling Zhang, Weikai Miao, and Geguang Pu. 2025. A study of in-context-learning-based text-to-sql errors. *Preprint*, arxiv:2501.09310.

Chang-Yu Tai, Ziru Chen, Tianshu Zhang, Xiang Deng, and Huan Sun. 2023. Exploring chain of thought style prompting for text-to-sql. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5376–5393.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and 1 others. 2025. Mac-sql: A multi-agent collaborative framework for text-to-sql. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 540–557.

Yihan Wang and Peiyu Liu. 2025. Linkalign: Scalable schema linking for real-world large-scale multi-database text-to-sql. *Preprint*, arxiv:2503.18596.

Wenxuan Xie, Gaochen Wu, and Bowen Zhou. 2024. Mag-sql: Multi-agent generative approach with soft schema linking and iterative sub-sql refinement for text-to-sql. *Preprint*, arXiv:2408.07930.

Yicun Yang, Zhaoguo Wang, Yu Xia, Zhuoran Wei, Haoran Ding, Ruzica Piskac, Haibo Chen, and Jinyang Li. 2025. Automated validation and fixing of text-to-sql translation with execution consistency. *Proc. ACM Manag. Data*, 3(3):134.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, and 1 others. 2018. 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*, pages 3911–3921.

Hongwei Yuan, Xiu Tang, Ke Chen, Lidan Shou, Gang Chen, and Huan Li. 2025. Cogsql: A cognitive framework for enhancing large language models in text-to-sql translation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 25778–25786.

John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1050–1055.

Xinyu Zhu, Junjie Wang, Lin Zhang, Yuxiang Zhang, Yongfeng Huang, Ruyi Gan, Jiaxing Zhang, and Yujiu Yang. 2023. Solving math word problems via cooperative reasoning induced language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4471–4485.

A Detailed Analysis of Gold Errors

To better understand the context of our error analysis, we first revisit the standard evaluation pipeline. As illustrated in Figure 3, the traditional evaluation process relies heavily on the exact match of execution results between the predicted SQL and the Gold SQL. This dependency makes the metric highly sensitive to any flaws in the ground truth (Gold Errors).

A.1 Expanded Gold Error Typology

We categorize benchmark flaws into three primary types (Type A: SQL-side, Type B: NLQ-side, Type C: Database-side). Figure 4 provides a hierarchical view of this classification system.

A.2 Distribution Analysis on Datasets

We analyzed the distribution of these error types on both Spider and BIRD datasets.

A.2.1 Spider Dataset Analysis

Figure 5 illustrates the breakdown of Gold Errors for samples in the Spider development set where our model initially failed (EX=0). The high prevalence of Type A and Type C errors explains the discrepancy between the reported score and the model's actual capability.

For the Spider test set, a similar distribution was observed, as shown in Figure 6.

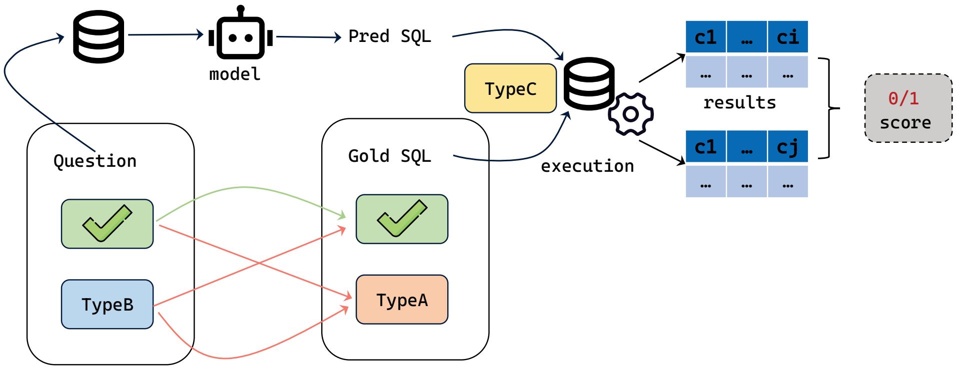


Figure 3: The standard Text2SQL evaluation pipeline. Any error in the "Gold SQL" or "Database" directly propagates to the final accuracy metric.

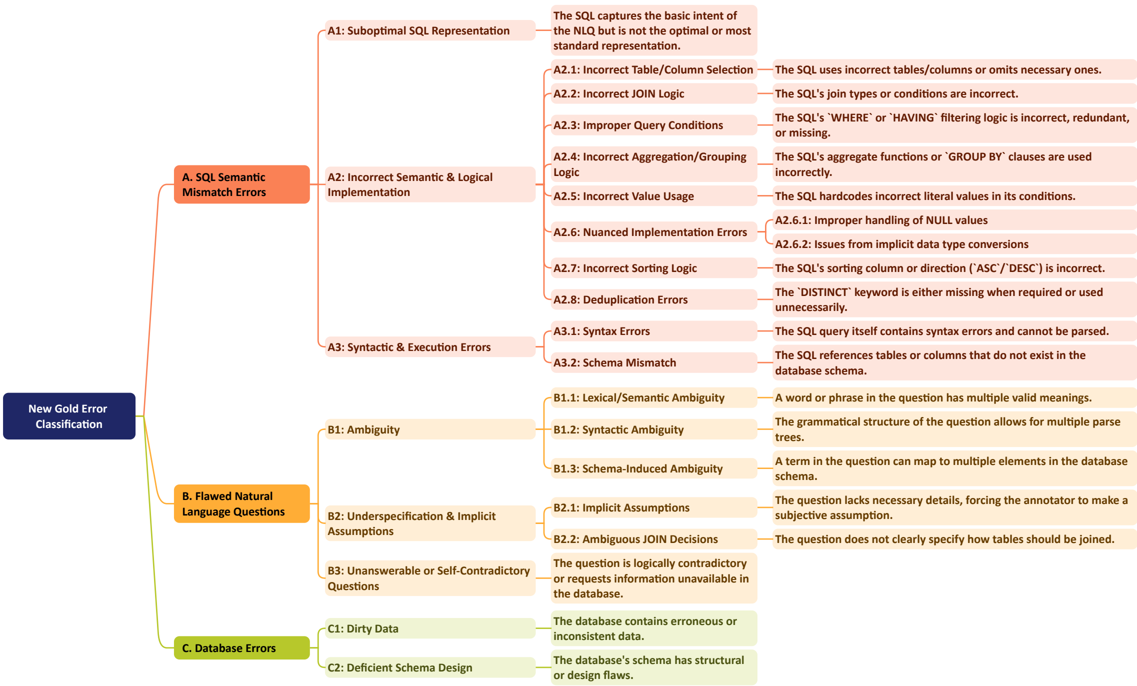


Figure 4: The new Gold Error Classification proposed in this work.

A.2.2 BIRD Dataset Analysis

Figure 7 shows the overall distribution of error types found across the entire BIRD development set.

B Case Studies of Gold Errors

We present detailed case studies to exemplify the error types defined in our typology.

Case 1: Suboptimal SQL Representation (A1)

ID Spider_dev(idx:484)

NL What are the names and ranks of the three youngest winners across all matches?

Gold SQL

```
SELECT DISTINCT winner_name , winner_rank
FROM matches ORDER BY winner_age LIMIT 3
```

Real SQL

```
WITH winner_min_age AS (
  SELECT winner_id, MIN(winner_age) AS min_age
   FROM matches
   GROUP BY winner_id
)
SELECT m.winner_name, m.winner_rank
FROM matches AS m
INNER JOIN winner_min_age AS w
   ON m.winner_id = w.winner_id
   AND m.winner_age = w.min_age
ORDER BY w.min_age ASC
LIMIT 3;
```

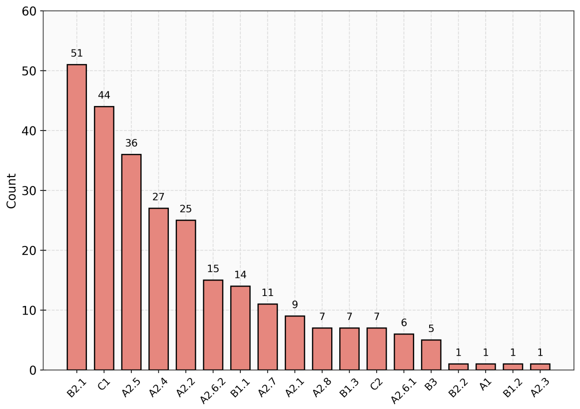


Figure 5: The distribution of Gold Error types in the Spider's dev set with EX=0.

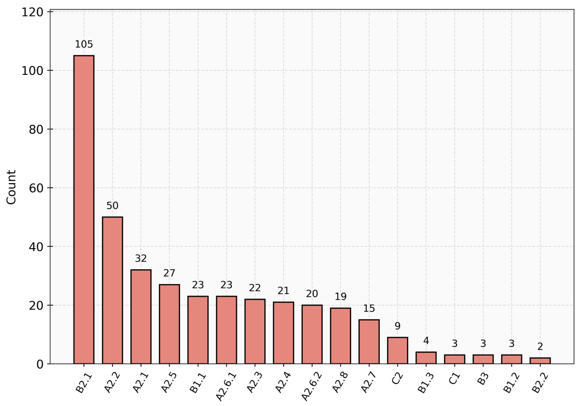


Figure 6: The distribution of Gold Error types in the Spider's test set with EX=0.

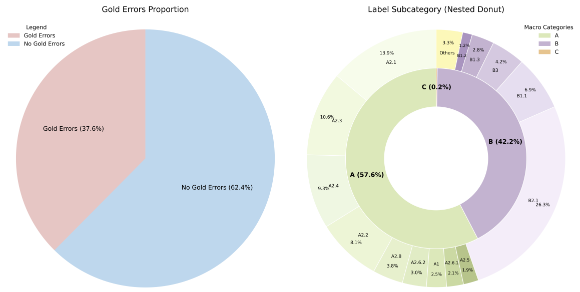


Figure 7: The distribution of Gold Error types in the BIRD's dev set.

Explanation Overlooked the fact that a contestant might have multiple wins, which ultimately resulted in always returning the same person.

Case 2: Improper Query Conditions (A2.3)

ID Spider_dev(idx:241)

NL Find all airlines that have at least 10 flights.

Gold SQL
```
SELECT T1.Airline
FROM AIRLINES AS T1
JOIN FLIGHTS AS T2 ON T1.uid = T2.Airline
GROUP BY T1.Airline
HAVING count( * ) > 10;
```

Real SQL
```
SELECT T1.Airline
FROM AIRLINES AS T1
JOIN FLIGHTS AS T2 ON T1.uid = T2.Airline
GROUP BY T1.Airline
HAVING count( * ) >= 10;
```

Explanation The filtering condition is incorrect. It should be `>= 10`.

Case 3: Incorrect Aggregation/Grouping Logic (A2.4)

ID Spider_test(idx:1601)

NL Find the make that has more than one team.

Gold SQL
```
SELECT make
FROM team
GROUP BY team
HAVING count( * ) > 1;
```

Real SQL
```
SELECT make
FROM team
GROUP BY team.make
HAVING count( * ) > 1;
```

Explanation Incorrect grouping conditions.

Case 4: Incorrect Value Usage (A2.5)

ID BIRD_dev(idx:93)

NL How many male customers who are living in North Bohemia have average salary greater than 8000? (Male means that gender = 'M'; A3 refers to region; A11 pertains to average salary.)

Gold SQL
```
SELECT COUNT(T1.client_id)
FROM client AS T1
INNER JOIN district AS T2
  ON T1.district_id = T2.district_id
WHERE T1.gender = 'M'
  AND T2.A3 = 'North Bohemia'
  AND T2.A11 > 8000;
```

Real SQL
```
SELECT COUNT(T1.client_id)
FROM client AS T1
INNER JOIN district AS T2
  ON T1.district_id = T2.district_id
WHERE T1.gender = 'M'
  AND T2.A3 = 'north Bohemia'
  AND T2.A11 > 8000;
```

Explanation The database actually stores north Bohemia (lowercase 'n').

Case 5: Improper handling of NULL values (A2.6.1)

ID Spider_test(idx:1841)

NL Find the name of theaters that has at least one movie playing.

Gold SQL
```
SELECT name
FROM movietheaters
GROUP BY name
HAVING count( * ) >= 1;
```

Real SQL
```
SELECT Name
FROM MovieTheaters
WHERE Movie IS NOT NULL;
```

Explanation Gold SQL ignores the possibility of NULL values in the data.

Case 6: Lexical/Semantic Ambiguity (B1.1)

ID BIRD_dev(idx:3)

NL What is the unabbreviated mailing address of the school with the highest FRPM count for K-12 students?

Gold SQL
```
SELECT T2.MailStreet FROM frpm AS T1
INNER JOIN schools AS T2 ON
T1.CDSCode = T2.CDSCode
ORDER BY T1.`FRPM Count (K-12)`
DESC LIMIT 1
```

New NL What is the unabbreviated mailing street address of the school with the highest FRPM count for K-12 students?

Explanation The phrase mailing address in the question is ambiguous and cannot accurately refer to the MailStreet column.

Case 7: Implicit Assumptions (B2.1)

ID BIRD_dev(idx:54)

NL Please specify all of the schools and their related mailing zip codes that are under Avetik Atoian's administration.

Gold SQL
```
SELECT School, MailZip
FROM schools
WHERE AdmFName1 = 'Avetik' AND
AdmLName1 = 'Atoian';
```

Pred SQL
```
SELECT School, MailZip
FROM schools
WHERE (AdmFName1 = 'Avetik' AND AdmLName1 =
'Atoian') OR (AdmFName2 = 'Avetik' AND
AdmLName2 = 'Atoian') OR
(AdmFName3 = 'Avetik' AND
AdmLName3 = 'Atoian');
```

Explanation Each school has up to three administrators, but the Gold SQL only considers the first one by default.

Case 8: Dirty Data (C1)

ID Spider_dev(idx:185)(flight_2)

NL List the airport code and name in the city of Anthony.

Gold SQL
```
SELECT AirportCode,
AirportName
FROM AIRPORTS
WHERE city = "Anthony";
```

Pred SQL
```
SELECT AirportCode,
AirportName
FROM airports
WHERE City = 'Anthony ';
```

Explanation Dirty data in the database causes the Gold SQL to return an empty result because the database stores 'Anthony ' (with a trailing space).

As shown in Figure 8, data inconsistency (extraneous spaces in the database values) causes valid queries to fail execution checks. The database stores 'Anthony ' instead of 'Anthony', causing standard equality checks to fail.

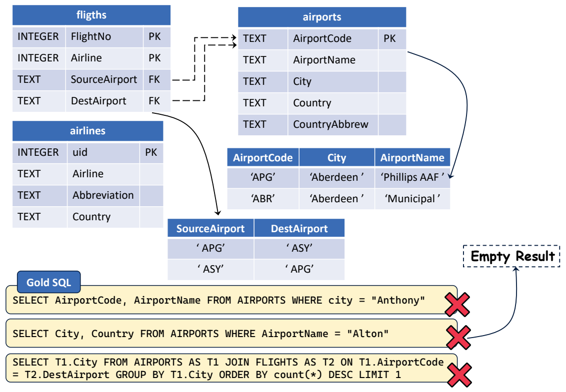


Figure 8: An example of (C1:Dirty Data) from database `flight_2` in Spider's dev set.

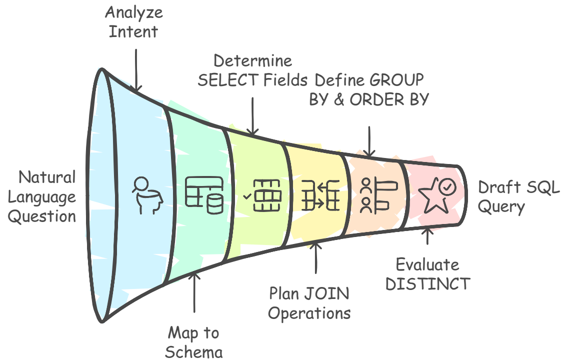


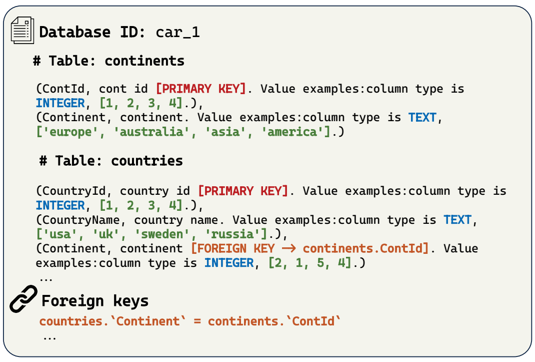
Figure 10: Human-like Chain-of-Thought Workflow for Drafting SQL Queries.

C Implementation Details and Prompts

This section details the specific prompt designs used in the GBV-SQL framework.

C.1 Schema Representation

The template used for Schema Representation in the Planner agent is shown in Figure 9.



```
Database ID: car_1
# Table: continents

(ContId, cont id [PRIMARY KEY]. Value examples:column type is
INTEGER, [1, 2, 3, 4].),
(Continent, continent. Value examples:column type is TEXT,
['europe', 'australia', 'asia', 'america'].)

# Table: countries

(CountryId, country id [PRIMARY KEY]. Value examples:column type is
INTEGER, [1, 2, 3, 4].),
(CountryName, country name. Value examples:column type is TEXT,
['usa', 'uk', 'sweden', 'russia'].),
(Continent, continent [FOREIGN KEY -> continents.ContId]. Value
examples:column type is INTEGER, [2, 1, 5, 4].)
...
Foreign keys
countries.`Continent` = continents.`ContId`
...
```

Figure 9: The prompt template for Schema Representation.

C.2 Human-like Chain-of-Thought

Figure 10 illustrates the workflow of our Human-like CoT reasoning process.

Prompt Content:

> As a SQL expert, your task is to complete the following steps according to the SQLite specification, based on the given [Question], [Evidence], and [Database Schema]:
>
> **Step 1:** Based solely on the [Question] and [Evidence], analyze the specific intent of the [Question] to determine what information needs to be retrieved.
>
> **Step 2:** Based on the results from Step 1 and in conjunction with the [Database Schema], analyze and identify the database tables and columns required to generate the corresponding SQL query.
>
> **Step 3:** Determine the exact columns to return based on the question's intent. Do not include extraneous columns. For example, if the question is "Who is the youngest singer?", you should only return the singer's name or ID, not their age.
>
> **Step 4:** Analyze if a JOIN operation is required. If so, specify the type of join (e.g., INNER JOIN, LEFT JOIN) and the columns to be used for the join condition.
>
> **Step 5:** Analyze if grouping is needed. If so, specify the columns for the GROUP BY clause.
>
> **Step 6:** Analyze if ordering is needed. If so, specify the columns for the ORDER BY clause. Do not add an ORDER BY clause unless it is explicitly required by the question.
>
> **Step 7:** Analyze whether the `DISTINCT` keyword is needed for deduplication. Generally, if the [Question] asks for specific attributes of each entity record, the results should have a one-to-one correspondence with the entities, and `DISTINCT` is unnecessary. However, when joining multiple tables, carefully consider whether join amplification might produce duplicate records, which would require using `DISTINCT`. This decision must be based on the specific problem.
>
> **Step 8:** Based on the information gathered above, generate the final SQL query using SQLite syntax.

C.3 SQL Formatting

The *SQLChecker* agent utilizes the prompt shown in Figure 11 for SQL formatting and optimization.

You are an SQL optimization assistant. Your task is, based on the user's Question and Evidence and the provided SQL statement and related Database Schema, to complete the following tasks:

Task1: Strictly follow the minimization principle: return only the fields requested in the question; do not perform any derived inference, even if extra fields could enrich the information.

Task2: Follow the minimal-usable principle: operations not explicitly mentioned in the question must not appear (e.g., avoid string concatenation; you may directly return two separate fields).

Task3: Ensure the return format and the fields exactly match what the Question requests, including the number and order of fields. In SQL, return the columns in the same order as they appear in the Question.

Task4: Finally, output an executable and semantically correct SQL code block, ```sql```, and do not include any explanatory comments inside the code block.

- - - - - - - - - - - - - - -

case 1: What is the age and student id for the student called `Mark`? Indicate the student's class id.

sql1: select T1.age, T1.student_id, T1.class_id from student as T1 where T1.name = 'Mark'

sql2: select T1.age, T1.class_id, T1.student_id from student as T1 where T1.name = 'Mark'

Reason: sql2 is incorrect because the field order should be age, class_id, student_id, while the order in sql2 is wrong.

- - - - - - - - - - - - - - -

case 2: Which student has the highest average score?

sql1: SELECT s.student_id FROM students AS T1 JOIN scores AS T2 ON T1.student_id = T2.student_id GROUP BY T1.student_id ORDER BY AVG(T2.score) DESC LIMIT 1;

sql2: SELECT s.student_id, AVG(T2.score) FROM students AS T1 JOIN scores AS T2 ON T1.student_id = T2.student_id GROUP BY T1.student_id ORDER BY AVG(T2.score) DESC LIMIT 1;

Reason: sql1 is correct.Because he original question only asks which student has the highest average score, and does not ask for the corresponding score. Therefore, fields not requested by the question must not be returned.

- - - - - - - - - - - - - - -

Please follow the above process, accept the following content, and output the modified SQL.

```sql ```

Question:{Question}

Evidence:{Evidence}

SQL: {Sql}

[Database Schema]

{desc_db}

[Foreign keys]

{fk_info}

Figure 11: The prompt used by the SQLChecker agent for query formatting and reduction.
```