RSL-SQL: Mitigating the Risk of Schema Linking in Text-to-SQL Generation

Anonymous ACL submission

Abstract

Text-to-SQL generation aims to translate natural language questions into SQL statements. In Text-to-SQL based on large language models, schema linking is a widely adopted strategy to streamline the input for LLMs by selecting only relevant schema elements, therefore reducing noise and computational overhead. However, schema linking faces risks that require caution, including the potential omission of necessary elements and disruption of the structural integrity of database. To address these challenges, we propose a novel framework called RSL-SQL that combines bidirectional schema linking, contextual information augmentation, and risk hedging selection strategy. On the BIRD dataset, we use both forward and backward pruning methods to improve the recall rate of schema linking, achieving a strict recall rate of 94%, while reducing the number of input columns by 83%. Furthermore, it hedges the risk by voting between a full mode and a simplified mode enhanced with contextual information. Experiments of our approach are comparable to state-of-the-art accuracy on different benchmarks. Furthermore, our approach outperforms a series of GPT-4 based Text-to-SQL systems when adopting DeepSeek-V2 (much cheaper) with the same intact prompts. Extensive analysis and ablation studies confirm the effectiveness of each component in our framework.

1 Introduction

In recent years, the task of *Text-to-SQL*, i.e., translating natural language questions into structured query language—has emerged as a critical technology for enabling non-expert users to access relational databases (Qin et al., 2022; Wang et al., 2019; Katsogiannis-Meimarakis and Koutrika, 2023). With the rapid advancement of large language models (LLMs) in both understanding and generation (Achiam et al., 2023; Anthropic, 2024; OpenAI, 2024), a growing body of research has explored prompt engineering to guide LLMs in SQL generation, yielding impressive results. Prior studies have demonstrated that providing richer and more detailed database descriptions—such as the database's structure, annotations, sample data, and relational mappings—can significantly improve the accuracy of the generated SQL queries (Lee et al., 2024; Li et al., 2024d).

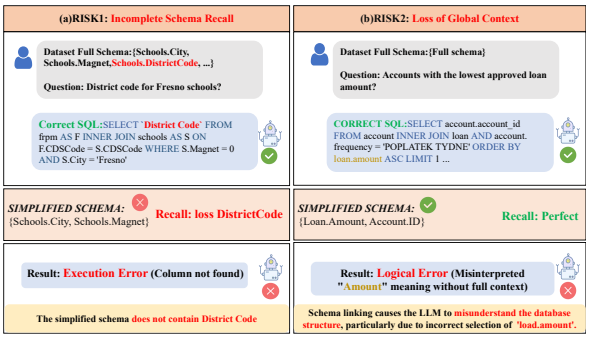

Figure 1: Two critical risks in schema linking: (a) **Incomplete Recall**, where necessary elements are discarded; and (b) **Volatile Gains**, where contextual loss hinders structural understanding.

For many real-world databases, the schema can be extensive, containing numerous tables and columns, especially with detailed descriptions (Li et al., 2024c; Lei et al., 2024). However, a user's natural language query usually pertains to only a much smaller subset of these database fields. Consequently, feeding the complete schema to the LLM for SQL generation may introduce a significant amount of irrelevant information, which can dilute the model's focus and hinder its ability to accurately comprehend the user's intent and generate precise SQL. To address this, many studies have adopted a **two-step cascading approach** in LLM-based Text-to-SQL pipelines (Li et al., 2024b; Talaei et al., 2024): **1) Schema Linking** aims to identify and extract relevant schema elements from the complete database schema, thereby forming a **simplified schema subset**; **2) SQL Generation** aims

to generated the target SQL query solely on above **simplified schema**.

However, schema linking can indeed bring about several risks. The **First Risk** is that *the performance of schema linking itself directly caps the upper bound of subsequent SQL accuracy*. If the simplified schema misses any necessary tables or columns required to answer the user's question, the generated SQL will inevitably be erroneous (Figure 1 (a)). The **Second Risk** is that *the gains from schema simplification on downstream SQL generation are somehow volatile*; even if all necessary schema elements are fully identified during the schema linking phase, it may not always yield positive gains (Figure 1 (b)). Precisely due to these risks, some studies (Maamari et al., 2024) have found that for particularly strong models like GPT-4o, the benefits brought by schema linking are often marginal or negligible. These risks highlight a delicate balance: while schema linking can reduce noise and computational cost, it may also introduce errors through incomplete schema recall or undermine the LLM's holistic comprehension of the complete database context.

To enhance the advantages of schema linking while mitigating its risks, we propose **RSL-SQL**, a framework for mitigating the risks of schema linking in Text-to-SQL generation. In RSL-SQL, we first introduce a bidirectional schema linking method to address the risk of incomplete recall. It combines *direct inference of relevant schema elements from the user question* with a reverse process that *parses schema elements from generated SQL draft for the user question*. By merging these, it achieves better coverage and efficiency than existing schema linking techniques. To handle the second risk of unstable effects from schema simplification, we enrich the simplified schema with additional contextual information. This makes it more informative and helps compensate for any loss in the LLM's overall understanding of the database. More critically, We apply a risk hedging strategy that selects the optimal SQL based on the two generated SQL queries from the complete schema and the enriched simplified schema. This approach provides a more focused alternative differs from traditional self-consistency methods (Li et al., 2024d), which generate multiple SQLs from the same schema. Together, these components allow RSL-SQL to amplify the benefits of schema linking while reducing its drawbacks, resulting in significant performance improvements.

In our experiments, we evaluate the proposed method, RSL-SQL, on the BIRD, Spider and Spider2.0 datasets, comparing its performance against a range of existing Text-to-SQL methods. RSL-SQL, powered by GPT-4o, delivers strong execution accuracy on BIRD and Spider. Additionally, it reaches high on Spider 2.0-lite when utilizing GPT-o3.Moreover, we demonstrate that when using the cost-effective model DeepSeek-v2, RSL-SQL outperforms many GPT-4-based methods on both datasets. The ablation study reveals that each component of our method contributes to the overall performance gains. The **main contributions** of this paper are summarized as follows: (1)**A Novel Text-to-SQL Framework:**We propose RSL-SQL, a comprehensive framework composed of two key modules: *Bidirectional Schema Linking* for efficient column pruning and *Question Information Augmentation* for semantic enhancement. This architecture effectively mitigates schema noise and enhances reasoning capabilities, enabling precise SQL generation in large-scale environments. (2)**Strong Generalizability Across Benchmarks:** RSL-SQL demonstrates strong transferability without domain-specific fine-tuning. It consistently achieves near-optimal performance (ranking second) across diverse scenarios, ranging from the standardized Spider dataset to the noisy BIRD and the massive Spider 2.0-Lite benchmarks. (3)**High Performance with Low Consumption:** We achieve a superior balance between accuracy and cost. Unlike methods relying on resource-intensive multi-candidate sampling, RSL-SQL delivers competitive execution accuracy with significantly lower token consumption and inference latency.

2 Related Work

2.1 Schema Linking in Text-to-SQL

Schema linking is a key step in text-to-SQL tasks, aiming to identify relevant database tables and columns for natural language queries. Models like RAT-SQL (Guo et al., 2019), SchemaGNN (Bogin et al., 2019), and ShadowGNN (Chen et al., 2021b) use relationship-aware self-attention mechanisms to enhance schema integration. SADGA (Cai et al., 2021) introduces a dual-graph framework to encode and aggregate information from both queries and schemas. With LLMs excelling in NLP tasks, methods like CHESS (Talaei et al., 2024) improve schema linking by retrieving relevant database information, while MCS-SQL (Lee

et al., 2024) uses multiple prompts and LLM responses to filter out irrelevant tables and columns. SQL-to-schema approaches perform schema linking by parsing the SQL query itself (Yang et al., 2024). Most recently, reasoning-enhanced models such as SQL-R1 (Ma et al., 2025) have emerged, which treat schema linking as a distinct reasoning step within a reinforcement learning or multi-agent framework to better align schema understanding with execution correctness.Our approach applies bidirectional schema linking to better analyze the relationship between the initial SQL and the necessary elements to answer the user's query.

2.2 Candidate Generation Strategies for SQL Synthesis

Early Text-to-SQL research relied on hand-designed templates (Zelle and Mooney, 1996), which worked well for simple cases but were limited by manual. With the rise of Transformer models, especially sequence-to-sequence architectures (Sutskever, 2014; Vaswani, 2017), Text-to-SQL research advanced significantly. Models like IRNet (Guo et al., 2019) and RAT-SQL (Wang et al., 2019) integrated schema information into SQL generation via relationship-aware attention mechanisms. As LLMs proved effective for NLP tasks, research increasingly focused on their potential for Text-to-SQL, introducing task decomposition and reasoning strategies like Chain of Thought (CoT) (Wei et al., 2022) to improve performance. Methods generating multiple candidate SQLs and selecting the best have shown success (Chen et al., 2021a; Li et al., 2022; Ni et al., 2023), such as using different prompts to generate and choose optimal solutions (Pourreza et al., 2024).In 2025, generation strategies have shifted towards reinforcement learning and self-correction ， such us Arctic-Text2SQL-R1 (Yao et al., 2025) optimizes generation via execution-based rewards. In contrast, our approach generates only two candidate SQLs—one from the full schema and one from the simplified schema—greatly reducing token usage and enhancing efficiency.

3 Method

We introduce **RSL-SQL**, a training-free framework for Text-to-SQL with LLMs. The framework addresses two primary risks: **Risk 1** (incomplete recall) is mitigated via a **Bidirectional Schema Linking** mechanism; **Risk 2** (volatile gains from simplification) is addressed through a pipeline comprising **Preliminary SQL Generation**, **Contextual Information Augmentation**, and a **Risk Hedging Selection Strategy**. Unlike self-consistency methods (Lee et al., 2024), RSL-SQL deterministically generates $\texttt{SQL}_1$ and $\texttt{SQL}_2$ using distinct schema information.

We formulate the task under two sub-tasks: (1) **Schema Element Recall**, identifying a subset $\mathcal{S}' \subseteq \mathcal{S}$ that maximally recalls all necessary elements required to generate the correct SQL. The objective is to ensure that the subset contains all essential schema elements; (2) **SQL Generation for Risk Hedging**, designing a strategy to maximize query reliability by balancing semantic precision and structural integrity.

The database schema $\mathcal{S}$ follows an M-schema style (Gao et al., 2024b), providing a modular description including table names, column types, foreign keys, and value samples to aid value-based reasoning (Li et al., 2024d). The detailed execution process is provided in **the Appendix A**

3.1 Bidirectional Schema Linking

To mitigate the risk of insufficient recall, we present **Bidirectional Schema Linking**. As compared in Figure 3 in the Appendix, unlike retrieval-based methods (Figure 3(a)) that score each elements individually or standard generative methods (Figure 3(b)) that select elements in a single pass, our approach (Figure 3(c)) combines forward schema linking and backward schema linking to derive a simplified database schema.

Forward Schema Linking In this step, we prompt the LLMs with database schema $\mathcal{S}$ and the natural language query $\mathcal{Q}$ to identify potentially relevant tables and columns. Since columns mentioned in the user query may be directly related to the correct SQL, we also perform an exact match to recall the columns explicitly referenced in the query. Together, these steps constitute *Forward Schema Linking*. To avoid naming conflicts, the output list is formatted as `table_name` and `table_name.column_name`. The prompt is shown in Figure 9. This yields a set of potentially relevant elements, denoted $L_{\texttt{fwd}}$:

$$L_{\texttt{fwd}} = \text{LLM}_{\texttt{fwd}}(\mathcal{Q}, \mathcal{S}) \cup \text{Match}_{\texttt{fwd}}(\mathcal{Q}, \mathcal{S}) \quad (1)$$

Backward Schema Linking We observed that while an initial SQL generated directly may be incorrect, the fields it references are often strongly

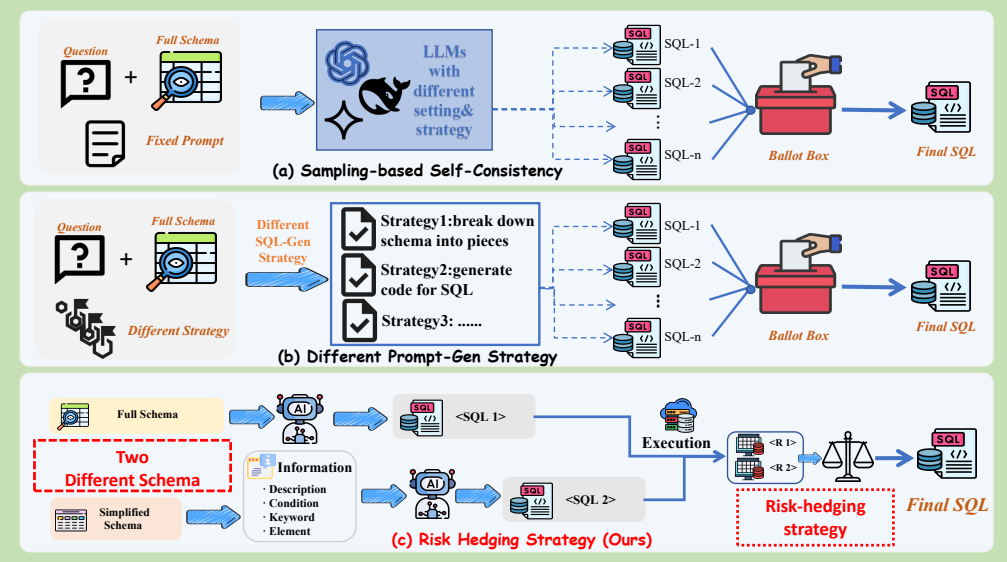

Figure 2: Comparison of SQL generation strategies. (a) Sampling-based Self-Consistency: Generates multiple candidate SQLs using a fixed schema and prompt with random decoding (Temperature > 0), followed by majority voting. (b) Prompt-based Strategy: Generates candidates using varying prompts and selects the best output. (c) Risk Hedging Strategy (Ours): Deterministically generates two complementary candidates—one from the Full Schema and one from the Simplified Schema —and employs a selection mechanism based on execution results to hedge against schema linking risks.

correlated with the correct SQL. Based on this, we propose *Backward Schema Linking*. First, the LLM is provided with the initial schema and L_{fwd} to guide SQL generation. Next, we parse the generated SQL using the sqlglot (tobymao, 2024) tool to extract referenced tables and columns, denoted as L_{bwd}: The backward linking set is derived as $L_{\text{bwd}} = \text{Tool}(\text{SQL}, \mathcal{S})$, where $\text{SQL} = \text{LLM}_{\text{gen}}(\mathcal{Q}, L_{\text{fwd}})$. This process complements forward schema linking, as L_{bwd} may capture important elements missed in L_{fwd}. *Particularly, if the generated SQL is already correct, L_{bwd} will certainly encompass all necessary tables and columns.*

Bidirectional Integration In each iteration, we perform forward and backward linking to obtain L_{fwd} and L_{bwd}. The number of iterations is adjustable. For Spider (Yu et al., 2018) and BIRD (Li et al., 2024c), we perform a single iteration, while for Spider2.0 (Lei et al., 2024), we perform three iterations. Finally, we merge the sets obtained in each iteration to derive the simplified schema $\mathcal{S}'$:

$$\mathcal{S}' = \bigcup_{i=1}^{N} \left(L_{\text{fwd}}^{(i)} \cup L_{\text{bwd}}^{(i)} \right) \quad (2)$$

We refer to this process as *Bidirectional Schema Linking*, which effectively eliminates unrelated details while preserving high recall. *In all subsequent steps, we rely only on the simplified schema $\mathcal{S}'$.*

3.2 Contextual Augmented SQL Generation

Schema linking simplifies the schema but may destroy the integrity of the complete database or obscure the connection between the user question and schema elements. To improve robustness, we introduce two augmentation strategies:Preliminary SQL Generation and Contextual Information Augmentation,which all collectively enhance the LLM's semantic and structural understanding of the database.Please refer to Appendix B for the detailed execution process of two steps.

3.3 Risk Hedging Selection Strategy

SQL_1 (from full schema) and SQL_2 (from simplified schema) each have distinct advantages: the former preserves structural integrity, while the latter reduces noise. To balance this trade-off, we design a **Risk Hedging Selection Strategy**.Figure 2 illustrates the difference between our approach and varying decoding strategies. Unlike Sampling-based Self-Consistency (Figure 2(a)) or Prompt-based strategies (Figure 2(b)) that rely on generating multiple candidates via randomness or varying prompts, our method (Figure 2(c)) deterministically generates two complementary candidates and employs execution-based binary selection.

SQL Execution and Result Annotation We execute both SQL_1 and SQL_2 on database $\mathcal{D}$ to obtain

results $\mathcal{R}_1$ and $\mathcal{R}_2$. We annotate these results by capturing error messages or summarizing returned rows like "No rows returned" or "100 rows total, showing first 5 rows".

LLM based Binary Selection We then prompt the LLM to select the better query between $\texttt{SQL}_1$ and $\texttt{SQL}_2$, denoted as $\texttt{SQL}_3$. The input includes $\hat{\mathcal{S}}'$, $\mathcal{Q}$, $\texttt{SQL}_1$, $\mathcal{R}_1$, $\texttt{SQL}_2$, and $\mathcal{R}_2$:

$$\texttt{SQL}_3 = \text{LLM}_{\text{select}}(\hat{\mathcal{S}}', \mathcal{Q}, \texttt{SQL}_1, \mathcal{R}_1, \texttt{SQL}_2, \mathcal{R}_2) \quad (3)$$

The goal of $\texttt{SQL}_3$ is to explicitly balance schema completeness and noise reduction. In cases where both candidates are flawed, the LLM may generate a new query (occurring in $\approx 2\%$ of cases). The prompt is shown in Figure 12 in the Appendix.

Besides, to address potential syntax errors in $\texttt{SQL}_3$, we employ an iterative self-correction mechanism guided by execution feedback (Pourreza et al., 2024). For **BIRD** and **Spider**, where schema complexity is moderate, we perform up to 3 rounds of refinement on $\texttt{SQL}_3$ to minimize token overhead. In contrast, for the more challenging **Spider 2.0** benchmark, we adopt an aggressive strategy by independently correcting $\texttt{SQL}_1$ and $\texttt{SQL}_2$ before performing the final selection to maximally mitigate hallucinations caused by massive schema fields.

4 Experiment

4.1 Benchmarks

To comprehensively evaluate the effectiveness and generalization capabilities of RSL-SQL across varying database complexities, we conduct extensive experiments on three prominent Text-to-SQL benchmarks: **BIRD**, **Spider**, and **Spider 2.0-Lite**. These datasets encompass a wide range of scenarios, from standardized academic schemas to massive, enterprise-scale real-world databases.

BIRD Benchmark The BIRD dataset (Li et al., 2024c) is a large-scale, cross-domain Text-to-SQL dataset. Its key characteristic is the emphasis on processing database values while highlighting the challenges posed by dirty external knowledge bases, noisy database values, and the efficiency of SQL queries—especially in large-scale database environments.

Spider Benchmark The Spider dataset (Yu et al., 2018) is a large-scale, cross-domain Text-to-SQL task dataset. Its key characteristic is that it not only contains complex SQL queries but also covers multi-table databases, making it an important resource for testing model generalization capabilities.

Spider2.0-Lite Benchmark The databases in Spider 2.0 (Lei et al., 2024) come from real applications, typically containing over 1,000 columns and stored in local or cloud systems such as BigQuery and Snowflake. Spider 2.0-Lite simplifies the task by offering a self-contained setup with well-prepared database metadata and documentation. With 547 examples, this version focuses on a text-in, text-out approach, enabling faster development and evaluation.

4.2 Evaluation Metrics

We evaluate the performance of RSL-SQL using four key metrics. **Execution Accuracy (EX)** measures the proportion of predicted SQL queries that functionally match the ground truth (Yu et al., 2018), while the **Valid Efficiency Score (VES)** assesses the execution efficiency of these correct queries (Li et al., 2024c). To quantify schema linking effectiveness, we employ **Non-Strict Recall (NSR)** for element-level overlap analysis and **Strict Recall Rate (SRR)** to determine the ratio of samples where all required elements are successfully recalled. Our framework prioritizes maximizing SRR while minimizing the linked schema size $|\tilde{\mathcal{S}}|$ to ensure both precision and robustness. Detailed mathematical formulations for each metric are provided in Appendix C.

4.3 Results and Analysis

4.3.1 SQL Generation Results

The comparative performance of RSL-SQL against representative baselines on BIRD (Li et al., 2024c), Spider (Yu et al., 2018), and Spider 2.0-Lite (Lei et al., 2024) is summarized in Table 1.

BIRD Benchmark On the BIRD dataset (Li et al., 2024c), RSL-SQL demonstrates high competitiveness using both GPT-4o and DeepSeek-V2 backbones. Within the landscape of prompt-based solutions, our framework ranks second only to $\text{CHESS}_{\text{IR+CG+UT}}$ (Talaei et al., 2024). However, it is important to note that CHESS requires a substantial computational budget to generate 20 candidate SQL queries, whereas RSL-SQL delivers superior Valid Efficiency Scores (VES) at a significantly lower inference cost. ***Crucially, as illustrated in Table 2, RSL-SQL significantly outperforms robust baselines such as E-SQL (Caferoğlu and Ulusoy, 2024) and CHESS$_{IR+SS+CG}$ (Talaei et al., 2024) when evaluated under identical DeepSeek***

Table 1: Comparison of different methods on BIRD, Spider, and Spider-2.0-Lite benchmarks. **Note:** 1) **MCS** column indicates Multi-Candidate Strategy; 2) We merge results from different benchmarks under the **Test EX** (Execution Accuracy) metric; 3) For RSL-SQL (Ours), we report performance across various models; 4) **Bold** denotes the best result, and underlined denotes the second best.

Method	Model	MCS	Test EX		
			BIRD	Spider	Spider 2.0
Fine-tuning-based methods					
DTS-SQL (Pourreza and Rafiei, 2024b)	DeepSeek-7B	✗	60.31	79.9	-
CodeS (Li et al., 2024b)	CodeS-7B	✗	59.25	-	-
CodeS (Li et al., 2024b)	CodeS-15B	✗	64.62	-	0.73
Distillery (Maamari et al., 2024)	GPT-4o (FT)	✗	71.83	-	-
MSc-SQL (Gorti et al., 2024)	Gemma-2-9B	✓	-	84.7	-
XiYan-SQL (Gao et al., 2024a)	-	✓	75.63	-	-
Prompt-based methods					
DIN-SQL (Pourreza and Rafiei, 2024a)	GPT-4	✗	55.90	85.3	1.46
DAIL-SQL (Gao et al., 2023)	GPT-4	✗	57.41	86.6	5.68
MCS-SQL (Lee et al., 2024)	GPT-4	✓	65.45	**89.6**	-
TA-SQL (Qu et al., 2024)	GPT-4	✗	59.14	-	-
SuperSQL (Li et al., 2024a)	GPT-4	✗	62.66	-	-
MAG-SQL (Xie et al., 2024)	GPT-4	✗	-	85.6	-
E-SQL (Caferoğlu and Ulusoy, 2024)	GPT-4o	✓	66.29	-	-
MAC-SQL (Wang et al., 2023)	GPT-3.5/4	✗	59.59	82.8	-
CHESS (Talaei et al., 2024)	Proprietary	✗	66.69	-	3.84
CHESS (Talaei et al., 2024)	Gemini-1.5	✓	**71.10**	-	-
Spider-Agent (Lei et al., 2024)	GPT-4o/R1	✗	-	-	13.71
ReFoRCE (Deng et al., 2025)	GPT-o3	✓	-	-	**37.84**
RSL-SQL (Ours)					
RSL-SQL	DeepSeek-V3	✗	64.08	87.5	26.14
RSL-SQL	GPT-4o	✗	68.70	87.9	19.01

Table 2: Comparison of different methods under the same LLM on the **BIRD and spider-2.0** benchmark.

Method	Model	MCS	Dev EX
BIRD			
E-SQL	DeepSeek-V3	✗	60.10
$\text{CHESS}_{\text{IR+SS+CG}}$	DeepSeek-V3	✗	61.34
RSL-SQL (Ours)	DeepSeek-V3	✗	**64.08**
Spider-2.0			
ReFoRCE (Deng et al., 2025)	GPT-o3	✓	**37.84**
RSL-SQL(Ours)	GPT-o3	✗	33.09

backbones, all while maintaining higher token efficiency.

Spider Benchmark To verify the generalization capability across simpler schema structures, we evaluated RSL-SQL on the Spider benchmark (Yu et al., 2018). Our approach secures the second-best position, trailing only MCS-SQL (Lee et al., 2024)—which relies on an intensive Self-Consistency strategy (Wang et al., 2022) involving up to 60 candidates. This result confirms that RSL-SQL remains highly effective even in environments with standardized naming and moderate complexity.

Spider 2.0-Lite Benchmark Transitioning to enterprise-scale schemas with thousands of fields, Spider 2.0-Lite (Lei et al., 2024) presents a far more rigorous challenge. While established methods like DIN-SQL (Pourreza and Rafiei, 2024a) and DAIL-SQL (Gao et al., 2023) experience sharp performance declines in this setting, RSL-SQL maintains strong accuracy. Utilizing GPT-4o, our framework outperforms these baselines by a margin exceeding 10 percentage points, ranking second overall behind only ReFoRCE (Deng et al., 2025).

4.3.2 Schema Linking Results

Schema linking quality is critical to both SQL accuracy and token efficiency. We evaluate our bidirectional schema linking method on the BIRD and Spider 2.0-Lite benchmarks, with results detailed in Table 3.

BIRD Benchmark On the BIRD development set, we compare RSL-SQL against two primary categories of schema linking. *Fine-tuning-based methods*, such as CodeS (Li et al., 2024b), RESD-SQL (Li et al., 2023), and FinSQL (Zhang et al., 2024), typically rely on trained T5-based Cross-Encoders (C-E), which score and rank schema elements based on their relevance to the natural language question. While our reproduction of the CodeS retriever shows that strict recall (SRR) improves as the number of retrieved columns increases (Top-10 to Top-30), its Top-30 performance remains only marginally higher than our DeepSeek-

Table 3: Comparison of **SRR** and **NSR** metrics across different methods on the BIRD development set and Spider2.0-Lite benchmark. **Avg. C** denotes the average number of columns input per question.

Data	Method	NSR	SRR	Avg. C
BIRD	Full Schema	100	100	76.28
	Gold	100	100	4.74
	Fine-tuning-based methods			
	C-E (K=10)	88.05	77.83	10.00
	C-E (K=20)	95.90	89.30	19.14
	C-E (K=30)	**97.69**	93.74	27.82
	Prompt-based methods			
	HySCSL [Maamari et al. (2024)]	-	90.36	-
	SCSL [Maamari et al. (2024)]	-	88.77	-
	HyTCSL [Maamari et al. (2024)]	-	83.00	-
	TCSL [Maamari et al. (2024)]	-	77.44	-
	MCS [Lee et al. (2024)]	-	89.80	-
	CHESS [Talaei et al. (2024)]	92.10	86.63	6.31
	RSL-SQL (DeepSeek-V3)	97.29	93.28	14.85
	RSL-SQL (GPT-4o)	**97.69**	**94.32**	13.02
Spider2.0	C-E (K=100)	81.27	65.60	107.33
	C-E (K=200)	86.86	75.20	168.98
	C-E (K=300)	89.43	80.40	222.18
	C-E (K=400)	90.44	81.20	264.97
	C-E (K=500)	91.49	82.40	305.42
	ReFoRCE(DeepSeek-V3)	93.94	**90.00**	410.57
	RSL-SQL (DeepSeek-V3)	**94.53**	85.60	87.75
	RSL-SQL (GPT-o1)	94.10	82.00	82.15

V2 results. In contrast, *prompt-based methods* like CHESS (Talaei et al., 2024) and MCS-SQL (Lee et al., 2024) require dozens of LLM calls, yet achieve SRR significantly lower than ours. RSL-SQL invokes the LLM only twice; with GPT-4o, we achieve a 94.32% SRR and 97.69% NSR with an average of only 13 columns, demonstrating a substantial advantage in both precision and efficiency.

Spider 2.0-Lite Benchmark For enterprise-scale schemas in Spider 2.0-Lite, RSL-SQL demonstrates superior field reduction capabilities. Compared to the 500-column cross-encoder baseline, using GPT-o1 reduces the input column count by approximately 73% while maintaining a comparable SRR. While ReFoRCE (Deng et al., 2025) achieves a high SRR of 90%, it is limited to table linking and results in an average of 410 columns, failing to achieve effective column reduction. ***Notably, RSL-SQL maintains strong performance when using the DeepSeek-V3 model, confirming that our approach's overhead is not a bottleneck even in complex environments.***

4.4 Ablation Study

We conduct ablation experiments on the BIRD development set to investigate the impact of each component in RSL-SQL on execution accuracy in Table 4. Additionally, we explore the effectiveness of bidirectional schema linking on complex benchmarks in Table 7.

Table 4: Ablation Study on Execution Accuracy and Contribution of Each Component at Different Difficulty Levels on the BIRD Development Set Based on GPT-4o Models.

Step	Execution Accuracy			
	Simple	Moderate	Challenging	Total
Step 1	70.05	52.59	48.28	62.71
Step 2	72.11	54.74	53.10	65.06
Step 3	**74.38**	**57.11**	**53.79**	**67.21**
Step 2	72.11	54.74	53.10	**65.06**
- w/o SIA	71.03	53.66	51.03	63.89
- w/o QIA	71.24	54.09	48.97	63.95
- w/o Both	70.70	49.35	49.66	62.26

SQL Generation and Risk Hedging As shown in Table 4, each component in our framework improves execution accuracy by 1% to 3%. Notably, performance significantly deteriorates without **contextual information augmentation (CIA)**, consistent with findings in Maamari et al. (2024). CIA improves accuracy from 62.26% to 65.06% by mitigating simplification risks. Detailed ablation on H_{aug} components ($H_{\text{e}}, H_{\text{c}}, H_{\text{k}}$) on the subsampled set confirms their collective effectiveness in Table 6.

The **risk-hedging selection strategy** balances schema integrity and noise reduction by choosing between SQL_1 and SQL_2, contributing a crucial 2% accuracy gain. As illustrated in Fig. 4 in the Appendix, this strategy maintains high execution accuracy even under low strict recall scenarios. In the left panel, while single-directional Forward linking leads to a decline in accuracy due to low recall, our selection strategy consistently yields substantial gains.

Schema Linking Analysis As shown in Table 7, our bidirectional mechanism effectively mitigates the risk of losing critical fields. On BIRD, forward linking achieves 84% recall, whereas backward linking reaches 90%, indicating its unique advantages. Their integration reaches a SOTA 94% recall without significantly increasing column overhead. Furthermore, evaluating retrieval-based linking at different k values reveals that even when recall is sub-optimal, RSL-SQL's risk-hedging maintains competitive accuracy across all stages.

4.5 Token Consumption

On BIRD benchmark, we replicate the MAC-SQL (Wang et al., 2023), TA-SQL (Qu et al., 2024), E-SQL (Caferoğlu and Ulusoy, 2024), and CHESS (Talaei et al., 2024) methods. Due to cost limitations, we randomly select 100 samples

Table 5: Token Consumption and Execution Accuracy Across Different Methods. **Bold** denotes the best result, and underlined denotes the second best.

Benchmark	Method	Model	Avg. Tokens		EX
			Input	Output	
BIRD Dev	MAC-SQL (Wang et al., 2023)	GPT-4	**6.20 k**	0.58 k	59.39
	TA-SQL (Qu et al., 2024)	GPT-4	6.90 k	**0.33 k**	56.19
	E-SQL (Caferoğlu and Ulusoy, 2024)	GPT-4o	43.80 k	0.93 k	65.58
	$\text{CHESS}_{\text{IR+SS+CG}}$ (Talaei et al., 2024)	–	114.25 k	3.26 k	65.00
	$\text{CHESS}_{\text{IR+CG+UT}}$ (Talaei et al., 2024)	Gemini-1.5	276.21 k	12.80 k	**68.31**
	RSL-SQL (ours)	DeepSeek-V3	14.57 k	0.53 k	63.56
	RSL-SQL (ours)	GPT-4o	14.22 k	0.47 k	67.21
Spider 2.0	ReFoRCE (Deng et al., 2025)	DeepSeek-R1	524.11 k	73.28 k	29.61
	RSL-SQL (ours)	DeepSeek-R1	70.45 k	29.55 k	**30.53**

Table 6: Ablation Study on the Question Information Augmentation Component on the Subsampled Development Set Using GPT-4o Model, $H_{\text{kec}} = \{H_{\text{e}}, H_{\text{c}}, H_{\text{k}}\}$.

Setting	Execution Accuracy			
	Simple	Moderate	Challenging	Total
Step 2: CIA	**72.84**	**51.85**	**41.67**	**62.59**
- w/o H_{e}	71.60	44.40	50.00	59.86
- w/o H_{k}	72.84	50.00	41.67	61.90
- w/o H_{c}	70.37	50.00	50.00	61.22
- w/o H_{kec}	69.14	50.00	50.00	60.54

Table 7: Ablation Study of Schema Linking on the BIRD and Spider2.0-Lite Benchmarks.

Data	Model	Setting	NSR	SRR	Avg. C
BIRD	GPT-4o	Bidirectional	97.69	94.32	13.02
		· Forward	90.04	84.74	9.12
		· Backward	95.54	90.48	10.29

from the BIRD development set to estimate the economic and time costs of these methods on this subset. As shown in Table 5, compared to MAC-SQL (Wang et al., 2023) and TA-SQL(Qu et al., 2024), our method consumes twice as many input tokens but incurs significantly lower economic overhead and achieves higher execution accuracy. In comparison to E-SQL (Caferoğlu and Ulusoy, 2024), which uses three times more input tokens and incurs higher costs, our method outperforms E-SQL (Caferoğlu and Ulusoy, 2024) in execution accuracy with the GPT-4o model. Compared to $\text{CHESS}_{\text{IR+SS+CG}}$ (Talaei et al., 2024), which has lower execution accuracy and consumes eight times more input tokens and almost twice the processing time, our method still outperforms. Compared to $\text{CHESS}_{\text{IR+CG+UT}}$ (Talaei et al., 2024), despite a slightly lower execution accuracy with GPT-4o, it consumes 19 times more input tokens, 27 times more output tokens, and has much higher economic overhead. Using the DeepSeek-V2 model, we maintain high accuracy with only need 80 times less economic overhead per question cost of CHESS to maintain comparable high accuracy. Overall, our method strikes an excellent balance between performance and cost-effectiveness.

For the Spider2.0-Lite benchmark, we reproduce the ReFoRCE method with its sampling set to 8. Although ReFoRCE achieves a higher execution accuracy of 37.84% when using the GPT-o3 model, surpassing our method, its performance is slightly lower than ours when using the same DeepSeek-R1 model. Moreover, token statistics reveal that the input token count for ReFoRCE is seven times higher than that of our method. Overall, our approach strikes a good balance between performance and cost-effectiveness.

5 Conclusion

We propose the RSL-SQL framework, which mitigates the risks brought by schema linking and achieves performance comparable to the SOTA on the BIRD development set. We conduct an in-depth analysis of how schema linking recall rates affect the execution accuracy of SQL generation for different models. Through ablation studies, we validate the effectiveness of each component within the framework and investigate their operational mechanisms. Additionally, we compare the computational costs of various methods, demonstrating that our framework achieves an excellent balance between performance and cost, making it a highly efficient and cost-effective solution.

Limitations

While our proposed text-to-SQL generation framework demonstrates significant improvements in performance and efficiency, it is essential to acknowledge some limitations.

First, our approach relies heavily on the quality and coverage of the schema linking process. Al-

though our bidirectional schema pruning technique achieves a high strict recall rate, there may still be edge cases where relevant schema elements are not captured (e.g., implicitly referenced columns in complex reasoning scenarios), potentially impacting the accuracy of the generated SQL queries.

Second, the effectiveness of our information augmentation strategy may vary depending on the complexity and domain of the database. In some cases, the generated additional elements may not fully capture the nuances of the database structure or specific domain knowledge, leading to suboptimal SQL generation.

Finally, like most prompt-based methods, our approach is sensitive to the underlying LLM's capability. While we demonstrated effectiveness on GPT-4o and DeepSeek models, performance variations across other models remain an area for further investigation. Despite these limitations, our work presents a significant step forward in mitigating the risks associated with schema linking.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Anthropic. 2024. Meet claude.

Ben Bogin, Matt Gardner, and Jonathan Berant. 2019. Representing schema structure with graph neural networks for text-to-sql parsing. *arXiv preprint arXiv:1905.06241*.

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

Ruichu Cai, Jinjie Yuan, Boyan Xu, and Zhifeng Hao. 2021. Sadga: Structure-aware dual graph aggregation network for text-to-sql. *Advances in Neural Information Processing Systems*, 34:7664–7676.

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *arXiv preprint arXiv:2402.03216*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021a. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Zhi Chen, Lu Chen, Yanbin Zhao, Ruisheng Cao, Zihan Xu, Su Zhu, and Kai Yu. 2021b. Shadowgnn: Graph projection neural network for text-to-sql parser. *arXiv preprint arXiv:2104.04689*.

Minghang Deng, Ashwin Ramachandran, Canwen Xu, Lanxiang Hu, Zhewei Yao, Anupam Datta, and Hao Zhang. 2025. Reforce: A text-to-sql agent with self-refinement, format restriction, and column exploration. In *ICLR 2025 Workshop: VerifAI: AI Verification in the Wild*.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023. Text-to-sql empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363*.

Yingqi Gao, Yifu Liu, Xiaoxia Li, Xiaorong Shi, Yin Zhu, Yiming Wang, Shiqi Li, Wei Li, Yuntao Hong, Zhiling Luo, Jinyang Gao, Liyu Mou, and Yu Li. 2024a. A preview of xiyan-sql: A multi-generator ensemble framework for text-to-sql. *arXiv preprint arXiv:2411.08599*.

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

Satya Krishna Gorti, Ilan Gofman, Zhaoyan Liu, Jiapeng Wu, Noël Vouitsis, Guangwei Yu, Jesse C Cresswell, and Rasa Hosseinzadeh. 2024. Msc-sql: Multi-sample critiquing small language models for text-to-sql translation. *arXiv preprint arXiv:2410.12916*.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-sql in cross-domain database with intermediate representation. *arXiv preprint arXiv:1905.08205*.

George Katsogiannis-Meimarakis and Georgia Koutrika. 2023. A survey on deep learning approaches for text-to-sql. *The VLDB Journal*, 32(4):905–936.

Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. 2024. Mcs-sql: Leveraging multiple prompts and multiple-choice selection for text-to-sql generation. *arXiv preprint arXiv:2405.07467*.

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. *arXiv preprint arXiv:2411.07763*.

Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024a. The dawn of natural language to sql: Are we fully ready? *arXiv preprint arXiv:2406.01265*.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023. 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.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024b. Codes: Towards building open-source language models for text-to-sql. *Proceedings of the ACM on Management of Data*, 2(3):1–28.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, and 1 others. 2024c. 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.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, and 1 others. 2022. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.

Zhishuai Li, Xiang Wang, Jingjing Zhao, Sun Yang, Guoqing Du, Xiaoru Hu, Bin Zhang, Yuxiao Ye, Ziyue Li, Rui Zhao, and 1 others. 2024d. Pet-sql: A prompt-enhanced two-stage text-to-sql framework with cross-consistency. *arXiv preprint arXiv:2403.09732*.

Peixian Ma, Xialie Zhuang, Chengjin Xu, Xuhui Jiang, Ran Chen, and Jian Guo. 2025. Sql-r1: Training natural language to sql reasoning model by reinforcement learning. *arXiv preprint arXiv:2504.08600*.

Karime Maamari, Fadhil Abubaker, Daniel Jaroslawicz, and Amine Mhedhbi. 2024. The death of schema linking? text-to-sql in the age of well-reasoned language models. *arXiv preprint arXiv:2408.07702*.

Ansong Ni, Srini Iyer, Dragomir Radev, Veselin Stoyanov, Wen-tau Yih, Sida Wang, and Xi Victoria Lin. 2023. Lever: Learning to verify language-to-code generation with execution. In *International Conference on Machine Learning*, pages 26106–26128. PMLR.

OpenAI. 2024. Hello gpt-4o.

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. 2024a. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems*, 36.

Mohammadreza Pourreza and Davood Rafiei. 2024b. Dts-sql: Decomposed text-to-sql with small large language models. *arXiv preprint arXiv:2402.01117*.

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*.

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. *arXiv preprint arXiv:2405.15307*.

I Sutskever. 2014. Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*.

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*.

tobymao. 2024. sqlglot: Python sql parser and transpiler. https://github.com/tobymao/sqlglot.

A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2019. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. *arXiv preprint arXiv:1911.04942*.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Qian-Wen Zhang, Zhao Yan, and Zhoujun Li. 2023. Mac-sql: Multi-agent collaboration for text-to-sql. *arXiv preprint arXiv:2312.11242*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

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. *arXiv preprint arXiv:2408.07930*.

Sun Yang, Qiong Su, Zhishuai Li, Ziyue Li, Hangyu Mao, Chenxi Liu, and Rui Zhao. 2024. Sql-to-schema enhances schema linking in text-to-sql. In *International Conference on Database and Expert Systems Applications*, pages 139–145. Springer.

Zhewei Yao, Guoheng Sun, Lukasz Borchmann, Zheyu Shen, Minghang Deng, Bohan Zhai, Hao Zhang, Ang Li, and Yuxiong He. 2025. Arctic-text2sql-r1: Simple rewards, strong reasoning in text-to-sql. *arXiv preprint arXiv:2505.20315*.

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. *arXiv preprint arXiv:1809.08887*.

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.

Chao Zhang, Yuren Mao, Yijiang Fan, Yu Mi, Yunjun Gao, Lu Chen, Dongfang Lou, and Jinshu Lin. 2024. Finsql: Model-agnostic llms-based text-to-sql framework for financial analysis. *Preprint*, arXiv:2401.10506.

A Detailed Process of RSL-SQL Framework

The execution process of the RSL-SQL framework follows a four-step pipeline designed to mitigate the inherent risks of schema linking while maintaining high generation accuracy.

In **Step 1: Bidirectional Schema Linking (BSL)**, the system identifies a precise subset of the database schema. It starts with *Forward Linking*, which extracts potentially relevant elements based on the user question and the full schema. An initial SQL draft (SQL_1) is then generated to perform *Backward Linking*, where schema elements are extracted directly from the predicted SQL to capture essential columns that might have been missed during the forward link. These elements are merged to form a simplified and pruned schema $\mathcal{S}'$.

In **Step 2: Contextual Information Augmentation (CIA)**, the framework enhances the LLM's understanding of the simplified schema. By generating semantic descriptions for columns and decomposing the user question into intermediate SQL components (such as specific conditions and keywords), the system provides an augmented context H_{aug}. This context guides the generation of a second candidate query, SQL_2.

In **Step 3: Risk Hedging Selection (RHS)**, the framework performs a deterministic selection between SQL_1 (representing structural integrity) and SQL_2 (representing noise reduction). Both queries are executed and their results are passed to an LLM-based binary selector. This selector evaluates the reliability of each candidate based on the execution outcomes and the original question, effectively hedging against potential failures in the schema linking process.

Algorithm 1: RSL-SQL Framework Execution Process

Input: Full Schema $\mathcal{S}$, User Question $\mathcal{Q}$, Value Samples $\mathcal{V}$, Few-Shot $\mathcal{E}$
Output: Final Executable SQL $\texttt{SQL}_4$

Step 1: Bidirectional Schema Linking (BSL)
$L_{\text{fwd}} \leftarrow \texttt{FwdLink}(\mathcal{S}, \mathcal{Q})$
$\texttt{SQL}_1 \leftarrow \texttt{GenSQL}(\mathcal{S}, L_{fwd}, \mathcal{Q}, \mathcal{E})$ `// Initial generation`
$L_{\text{bwd}} \leftarrow \texttt{ExtractElements}(\texttt{SQL}_1)$
$\mathcal{S}' \leftarrow \texttt{SimplifiedSchema}(\mathcal{S}, L_{fwd} \cup L_{bwd})$ `// Schema Pruning`
Step 2: Contextual Information Augmentation (CIA)
$\mathcal{D}, H_{\text{cond}} \leftarrow \texttt{AugmentInfo}(\mathcal{S}', \mathcal{Q})$
$H_{\text{aug}} \leftarrow \{\mathcal{D}, H_{\text{cond}}\}$
$\texttt{SQL}_2 \leftarrow \texttt{GenSQL}(\mathcal{S}', H_{aug}, \mathcal{Q}, \mathcal{E})$ `// Augmented generation`
Step 3: Risk Hedging Selection (RHS) `// Core Strategy`
$\mathcal{R}_1 \leftarrow \texttt{Execute}(\texttt{SQL}_1)$; $\mathcal{R}_2 \leftarrow \texttt{Execute}(\texttt{SQL}_2)$
if $\mathcal{R}_1 == \mathcal{R}_2$ *and* $\mathcal{R}_2 \neq \emptyset$ then `// Consistency`
  $\texttt{SQL}_3 \leftarrow \texttt{SQL}_2$
else if $\texttt{SQL}_1$ *is Error and* $\texttt{SQL}_2$ *is Valid* then
  $\texttt{SQL}_3 \leftarrow \texttt{SQL}_2$ `// Avoid noise error`
else `// Conflict/Error`
  $\texttt{SQL}_3 \leftarrow \texttt{LLM\_Select}(\texttt{SQL}_1, \mathcal{R}_1, \texttt{SQL}_2, \mathcal{R}_2, \mathcal{Q})$
end
Step 4: Multi-Turn Self-Correction (MTSC)
for *iteration* $i = 1$ to N do
  $E \leftarrow \texttt{Execute}(\texttt{SQL}_3)$
  if E *is Empty* then **break**
  $\texttt{SQL}_3 \leftarrow \texttt{SelfCorrect}(\texttt{SQL}_3, E, \mathcal{S}')$
end
return $\texttt{SQL}_3$

Finally, in **Step 4: Multi-Turn Self-Correction (MTSC)**, the selected query SQL_3 undergoes an iterative refinement process. The system executes the query and, if errors are detected, uses the execution feedback to correct the SQL until a valid or optimal statement SQL_4 is produced. The complete logic is formalized in Algorithm 1.

B Details of Contextual Augmented SQL Generation

To improve the robustness of the framework against information loss during schema pruning, we introduce a contextual augmentation pipeline. As discussed in Section 3, while schema linking simplifies the input, it may inadvertently obscure structural connections or semantic nuances. We address this through the following two augmentation strate-

gies:

Step 1: Preliminary SQL Generation To leverage the structural integrity of the full database, we first provide the LLM with the complete schema $\mathcal{S}$, query $\mathcal{Q}$, few-shot demonstrations $\mathcal{E}$, and the forward linking result $L_{\texttt{fwd}}$ to generate an initial SQL query, denoted as $\texttt{SQL}_1$. The prompt is shown in Figure 7. Empirically, incorporating $L_{\texttt{fwd}}$ slightly enhances accuracy:

$$\texttt{SQL}_1 = \text{LLM}_{\texttt{gen}}(\mathcal{Q}, \mathcal{S}, \mathcal{E}, L_{\texttt{fwd}}) \tag{4}$$

Step 2: Contextual Information Augmentation This step enhances the LLM's understanding of the simplified schema $\mathcal{S}'$ from two perspectives:

1) Schema Information Augmentation. Describing a database solely with $\mathcal{S}'$ may lack semantic clarity. Following (Qu et al., 2024), we prompt LLMs to generate natural language descriptions for each column. This process is performed *once offline per database*, serving to semantically enrich the simplified schema $\mathcal{S}'$. We denote the augmented simplified schema as $\hat{\mathcal{S}}'$:

$$\hat{\mathcal{S}}' = \left\{ \text{LLM}_{\texttt{desc}}(c) \;\middle|\; c \in \mathcal{S}' \right\} \tag{5}$$

2) Question Information Augmentation $H_{\texttt{aug}}$. To reduce reasoning burden, we decompose the user query $\mathcal{Q}$ to extract partial SQL components. We issue targeted prompts with $\hat{\mathcal{S}}'$ and $\mathcal{Q}$ to generate: **Elements** (H_e) (relevant tables/columns), **Conditions** (H_c) (constraints for WHERE clause), and **Keywords** (H_k) (e.g., `DISTINCT`, `GROUP BY`). These form the augmented context $H_{\texttt{aug}}$:

$$H_{\texttt{aug}} = \text{LLM}_{\texttt{gen}}(\mathcal{Q}, \hat{\mathcal{S}}') \tag{6}$$

Finally, we prompt the LLM to generate a second candidate SQL query, denoted as $\texttt{SQL}_2$, using the simplified schema and augmented context:

$$\texttt{SQL}_2 = \text{LLM}_{\texttt{gen}}(\mathcal{Q}, \hat{\mathcal{S}}', \mathcal{E}, H_{\texttt{aug}}) \tag{7}$$

The prompts for $H_{\texttt{aug}}$ and $\texttt{SQL}_2$ are shown in Figures 8,9,10,and11 .

C Evaluation Metrics

To rigorously evaluate the performance of the RSL-SQL framework, we employ a comprehensive set of metrics that assess both the final SQL generation quality and the intermediate schema linking effectiveness.

C.1 SQL Generation and Efficiency

The following metrics are used to determine the correctness and execution performance of the generated SQL statements:

Execution Accuracy (EX) This metric measures the proportion of predicted SQL queries whose execution results match the ground truth. It serves as the primary indicator of the model's functional correctness.

Valid Efficiency Score (VES) Beyond mere correctness, VES assesses the relative execution efficiency of valid SQL queries. It is particularly relevant for large-scale databases where query optimization is crucial to minimize computational costs.

C.2 Schema Linking Effectiveness

To quantify the precision and recall of our bidirectional schema linking strategy, we define the following schema-level metrics:

Non-Strict Recall (NSR) NSR evaluates the recall of schema elements at the individual level. It is defined as the ratio of successfully linked schema elements to the total number of ground truth elements across all questions:

$$\text{NSR} = \frac{\sum_{i=1}^{n} |S_{\text{gt},i} \cap \tilde{S}_i|}{\sum_{i=1}^{n} |S_{\text{gt},i}|} \tag{8}$$

where n is the total question count, while $S_{\text{gt},i}$ and $\tilde{S}_i$ denote the ground truth and linked schema elements for the i-th question, respectively.

Strict Recall Rate (SRR) SRR provides a more stringent assessment by calculating the ratio of samples where *all* required schema elements are successfully recalled:

$$\text{SRR} = \frac{\sum_{i=1}^{n} I(\tilde{S}_i \supseteq S_{\text{gt},i})}{n} \tag{9}$$

where $I(\cdot)$ is the indicator function. Because missing a single essential column often leads to unrecoverable errors in the final SQL, our framework prioritizes maximizing SRR to ensure a high performance ceiling for downstream generation.

D Case Study

Our analysis before reveals that schema linking involves a fundamental trade-off between noise reduction and information loss. Simplifying the complete schema through schema linking has dual effects: it can correct queries that were originally

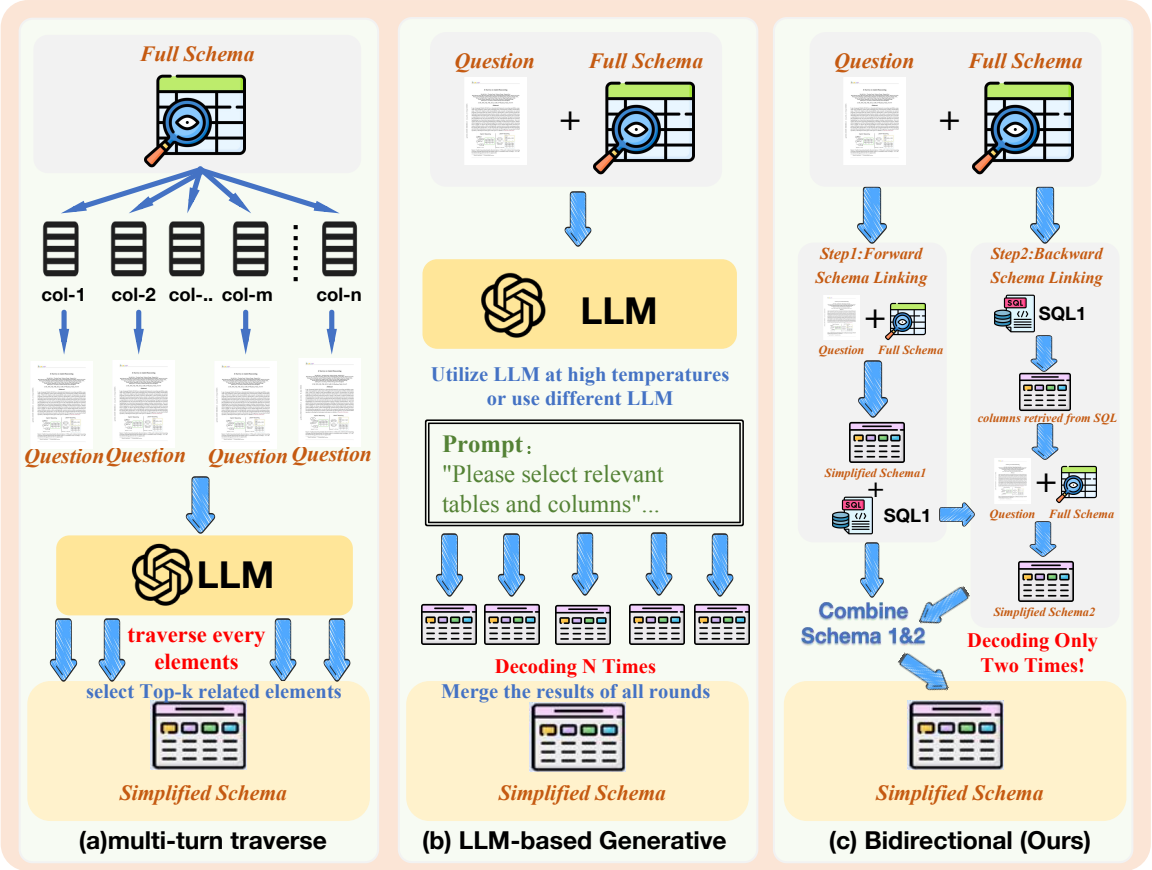

Figure 3: Comparison of schema linking paradigms. (a) Retrieval-based Schema Linking: Computes similarity scores between the question and each schema element individually using a neural network. (b) Generative Schema Linking: Prompts an LLM to directly select relevant elements from the full schema. (c) Bidirectional Schema Linking (Ours): Combines Forward Linking (direct selection) and Backward Linking (parsing schema elements from a preliminarily generated SQL) to maximize recall and mitigate the risk of missing necessary columns.

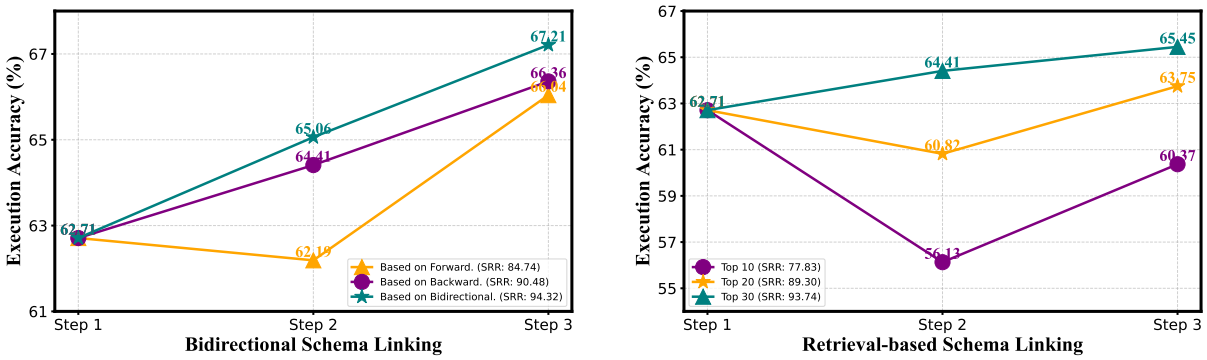

Figure 4: Risk Mitigation Analysis on BIRD. **Left:** Comparison of bidirectional linking vs. single-directional baselines across stages. **Right:** Impact of different k values in retrieval-based schema linking (Li et al., 2024b). The selection strategy consistently buffers against recall failures.

incorrect in the complete schema (positive gain, denoted as y) but may also cause initially correct queries to become incorrect (negative impact, denoted as x). Thus, the net benefit of schema linking depends on the balance between these effects. Schema linking yields a net positive effect only when $y - x > 0$; otherwise, it may introduce more harm than benefit.

The RSL-SQL framework ensures the correctness of SQL generation through contextual information augmentation, and risk hedging selection strategy. As shown in Figure 5, even if errors occur

Figure 5: Four Examples Demonstrating the Robustness of Our Framework.

in earlier steps, the subsequent steps can correct these errors and improve SQL accuracy through specific methods.

The core of the framework lies in *Step 2: Contextual Information Augmentation* and *Step 3: Risk Hedging Selection Strategy*. To illustrate more intuitively how contextual information augmentation enhances performance by increasing positive gain and how risk hedging selection strategy mitigates risks by reducing negative impact, we have selected two representative examples to demonstrate their underlying mechanisms in detail.

D.1 Maximize Positive Gain

The description of each column is undoubtedly useful because it helps LLM understand the specific meaning of each column and build a mapping relationship between user questions and columns. The textual information generated by LLM covers database elements, SQL keywords, and query conditions. Among them, the further filtered database elements can guide LLM to focus on elements that are more likely to be used; the generated SQL keywords can remind LLM of critical but often overlooked constraints; the generated conditions are obtained from the user which is obtained by decomposing the problem can help LLM solve the problem step by step in a more orderly manner. With this enhanced information, LLM is able to focus more attention on key parts that were previously under-focused, thereby significantly increasing positive gain. For example, the following is an example in BIRD,

Question:

What is the preferred foot when attacking of the player with
the lowest potential?

Contextual Information:

```
Elements:
[player_attributes.potential]
[player_attributes.preferred_foot]
Key Words: ['MIN', '=']
Conditions:
['preferred foot when attacking']
['player with the lowest potential']
```

✗ SQL_1 Genereted in Step 1:

```
SELECT preferred_foot
FROM Player_Attributes
ORDER BY potential ASC
LIMIT 1
```

✓ SQL_2 Generated in Step 2:

```
SELECT preferred_foot
FROM player_attributes
WHERE potential = (
 SELECT MIN(potential)
 FROM player_attributes
)
```

In this example, the incorrect generation of SQL_1 stems from failing to account for parallel conditions, leading to an initial SQL with deviations. The generated textual information comprehensively covered the critical columns required for SQL generation, and the inclusion of the keyword **'MIN'** explicitly indicated the possibility of parallel conditions. This supplementary information enabled the LLM to better grasp the semantic nuances of the

query and the structure of the database, effectively guiding it to produce the correct SQL.

D.2 Minimize Negative Impact

The risk hedging selection strategy is a pivotal step in our approach. By choosing the more accurate SQL from the two statements generated using the complete schema and the simplified schema, respectively, this strategy strikes a balance between preserving schema completeness and minimizing noise. An illustrative example is provided below,

Question:

Is molecule TR151 carcinogenic?

✓ **SQL_1 Generated in Step 1:**

```
SELECT label
FROM molecule
WHERE molecule_id = 'TR151'
```

✗ **SQL_2 Generated in Step 2:**

```
SELECT label
FROM molecule
WHERE molecule_id = 'TR151'
AND label = '+'
```

SQL execution results:

```
SQL1 execution results:
[Row_count, Column_count] = [1, 1]
[Result] = [('-',)]
SQL2 execution results:
[Row_count, Column_count] = [0, 0]
[Result] = []
```

✓ **SQL_3 Selected in Step 3:**

```
SELECT label
FROM molecule
WHERE molecule_id = 'TR151'
```

Introducing new components in Step 2 may inevitably lead to some originally correct SQL statements being modified incorrectly. However, our risk hedging selection strategy minimizes this negative impact as much as possible. By incorporating execution information, the strategy enables the LLM to further analyze the candidate results, facilitating a more accurate selection of the correct SQL and significantly reducing associated risks.

E Prompt

E.1 Prompt Construction

In this section, we describe the construction of our prompt, which consists of the user's question $\mathcal{Q}$, value samples, and few-shot demonstrations $\mathcal{E}$. A concrete example illustrating how these components are structured in the LLM prompt is shown in Figure 6.

User's Question $\mathcal{Q}$ For the BIRD benchmark (Li et al., 2024c), the user question is merged with its corresponding *evidence* (e.g., domain-specific definitions or formula explanations) to form a complete query. For the Spider benchmark (Yu et al., 2018), we directly utilize the original question as no additional evidence is available.

Value Samples To assist the model in understanding the database content, we randomly select 5 rows from each table as samples. To ensure prompt conciseness and prevent lengthy text fields from cluttering the schema, any column value exceeding 50 characters is truncated using an ellipsis marker ("[...]").

Few-shot Demonstrations $\mathcal{E}$ We employ bge-large-zh (Chen et al., 2024), a dual-encoder retriever, to obtain the top-3 most semantically similar questions from the training set. The retrieval process calculates the cosine similarity between the embeddings of the user question $\mathcal{Q}$ and the training questions, where each retrieved example includes both the question and its ground-truth SQL query.

E.2 Prompt Templates

In this section we provide the exact prompts that have been used for each of the sub-modules in our framework.

Detailed Prompt Engineering Example

Database Schema

- **Table frpm:**
  - (**Column Name: CDSCode**; Sample Value: [01100170109835, 01100170112607]; **Column Description**: The CDSCode column represents the unique identifier for each school; Primary Key; Foreign Key references Table *schools*(CDSCode); 0% missing value)
  - (**Column Name: Free Meal Count (K-12)**; Sample Value: [565.0, 186.0]; **Column Description**: number of free meals provided; **eligible free rate = Free Meal Count / Enrollment**; 0.56% missing value)
  - ...
- **Table satscores:**
  - (**Column Name: cds**; Sample Value: [1100170000000, 1100170109835]; **Column Description**: refers to California Department Schools; Primary Key; Foreign Key references Table *schools*(CDSCode); 0% missing value)
  - ...
- **Table schools:**
  - (**Column Name: CDSCode**; Sample Value: [01100170000000...]; **Column Description**: unique identifier for each school in California; Primary Key; 0% missing value)

Few Shot

- **Question 1:** What is the average total donations received by Fresno County colleges?
  SQL 1: `SELECT SUM(T2.donation_optional_support + T2.donation_to_project) / COUNT(donationid) FROM projects AS T1 INNER JOIN donations AS T2 ON T1.projectid = T2.projectid WHERE T1.school_county = 'Fresno'`
- **Question 2:** How many schools in the West New York School District have the highest poverty level?
  SQL 2: `SELECT COUNT(poverty_level) FROM projects WHERE school_district = 'West New York School District' AND poverty_level = 'highest poverty'`

User Question

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

Figure 6: Organization of Elements in the Prompt

Templates for Generating $\texttt{SQL}_1$ Queries

Instruction:
You are an intelligent agent responsible for identifying the database tables.Your main tasks are:
- Generate SQL statements
- Return the results in json format, the format is: {
  "sql": "SQL statement that meets the user's question requirements"
  }

Input:
- Few-shot Examples ($\mathcal{E}$)
- Complete Database Schema ($\mathcal{S}$)
- User Question ($\mathcal{Q}$)

Note:
- In the generated SQL statement, table names and field names need to be enclosed in backticks, such as 'table_name', 'column_name'.

Figure 7: Templates for SQL Generation: $\texttt{SQL}_1$.

Templates for Generating $\texttt{SQL}_2$ Queries

Instruction:
You are an intelligent agent responsible for identifying the database tables.Your main tasks are:
- Generate SQL statements
- Return the results in json format, the format is: {
  "sql": "SQL statement that meets the user's question requirements"
  }

Input:
- Few-shot Examples ($\mathcal{E}$)
- Simplified Database Schema ($\hat{\mathcal{S}}'$ (with Schema Description))
- User Question ($\mathcal{Q}$)
- Augmented Context for User's Question H_{aug}

Note:
- In the generated SQL statement, table names and field names need to be enclosed in backticks, such as 'table_name', 'column_name'.

Figure 8: Templates for SQL Generation: $\texttt{SQL}_2$.

Template for Table and Column Selection Process

Instruction:
You are an intelligent agent responsible for identifying the database tables.Your main tasks are:
- Identify relevant tables
- Identify relevant columns
- Return the results in json format, the format is: {
  "tables": ["table1", "table2", ...],
  "columns":["table1.'column1'","table2.'column2'",...]
  }

Input:
- Database Schema ($\mathcal{S}/\hat{\mathcal{S}}'$)
- User Question ($\mathcal{Q}$)

Note:
- Ensure that all possible intermediate tables are considered, especially tables involving many-to-many relationships.
- Ensure that the output table list is unique and without duplicates.

Figure 9: Template for Table and Column Selection Using LLM to Identify Relevant Tables and Columns.

Template for Question Decomposition

Instruction:
You are an intelligent agent responsible for identifying the conditions in the user's question.Your main tasks are:
- Understand the user's question
- Identify conditions
- Return the results in json format, the format is: {
  "conditions": ["condition1", "condition2", ...]
  }

Input:
- User Question ($\mathcal{Q}$)

Note:
- Ensure that all conditions in the user's question are correctly extracted and understood.
- If the user's question contains complex conditions or multiple relationships, please make a reasonable judgment based on the context.

Figure 10: Template for Using LLM to Analyze Questions and Identify Possible Conditions for the WHERE Clause.

Template for Generation of SQL Keywords

Instruction:
You are an AI tasked with determining whether SQL statements need to use the following keywords:'DISTINCT', fuzzy matching, exact matching, 'INTERSECT', 'UNION', etc. Your main tasks are:

- Understand the user's question
- Determine keywords
- Return the results in json format, the format is: {
  "sql_keywords": ["keyword1", "keyword2", ...]
  }

Input:

- Simplified Database Schema ($\hat{\mathcal{S}}'$)
- User Question ($\mathcal{Q}$)

Note:

- Ensure that you understand the query requirements in user questions to accurately suggest SQL keywords and operations.
- Based on database structure and sample data, make reasonable judgments on whether specific SQL keywords or operations are needed.

Figure 11: Template for Using LLMs to Analyze Questions and Database Schemas for Extracting Relevant SQL Keywords (e.g., DISTINCT, GROUP BY).

Template for Binary Selection Strategy

Instruction:

- Synthesize a new SQL query, which can be the better query between $\texttt{SQL}_1$ and $\texttt{SQL}_2$, or a complete new query if both candidates are considered flawed.
- Return the results in json format, the format is: {
  "sql": "...",# your SQL query
  }

Input:

- Simplified Database Schema ($\hat{\mathcal{S}}'$)
- User Question ($\mathcal{Q}$)
- $\texttt{SQL}_1$ and Execution Results $\mathcal{R}_1$
- $\texttt{SQL}_2$ and Execution Results $\mathcal{R}_2$

Note:

- In the selected SQL statement, table names and field names need to be enclosed in backticks, such as 'table_name', 'column_name'.

Figure 12: Selection Template for Optimal SQL Statement Based on Execution Results of $\texttt{SQL}_1$ and $\texttt{SQL}_2$, Balancing Redundancy and Completeness.