

DSR-SQL: Dual-Schema Reasoning for Text-to-SQL and PI-MCE Optimized Schema Linking

Hu Lu Fanghao Li Dongqi Han Yao Zhang

Jiangsu University, Zhenjiang, China

luhu@ujs.edu.cn, 2232408005@stmail.ujs.edu.cn

2232308015@stmail.ujs.edu.cn, 2232408012@stmail.ujs.edu.cn

Abstract

Small open-source LLMs for Text-to-SQL face a trade-off between schema completeness and input noise. Pruned schemas provide compact contexts for SQL generation but are vulnerable to schema-linking errors, while full schemas preserve complete database information but may introduce substantial irrelevant context. We propose DSR-SQL, a dual-schema reasoning framework that generates SQL candidates from both pruned and full schemas. The pruned-schema path supports focused reasoning, whereas the full-schema path helps recover schema elements that may be missed during pruning. The generated candidates are refined with execution feedback and integrated through an execution-aware merger or a deterministic fallback selector. To improve the reliability of the pruned-schema path, we further introduce a Permutation-Invariant Minimum Cross-Entropy (PI-MCE) loss for table selection, which reduces the sequence-order bias caused by autoregressive training on unordered table sets. Experiments on the BIRD benchmark show that DSR-SQL achieves 67.41% execution accuracy on the test set using a 7B-parameter model. These results demonstrate the effectiveness of dual-schema reasoning, with PI-MCE providing complementary improvements in table selection for smaller open-source LLMs.

1 Introduction

Large language models (LLMs) have become a major direction in Text-to-SQL, which aims to translate natural language questions into executable SQL queries (Deng et al., 2022; Katsogiannis-Meimarakis and Koutrika, 2023). Closed-source models such as GPT-4 achieve strong performance, but their deployment in enterprise scenarios remains constrained by inference cost, privacy concerns, and controllability issues (Pourreza and Rafiei, 2024). This has motivated interest

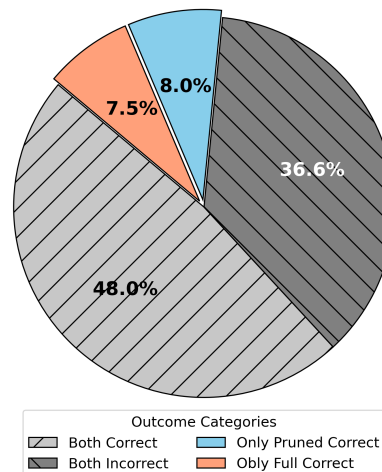


Figure 1: Performance distribution of Pruned and Full Schema strategies using a fine-tuned Qwen2.5-7B model on the BIRD dev set.

in adapting smaller open-source LLMs, such as 7B-parameter models, through task-specific fine-tuning (Pourreza and Rafiei, 2024; Li et al., 2024; Sheng and Xu, 2025).

A common approach to improving smaller LLMs for Text-to-SQL is schema pruning: selecting schema elements likely relevant to the question to reduce input length and remove irrelevant context (Pourreza and Rafiei, 2024; Wang et al., 2025; Lee et al., 2025). However, this strategy is vulnerable to schema-linking errors. Once relevant tables or columns are mistakenly filtered out, the generator loses access to information required for correct SQL construction. We refer to this irreversible information loss as the “Information Masking Effect” (Pourreza and Rafiei, 2024; Maamari et al., 2024). In contrast, using the full schema preserves complete database context but may introduce irrelevant information that overwhelms smaller LLMs. In a BIRD case (Appendix A.1), schema pruning omits the required yearmonth table, leading to an incorrect temporal condition. The full-schema path

recovers the missing table and generates the intended join.

We examine this trade-off on the BIRD dev set using a fine-tuned Qwen2.5-7B model. As shown in Figure 1, the Pruned and Full Schema strategies jointly solve 48.0% of examples, while 8.0% are solved only by Pruned Schema and 7.5% only by Full Schema. The two strategies are complementary: pruned schemas provide compact reasoning contexts, whereas full schemas can recover cases where schema linking misses important elements. If coordinated, their upper-bound accuracy would reach 63.4%.

We propose Dual-Schema Reasoning for Text-to-SQL (DSR-SQL), a framework that combines pruned-schema and full-schema reasoning for smaller LLMs. DSR-SQL contains two generation paths. The *Precision-First* path generates SQL from the pruned schema to exploit a compact context, while the *Recall-First* path uses the full schema to reduce the risk of missing relevant schema elements. Since the two paths may produce different SQL candidates, we introduce an execution-aware Merger module that selects the final output by considering both execution results and semantic consistency with the user question.

To improve the reliability of the Precision-First path, we address the order-sensitivity issue in table selection. Relevant tables form an unordered set, but standard autoregressive training treats the target table sequence as ordered and penalizes other valid permutations. We introduce the Permutation-Invariant Minimum Cross-Entropy (PI-MCE) loss, which treats the target order as a latent variable and optimizes the minimum cross-entropy over possible permutations. This encourages the model to select the correct tables rather than imitate an arbitrary order.

Experiments on the BIRD benchmark show that DSR-SQL achieves 67.41% execution accuracy (EX) on the test set using a 7B-parameter model. This demonstrates that dual-schema reasoning can improve smaller open-source LLMs for Text-to-SQL, especially when either pruned-schema or full-schema reasoning alone is insufficient. Our code is available in a GitHub repository¹.

2 Related Work

Early Text-to-SQL Models. Early natural language interfaces to databases used learned sym-

bolic parsers (Zelle and Mooney, 1996), while later neural approaches draw on general sequence-to-sequence architectures (Sutskever et al., 2014). Since database schemas contain complex structural relationships, many studies focus on schema encoding. Graph-based and relation-aware methods, such as SADGA (Cai et al., 2021), LGESQL (Cao et al., 2021), RAT-SQL (Wang et al., 2020), and RASAT (Qi et al., 2022), explicitly model schema structures, foreign-key relations, and question-schema links to improve SQL generation.

LLM-based Text-to-SQL. The emergence of LLMs has enabled Text-to-SQL through in-context learning and prompting (Rajkumar et al., 2022). However, real-world databases often contain large schemas that exceed or burden the model context, making schema linking essential (Dong et al., 2023). Existing methods use neural ranking models (Li et al., 2023a) or LLM-based filtering (Pourreza and Rafiei, 2023; Dong et al., 2023) to select relevant tables and columns before SQL generation. Prior work emphasizes the importance of moving beyond unrealistic “gold table” assumptions and explicitly modeling table selection in practical settings (Han et al., 2024). Nevertheless, hard pruning creates a recall ceiling: if critical schema elements are removed, downstream generation cannot recover them (Maamari et al., 2024).

C3 (Dong et al., 2023) applies Self-Consistency (SC) (Wang et al., 2023) to Text-to-SQL by aggregating multiple sampled outputs. Related inference-time methods include Chain-of-Thought reasoning (Wei et al., 2022), execution-guided refinement, and iterative self-correction (Talaie et al., 2024). However, standard SC relies on stochastic variations under the same prompt, which may not address systematic schema-linking errors. DSR-SQL differs by constructing two deterministic and structurally different reasoning paths based on pruned and full schemas. The final prediction is chosen by an execution-aware Merger, which is related to SQL generation and merge revision pipelines (Sheng and Xu, 2025) but focuses on coordinating complementary schema contexts.

Small LLM Adaptation and Order Sensitivity.

Cost and privacy constraints have driven interest in adapting smaller open-source LLMs through supervised fine-tuning (Li et al., 2024; Domínguez et al., 2024). Recent systems such as OmniSQL (Li et al., 2025) show that high-quality synthetic data can substantially improve small-model Text-to-SQL perfor-

¹<https://github.com/lifh1998/DSR-SQL>

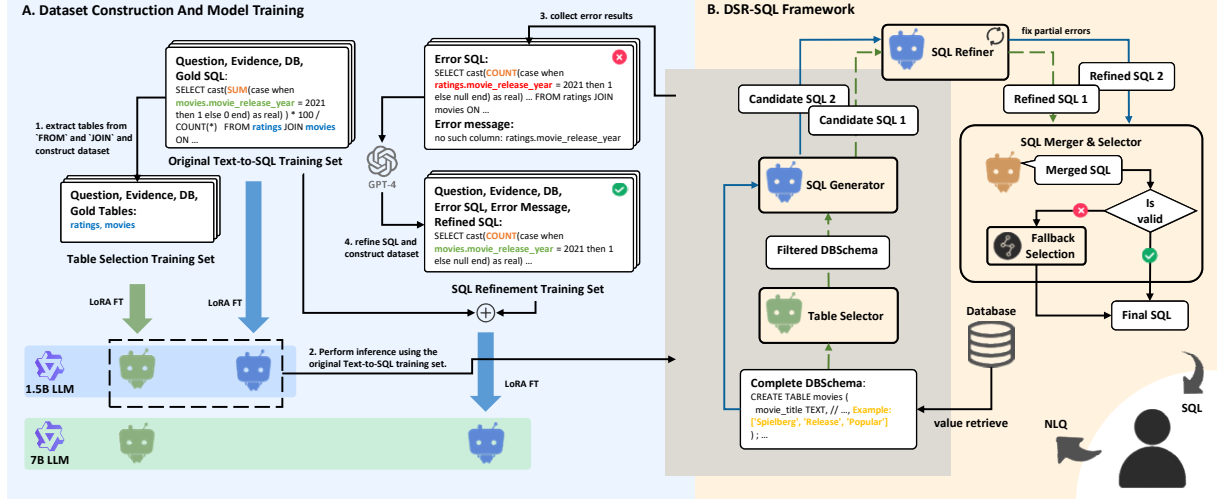


Figure 2: Part A details the methodology for constructing the training set used to train our specialized model. Part B outlines our DSR-SQL framework. Specifically, during the construction of the SQL Refinement training set, the fine-tuned 1.5B model is employed to perform inference and identify erroneous prediction results.

mance. However, fine-tuning autoregressive models for schema linking introduces order sensitivity: relevant tables form an unordered set, yet standard cross-entropy penalizes valid permutations that differ from the annotated order (Vinyals et al., 2016). Reinforcement learning methods such as GRPO optimize task-level rewards but require an additional training stage (Guo et al., 2025; Pourreza et al., 2025). PI-MCE instead addresses target-order sensitivity within supervised table-selection training.

3 Method

3.1 DSR-SQL: Dual-Schema Reasoning for Text-to-SQL

Figure 2 illustrates the DSR-SQL architecture. DSR-SQL balances information completeness and reasoning precision with two parallel paths. Given a question and a database schema, we first retrieve relevant values to anchor SQL clauses (detailed in Appendix A.2). The question then flows through two paths: Path 1 uses a compact schema subset to emphasize precision; Path 2 uses the full schema to ensure recall. The final SQL comes from an intelligent merger or a simple fallback selector.

3.1.1 Table Selector

Let the full schema be $\mathcal{S} = \{T_1, T_2, \dots, T_N\}$. The Table Selector outputs a compact, question-relevant subset $\mathcal{S}' \subseteq \mathcal{S}$:

$$\mathcal{S}' = f_{\text{sel}}(Q, \mathcal{S} \mid \theta_{\text{sel}}). \quad (1)$$

The required tables form an unordered set, not a sequence. Standard autoregressive training would

introduce order bias. To reduce this bias, we fine-tune the selector with a Permutation-Invariant Minimum Cross-Entropy (PI-MCE) loss (Section 3.3). This loss emphasizes semantic relevance without enforcing a fixed table order.

3.1.2 SQL Generator

The SQL Generator runs two parallel paths to produce candidate SQLs sql_1 and sql_2 . Both paths share the same autoregressive scheme but differ in the schema subset provided. For $i \in \{1, 2\}$:

$$P(sql_i \mid Q, \mathcal{S}_i) = \prod_{t=1}^{|sql_i|} P(y_{i,t} \mid y_{i,<t}, Q, \mathcal{S}_i; \theta_{\text{gen}}) \quad (2)$$

where $\mathcal{S}_1 = \mathcal{S}'$ denotes the pruned schema and $\mathcal{S}_2 = \mathcal{S}$ denotes the full schema.

Path 1 (**Precision-first**) limits the context to the most relevant schema elements, which reduces distraction and enables the model to reason precisely over the essential tables and columns. Path 2 (**Recall-first**) retains the full schema to ensure that no useful table or column is lost when the selector misses it, thereby protecting recall. The two paths are complementary: the former favors a compact and focused context, while the latter safeguards against schema-linking errors.

3.1.3 SQL Refiner

The SQL Refiner follows MAC-SQL (Wang et al., 2025) and uses execution feedback to correct SQL queries. It does not rely on intrinsic self-

correction (Pourreza and Rafiei, 2023) alone; the database executor provides reliable error messages.

Let $sql^{(0)}$ be an initial SQL candidate from Path i , where $i \in \{1, 2\}$. At iteration t , we run $sql^{(t)}$ against the database. If it produces an error (syntax error or runtime exception) or an empty result, we denote the corresponding execution feedback by $E^{(t)}$. The Refiner then produces a revised query:

$$sql^{(t+1)} = f_{\text{ref}}(Q, \mathcal{S}_i, sql^{(t)}, E^{(t)} \mid \theta_{\text{ref}}). \quad (3)$$

This loop stops when the query executes successfully and returns a non-empty result or when T_{max} rounds are reached. We set $T_{\text{max}} = 3$. Empty results are treated as heuristic feedback for refinement.

3.1.4 SQL Merger & Selector

The final module picks or combines the two refined candidates. We use the SQL Merge Revision model from CSC-SQL (Sheng and Xu, 2025) as the intelligent merger. It takes the question, the full schema, sql_1 , sql_2 , and their execution feedback r_1 and r_2 as input:

$$sql_{\text{mer}} = f_{\text{mer}}(Q, \mathcal{S}, sql_1, sql_2, r_1, r_2 \mid \theta_{\text{csc}}). \quad (4)$$

If the merger cannot output a valid SQL, DSR-SQL falls back to a deterministic selector. This selector prefers executable queries and, when both are executable, compares their execution signals. The full algorithm appears in Appendix A.4.

3.2 Training Set Construction and Fine-Tuning Details

Table Selection Training Set. The primary role of the Table Selector is to accurately identify all relevant tables from a full database schema based on a given natural language question. The construction of its training set began with established Text-to-SQL benchmarks, such as BIRD and Spider. For each sample, we statically parsed the provided Gold SQL to extract all table names from its FROM and JOIN clauses. This forms an inherently unordered ‘‘Gold Table Set’’, denoted as $\mathcal{T}_{\text{gold}} = \{t_1, t_2, \dots, t_k\}$.

These sets were then paired with the original input context (the Question and Full DB Schema) to create training samples formatted as $\mathcal{D}_{\text{sel}} = \{(X, \mathcal{T}_{\text{gold}})\}$. Because $\mathcal{T}_{\text{gold}}$ is unordered, standard autoregressive training makes the loss sensitive to the arbitrary order chosen for the target sequence.

We therefore optimize these samples with PI-MCE, as detailed in Section 3.3, which selects the lowest-loss valid permutation during training.

SQL Generation Training Set. For the fundamental SQL generation task, we utilized the standard training pairs provided by the benchmarks, formalized as $\mathcal{D}_{\text{gen}} = \{(X, sql_{\text{gold}})\}$. Unlike table selection, SQL queries possess a strict, deterministic syntactic structure; thus, standard autoregressive cross-entropy optimization is naturally suitable for this phase.

SQL Refinement Training Set. To construct a dataset that effectively teaches the model how to correct itself, we adopted a comprehensive ‘‘Generate-Execute-Refine’’ methodology to synthesize high-quality debugging trajectories. We first fine-tuned a preliminary model (based on Qwen2.5-1.5B) solely on $\mathcal{D}_{\text{gen}}$. This model generated predictions sql_{err} over the training set. We then executed these predictions and collected instances where the outcomes were problematic.

A naive approach to teaching refinement would directly pair the flawed query sql_{err} with the sql_{gold} as the learning target. However, sql_{gold} often differs significantly in logical structure and style from the model’s generated output, causing a severe distribution shift and making it an unsuitable target for learning specific, localized debugging steps.

To overcome this, we employed GPT-4 (OpenAI, 2023) as a ‘‘reasoning teacher’’ to generate minimal edit trajectories using sql_{gold} as a reference. We formalized the prompt to GPT-4 as a correction function:

$$sql_{\text{ref}} = \text{GPT-4}(X, sql_{\text{err}}, E, sql_{\text{gold}}). \quad (5)$$

GPT-4 was explicitly instructed to perform a minimal, logical correction: it had to fix the specific error identified by the environmental feedback E , while preserving the original abstract syntax tree (AST) structure of sql_{err} as much as possible. This yielded sql_{ref} , representing a step-by-step cognitive correction rather than a disjointed rewrite. The final refinement samples were formatted as $\mathcal{D}_{\text{ref}} = \{(X, sql_{\text{err}}, E) \rightarrow sql_{\text{ref}}\}$, and subsequently merged with the generation samples to create a unified multi-task dataset $\mathcal{D}_{\text{unified}} = \mathcal{D}_{\text{gen}} \cup \mathcal{D}_{\text{ref}}$.

Fine-Tuning Strategy. We fine-tune our two models separately using Low-Rank Adaptation

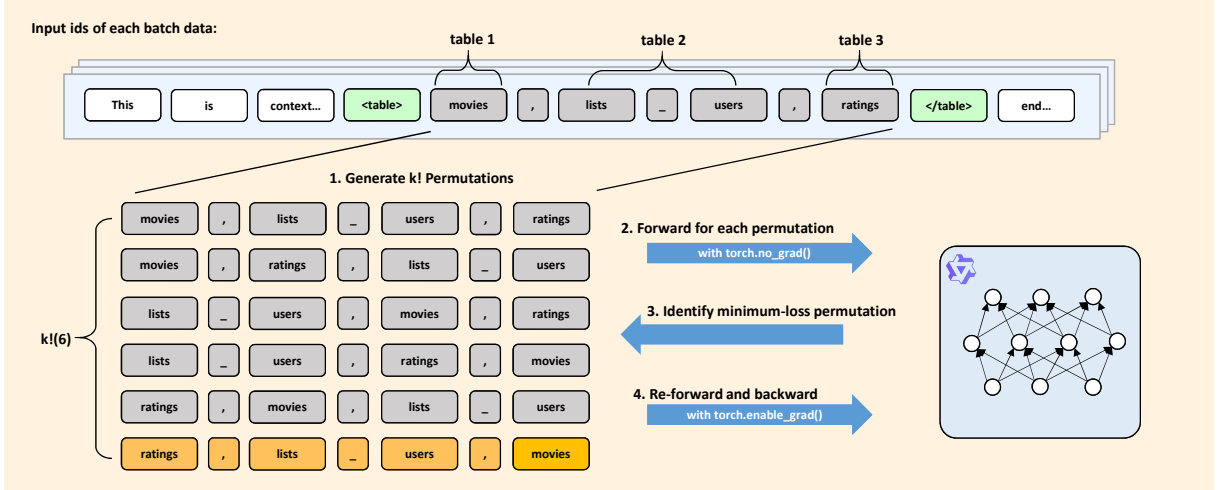


Figure 3: The implementation pipeline for PI-MCE. We first evaluate all permutations in a gradient-free pass, and then perform one gradient-enabled backward pass on the optimal alignment.

(LoRA) (Hu et al., 2022). The table selection model is trained with LoRA on its corresponding selection data, while the SQL generation and refinement model is trained with LoRA on the unified multi-task dataset $\mathcal{D}_{unified} = \mathcal{D}_{gen} \cup \mathcal{D}_{ref}$. All training hyperparameters and implementation details are provided in Appendix A.3.

3.3 PI-MCE: Permutation-Invariant Minimum Cross-Entropy

Standard sequence-to-sequence training optimizes a fixed target order. Given an input X and a target sequence $Y = (y_1, \dots, y_T)$, the token-averaged cross-entropy loss is:

$$\mathcal{L}_{CE}(Y|X; \theta) = -\frac{1}{T} \sum_{t=1}^T \log P(y_t | y_{<t}, X; \theta), \quad (6)$$

where θ denotes the model parameters.

However, table selection is an unordered set prediction task. For example, “Teacher Table, Student Table” and “Student Table, Teacher Table” represent the same table set, but standard CE penalizes one ordering if the other is used as the fixed label. This introduces artificial order bias and encourages the model to fit an arbitrary sequence order rather than the semantic relevance of tables.

To address this issue, we propose Permutation-Invariant Minimum Cross-Entropy (PI-MCE). Given a ground-truth table set T_{gold} with k tables, we enumerate all valid target sequences $\Pi_{seq}(T_{gold})$ induced by its $k!$ permutations. For each permuta-

tion $Y_\pi \in \Pi_{seq}(T_{gold})$, we compute:

$$\mathcal{L}_{CE}(Y_\pi|X; \theta) = -\frac{1}{|Y_\pi|} \sum_{t=1}^{|Y_\pi|} \log P(y_t^\pi | y_{<t}^\pi, X; \theta) \quad (7)$$

PI-MCE then selects the minimum loss over all permutations of T_{gold}^Y , the ground-truth table set corresponding to the target table-name sequence Y :

$$\mathcal{L}_{PI-MCE}(Y|X; \theta) = \min_{Y_\pi \in \Pi_{seq}(T_{gold}^Y)} \mathcal{L}_{CE}(Y_\pi|X; \theta) \quad (8)$$

Thus, the model is optimized toward the permutation that best matches its current predictive distribution, while all semantically equivalent table orders remain valid.

From a probabilistic perspective, PI-MCE treats the table order as a latent alignment variable between the unordered ground-truth set and the autoregressive decoding steps. The implementation follows a hard-EM-style alternating procedure (Neal and Hinton, 1998): in the E-step, all $k!$ permutations are evaluated without gradients to find the minimum-loss permutation Y_{opt} ; in the M-step, a single gradient-enabled pass is performed on Y_{opt} to update θ . This makes the selector focus on *what* to select rather than *how* to order it, reducing the fixed-order bias of standard CE for set prediction (Vinyals et al., 2016; Yu et al., 2017).

Practical Complexity. PI-MCE enumerates $k!$ target orders for k gold tables. Figure 4 shows the distribution after filtering execution-error examples. In BIRD and Spider, 96.21% and 95.37% of the examples have $k \leq 3$, respectively, requiring at most

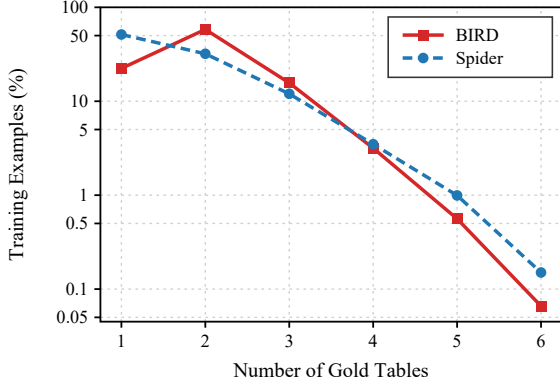


Figure 4: Distribution of gold-table-set sizes in the filtered BIRD and Spider training data. The analysis contains 9,098 BIRD examples and 8,656 Spider examples after excluding instances marked as execution errors. The vertical axis uses a logarithmic scale.

six orders. Across both datasets, $\mathbb{E}[k!] = 4.60$. We therefore estimate roughly $3\times$ the computation of standard CE; this is an analytical estimate, not a wall-clock measurement. The observed maximum is $k = 6$ (720 orders).

4 Experiments

4.1 Experimental Setup

Models. We mainly use Qwen2.5-7B (Hui et al., 2024) as the backbone for fair comparison with recent 7B-scale Text-to-SQL methods. To examine cross-backbone robustness at this model scale, we also evaluate DSR-SQL with DeepSeek-6.7B (Guo et al., 2024), Mistral-7B, and Gemma-7B.

Datasets. We evaluate on BIRD (Li et al., 2023b), Spider (Yu et al., 2018), Spider-SYN (Gan et al., 2021a), and Spider-DK (Gan et al., 2021b). BIRD is used as the primary benchmark, Spider evaluates cross-domain generalization, while Spider-SYN and Spider-DK assess robustness to lexical variation and domain knowledge. Dataset statistics are shown in Table 1. Schemas are serialized with DDL statements and sampled database content.

Dataset	Domains	Databases	Train	Dev	Test
BIRD (Li et al., 2023b)	37	95	9,428	1,534	1,789
Spider (Yu et al., 2018)	138	200	8,659 [†]	1,034	2,147

[†]The Spider training set combines 7,000 train_spider examples and 1,659 train_others examples.

Table 1: Detailed statistics for BIRD and Spider.

Metrics. We report Execution Accuracy (EX) as the main metric, which compares execution results

Model	LLM	#Cand.	Test	Dev
XIYAN-SQL (Liu et al., 2026)	GPT-4o	10	75.63	73.34
CSC-SQL (Sheng and Xu, 2025)	XiYanSQL-32B	64	73.67	71.33
CHASE-SQL (Pourreza et al., 2024)	Gemini-1.5-pro	21	73.0	73.01
Reasoning-SQL (Pourreza et al., 2025)	Qwen2.5-14B	21	72.78	72.29
CSC-SQL (Sheng and Xu, 2025)	Qwen2.5-7B	64	71.72	69.19
Omni-SQL (Li et al., 2025)	Qwen2.5-7B	8	–	66.10
Omni-SQL (Li et al., 2025)	Qwen2.5-7B	1	–	63.90
DSR-SQL (Ours)	Qwen2.5-7B	2	67.41	62.26
SFTCodeS (Li et al., 2024)	CodeS-15B	4	60.37	58.47
DTS-SQL (Pourreza and Rafiei, 2024)	DeepSeek-6.7B	1	60.31	55.8
MAC-SQL (Wang et al., 2025)	GPT-4	1	59.59	59.39
SFTCodeS (Li et al., 2024)	CodeS-7B	4	59.25	57.17
DAIL-SQL (Gao et al., 2024)	GPT-4	–	57.41	54.76
DIN-SQL (Pourreza and Rafiei, 2023)	GPT-4	1	55.90	50.72

Table 2: Execution accuracy (EX, %) on BIRD. #Cand. counts initial complete SQL candidates under reported or default settings, before deduplication and excluding subsequent refinement and merging; “–” in this column indicates an unverified count.

Model	LLM	#Cand.	Test	Dev
XIYAN-SQL (Liu et al., 2026)	GPT-4o	10	89.65	–
Omni-SQL (Li et al., 2025)	Qwen2.5-7B	8	88.9	–
CHASE-SQL (Pourreza et al., 2024)	Gemini-1.5-pro	21	87.6	–
Omni-SQL (Li et al., 2025)	Qwen2.5-7B	1	87.9	–
DAIL-SQL (Gao et al., 2024)	GPT-4	5	86.6	83.6
DSR-SQL (Ours)	Qwen2.5-7B	2	85.5	86.5
DIN-SQL (Pourreza and Rafiei, 2023)	GPT-4	1	85.3	82.8
DTS-SQL (Pourreza and Rafiei, 2024)	DeepSeek-6.7B	1	84.4	85.5
MAC-SQL (Wang et al., 2025)	GPT-4	1	82.8	86.75

Table 3: Reported performance (%) on Spider. #Cand. follows the definition in Table 2.

between predicted and gold SQLs. For table selection, we use Exact Accuracy, Filtered Accuracy, Precision, and Recall, as detailed in Section 4.5.1.

Implementation Details. Unless otherwise specified, inference uses greedy decoding with temperature 0. For the multi-candidate comparison in Table 4, candidates are sampled with temperature 0.8 and top- p 0.9. The SQL Generator and SQL Refiner share model weights.

4.2 Main Results

Table 2 and Table 3 compare our method with representative Text-to-SQL systems on the BIRD and Spider benchmarks. Using Qwen2.5-7B without Self-Consistency decoding, DSR-SQL achieves 67.41% EX accuracy on BIRD and 85.5% on Spider. On BIRD, this represents a 7.10-point gain over DTS-SQL (60.31%). On Spider, DSR-SQL reports 85.5%, compared with 84.4% reported by DTS-SQL, and approaches the reported scores of methods using larger models. These results establish a strong non-SC baseline with one candidate per path for 7B-scale models on BIRD and Spider. Several higher-ranked systems achieve stronger absolute scores. However, they rely on larger proprietary models (GPT-4o, Gemini-1.5-pro), larger

Method	SQL Generator	SC Setting	BIRD Dev EX
DSR-SQL (Ours)	CSC-SQL-7B	n=4(8)	69.75
CSC-SQL	CSC-SQL-7B	n=64	69.19
CSC-SQL	CSC-SQL-7B	n=8	67.06
DSR-SQL (Ours)	OmniSQL-7B	n=4(8)	66.04
DSR-SQL (Ours)	Qwen2.5-7B	n=4(8)	64.99
CSC-SQL	CSC-SQL-7B	n=4	64.91
DSR-SQL (Ours)	CSC-SQL-7B	n=1(2)	64.60

Table 4: Comparison with CSC-SQL under different Self-Consistency settings on BIRD Dev. For DSR-SQL, n denotes samples per path (total $2n$ candidates) with single-pass merging; for CSC-SQL, n denotes total candidates with SC-based merging.

open-source backbones (14B+, 32B), or larger initial SQL candidate pools. For example, CSC-SQL samples up to 64 SQL candidates. In contrast, the base DSR-SQL setting generates two initial SQL candidates, one per schema path.

To compare performance under different candidate budgets, Table 4 compares DSR-SQL against brute-force SC scaling using CSC-SQL-7B as the generator. With one sample per path, DSR-SQL reaches 64.60%. This nearly matches CSC-SQL with four candidates (64.91%). With four samples per path, DSR-SQL achieves 69.75%. This surpasses CSC-SQL with 8 candidates (67.06%) and even its heavily scaled 64-candidate setting (69.19%). These results suggest that structured dual-path exploration offers benefits beyond simply increasing the number of sampled SQL candidates. With 8 total candidates (4 per path), DSR-SQL outperforms CSC-SQL with 64 candidates. This uses $8\times$ fewer generated candidates than CSC-SQL’s 64-candidate setting while maintaining or improving accuracy. The dual-path design enables complementary schema coverage. Path1 reduces noise through precision-focused pruning. Path2 compensates for information masking by retaining broader context.

These results suggest that DSR-SQL can complement recent training-side advancements. OmniSQL benefits from large-scale synthetic data augmentation. CSC-SQL further leverages GRPO-style optimization. Combining DSR-SQL with stronger data augmentation or reinforcement-learning-based training could further improve performance. For instance, applying DSR-SQL to the CSC-SQL-7B generator (which already uses GRPO) yields 69.75% with only 8 candidates. This result suggests that inference-time structure and training-time optimization can be combined.

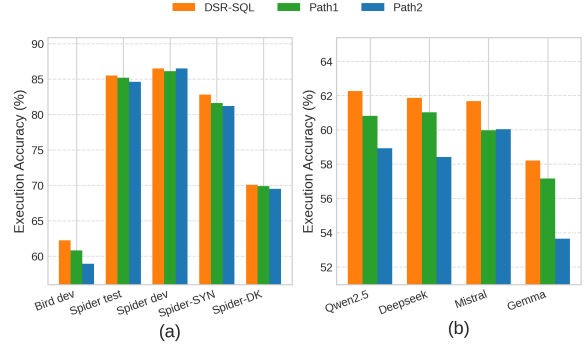


Figure 5: Generalization analysis of the DSR-SQL framework. (a) Execution accuracy across five datasets with varying complexity levels using Qwen2.5-7B as the base model. (b) Performance across four evaluated ~ 7 B open-source language models on the BIRD dev dataset.

4.3 Generalization Analysis

We evaluate DSR-SQL across multiple benchmarks and open-source LLMs with approximately 7B parameters to assess its generalization across the evaluated settings. Figure 5 shows that DSR-SQL outperforms both single-path variants in most of these settings. Compared with Path1, DSR-SQL achieves relative improvements of 2.37% on BIRD dev, 0.35% on Spider test, and 0.46% on Spider dev. These results suggest that dual-schema reasoning may offer greater benefits on more challenging benchmarks such as BIRD. Furthermore, DSR-SQL consistently outperforms both single-path variants across the four evaluated base models on BIRD dev. For instance, with Gemma-7B, DSR-SQL achieves 58.21% accuracy, compared with 53.65% for Path2, representing a relative improvement of 8.50%. Representative error recovery cases are provided in Appendix A.6.

4.4 Ablation Study

We systematically evaluate each component on the BIRD dev set using Qwen2.5-7B. Table 5 shows the results of progressively removing key modules. We also report results with gold tables to assess errors attributable to table-level schema linking.

Without the SQL Merger, no fusion is performed and DSR-SQL defaults to Path1, reducing accuracy from 62.26% to 60.82%. The 1.44-point gap therefore reflects the overall benefit of integrating the two paths, rather than the merger alone. Removing the Table Selector instead yields Path2 only and lowers accuracy to 58.93%, showing that Path1 is stronger in this setting. Without the SQL Refiner,

Setting	DSR-SQL	Path1	Path2
base	62.26	60.82	58.93
w/o sql refiner	59.65 (−2.61)	56.65 (−4.17)	55.61 (−3.32)
w/o PI-MCE	61.67 (−0.59)	59.97 (−0.85)	-
w/ gold tables	64.08 (+1.82)	62.91 (+2.09)	-

Table 5: Ablation study on the BIRD dev set using Qwen2.5-7B (Execution Accuracy, %). w/ gold tables replaces predicted tables with gold tables to assess table-level schema-linking errors. Changes relative to the base method are shown in parentheses.

DSR-SQL drops by 2.61 points, with Path1 suffering a larger loss (4.17 points) than Path2 (3.32 points). This asymmetry suggests that the pruned-schema path may produce more localized errors, such as wrong column names within correct tables, which execution feedback can more readily repair.

To assess the Refiner’s conditional inference overhead, we further measure its usage on the development sets. As shown in Table 6, refinement is not invoked for 82.53% of BIRD examples and 90.33% of Spider examples. Across both paths, the average numbers of Refiner calls are only 0.44 and 0.38 per question, respectively. These results indicate that refinement is concentrated on a relatively small subset of examples while providing the accuracy gains reported above.

Dataset	Path1	Path2	Combined	No Ref.
BIRD	0.20	0.24	0.44	82.53%
Spider	0.20	0.18	0.38	90.33%

Table 6: Refiner usage on the BIRD and Spider development sets. Path1, Path2, and Combined report the average numbers of Refiner calls per question. No Ref. denotes the proportion of examples requiring no refinement on either path.

Using gold tables improves DSR-SQL from 62.26% to 64.08%, a 1.82-point gain, showing that table-level schema linking remains an important but not exclusive source of error. Replacing PI-MCE with standard CE reduces DSR-SQL to 61.67%, indicating a modest 0.59-point end-to-end benefit. Detailed analysis is provided in Section 4.5.1. Cross-model and cross-dataset ablations in Appendix A.5 show consistent gains across the evaluated backbones and datasets.

4.5 Discussion

4.5.1 Why PI-MCE Improves Table Selection

We evaluate the Table Selector using four metrics. Exact Accuracy requires the predicted table set

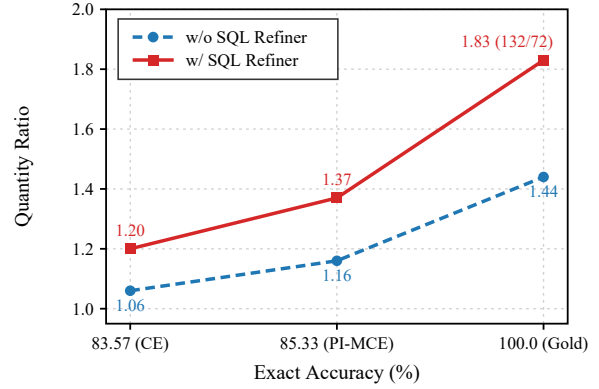


Figure 6: Evolution of the Quantity Ratio $QR = \frac{|\mathcal{S}_1 \setminus \mathcal{S}_2|}{|\mathcal{S}_2 \setminus \mathcal{S}_1|}$ across the three table-selection settings. The gap between curves reflects the effect of the SQL Refiner, while the finite ratio with gold tables confirms the complementarity of the two paths.

to exactly match the ground truth. Precision and Recall measure correctness and completeness. Filtered Accuracy counts a prediction as correct if it contains all ground-truth tables, allowing redundant tables but penalizing omissions. Since missing required tables causes severe information masking, Recall is the most critical metric.

Loss	Exact Accuracy	Filtered Accuracy	Precision	Recall
CE	0.8357	0.867	0.9628	0.9392
PI-MCE	0.8533	0.882	0.9659	0.9468

Table 7: Table selection performance on the BIRD development set. Qwen2.5-7B is fine-tuned under identical settings with CE and PI-MCE losses.

Table 7 shows that PI-MCE improves all four table-selection metrics. Exact Accuracy increases from 83.57% to 85.33%, and Recall from 93.92% to 94.68%. These improvements translate into a modest 0.59-point end-to-end gain in Table 5, indicating that PI-MCE primarily benefits table selection.

4.5.2 Understanding Path Complementarity

We examine why both paths are necessary by analyzing their complementary error patterns. We define the Quantity Ratio as $QR = \frac{|\mathcal{S}_1 \setminus \mathcal{S}_2|}{|\mathcal{S}_2 \setminus \mathcal{S}_1|}$, where $\mathcal{S}_1$ and $\mathcal{S}_2$ are the sets of questions correctly answered by Path1 and Path2. Figure 6 shows this ratio under three table-selection settings: 83.57% (CE), 85.33% (PI-MCE), and 100% (gold tables). With the SQL Refiner, the ratio increases from 1.20 to 1.83 across all three settings, indicating it

benefits Path1 more substantially. This is because Path1 operates on a pruned schema and tends to produce syntactically correct SQL with localized errors, such as wrong column names within the correct table. These errors are easier to fix when the refiner has execution feedback. Path2 works with the full schema and may generate queries with more diffuse errors caused by irrelevant context, which are harder to repair.

With gold tables and the standard SQL Refiner, 72 questions are solved only by Path2 while 132 are solved only by Path1. This persistent asymmetry suggests that seemingly redundant schema information may provide implicit reasoning support for LLMs. The full schema could offer additional context cues such as foreign key relationships between tables, structural patterns in column naming conventions, or the overall database topology that helps the model infer correct query logic even when those elements are not directly referenced in the final SQL. The finite ratio with gold tables (1.83) confirms that neither path dominates, supporting the balance between noise reduction (Path1) and contextual reasoning (Path2).

5 Conclusion

In conclusion, we propose DSR-SQL, a dual-path framework that improves Text-to-SQL performance across the evaluated small open-source language models. The framework reasons over both full and pruned schemas in parallel, balancing the high recall of comprehensive information with the high precision of a focused view. This design mitigates the information masking problem inherent in single-path methods. We also introduce PI-MCE, which improves order-robust table selection and provides a modest end-to-end gain. On BIRD, our model achieves 67.41% execution accuracy, demonstrating competitive performance among the evaluated open-source systems and supporting the effectiveness of the dual-path design.

Limitations

Despite its strong performance, DSR-SQL has several limitations that suggest directions for future work.

First, the main DSR-SQL contribution is its inference-time dual-path design. Integrating advanced training techniques such as large-scale synthetic data augmentation and GRPO could further improve performance. Second, PI-MCE uses ex-

act enumeration and has factorial complexity in the number of ground-truth tables. Its cost grows quickly for many-table joins, and scalable approximations are needed for larger target sets. Finally, our dual-schema strategy uses complete and pruned representations. Exploring finer granularities, such as column-level selection, could enhance context robustness.

Ethics Statement

We use publicly available datasets (BIRD and Spider) under the CC BY-SA 4.0 license for research purposes, consistent with their intended use. No human subjects or personally identifiable information are involved. Code and annotations will be released under compatible licenses.

References

- 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.
- Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu. 2021. Lgesql: line graph enhanced text-to-sql model with mixed local and non-local relations. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2541–2555.
- Naihao Deng, Yulong Chen, and Yue Zhang. 2022. Recent advances in text-to-sql: A survey of what we have and what we expect. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 2166–2187.
- José Manuel Domínguez, Benjamín Errázuriz, and Patriocio Daher. 2024. Blar-sql: Faster, stronger, smaller nl2sql. *arXiv preprint arXiv:2401.02997*.
- Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Lu Chen, Jinshu Lin, and Dongfang Lou. 2023. C3: Zero-shot text-to-sql with chatgpt. *arXiv preprint arXiv:2307.07306*.
- Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R Woodward, Jinxia Xie, and Pengsheng Huang. 2021a. Towards robustness of text-to-sql models against synonym substitution. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2505–2515.
- Yujian Gan, Xinyun Chen, and Matthew Purver. 2021b. Exploring underexplored limitations of cross-domain text-to-sql generalization. In *Proceedings of the 2021*

- Conference on Empirical Methods in Natural Language Processing*, pages 8926–8931.
- Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. [Text-to-sql empowered by large language models: A benchmark evaluation](#). *Proc. VLDB Endow.*, 17(5):1132–1145.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, and 175 others. 2025. [DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning](#). *Nature*, 645:633–638.
- Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. Deepseek-coder: When the large language model meets programming—the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.
- Mirae Han, Seongsik Park, Seulgi Kim, and Harksoo Kim. 2024. [Bridging the gap between text-to-sql research and real-world applications: A unified all-in-one framework for text-to-sql](#). *Knowledge-Based Systems*, 306:112697.
- Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.
- Binyuan Hui, Jian Yang, Zeyu Cui, Jiayi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, and 5 others. 2024. Qwen2.5-Coder technical report. *arXiv preprint arXiv:2409.12186*.
- 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. 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.
- Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tiejing Zhang, Jianjun Chen, Rui Shi, Hong Chen, and Cuiping Li. 2025. [Omnisql: Synthesizing high-quality text-to-sql data at scale](#). *Proceedings of the VLDB Endowment*, 18(11):4695–4709.
- Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023a. [Resdsq: 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. 2024. [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, Jiayi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023b. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36:42330–42357.
- Yifu Liu, Yin Zhu, Yingqi Gao, Zhiling Luo, Xiaoxia Li, Xiaorong Shi, Yuntao Hong, Jinyang Gao, Yu Li, Bolin Ding, and Jingren Zhou. 2026. [Xiyan-sql: A novel multi-generator framework for text-to-sql](#). *IEEE Transactions on Knowledge and Data Engineering*, 38(4):2474–2487.
- 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. In *NeurIPS 2024 Third Table Representation Learning Workshop*.
- Radford M Neal and Geoffrey E Hinton. 1998. [A view of the em algorithm that justifies incremental, sparse, and other variants](#). In *Learning in graphical models*, pages 355–368. Springer.
- OpenAI. 2023. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*.
- Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O Arik. 2024. Chase-sql: Multi-path reasoning and preference optimized candidate selection in text-to-sql. *arXiv preprint arXiv:2410.01943*.
- Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: decomposed in-context learning of text-to-sql with self-correction. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, 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.
- Mohammadreza Pourreza, Shayan Talaei, Ruoxi Sun, Xingchen Wan, Hailong Li, Azalia Mirhoseini, Amin Saberi, and Sercan Ö. Arik. 2025. Reasoning-sql: Reinforcement learning with sql tailored partial rewards for reasoning-enhanced text-to-sql. In *Proceedings of the Second Conference on Language Modeling*.

- Jiexing Qi, Jingyao Tang, Ziwei He, Xiangpeng Wan, Yu Cheng, Chenghu Zhou, Xinbing Wang, Quanshi Zhang, and Zhouhan Lin. 2022. Rasat: Integrating relational structures into pretrained seq2seq model for text-to-sql. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3215–3229.
- Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. 2022. Evaluating the text-to-sql capabilities of large language models. *arXiv preprint arXiv:2204.00498*.
- Lei Sheng and Shuai-Shuai Xu. 2025. Csc-sql: Corrective self-consistency in text-to-sql via reinforcement learning. *arXiv preprint arXiv:2505.13271*.
- Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27.
- 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*.
- Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. 2016. Order matters: Sequence to sequence for sets. In *International Conference on Learning Representations*.
- Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578.
- Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2025. Mac-sql: A multi-agent collaborative framework for text-to-sql. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 540–557.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.
- Dong Yu, Morten Kolbæk, Zheng-Hua Tan, and Jesper Jensen. 2017. Permutation invariant training of deep models for speaker-independent multi-talker speech separation. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 241–245. IEEE.
- Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 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.
- 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.

A Appendix

A.1 Motivating Example

Figure 7 presents a motivating example that illustrates the limitation of relying solely on a pruned schema and the potential benefit of dual-schema reasoning. Given a question about transactions within a specific time period, the schema linking module fails to select the critical yearmonth table. When the model is restricted to the incomplete pruned schema (Figure 7a), it attempts to infer the missing temporal relation from other available columns, resulting in an incorrect query. This example demonstrates that once necessary schema elements are removed before generation, the downstream model may not be able to recover them. By contrast, if an additional reasoning path over the full schema is retained (Figure 7b), the model has an opportunity to identify the omitted table and generate the intended relational structure. This case motivates our dual-schema reasoning approach, which combines both pruned-schema and full-schema paths to leverage their complementary strengths.

A.2 Value Retrieval Details

Let Q denote the retrieval context, obtained by concatenating the input question with external evidence when available. Let $\mathcal{D}$ denote the set of textual cell values in the target database. For efficiency, we build a Lucene inverted index over distinct non-null textual values for each table and column, excluding numeric values and overly long strings. In our implementation, only textual values whose lengths are no greater than 40 are indexed.

Given Q , we construct a set of lexical search queries $K = \{k_1, k_2, \dots, k_m\}$ by extracting n -grams from Q , where $1 \leq n \leq 8$, and also include the full retrieval context as an additional query. For each query $k_i \in K$, we perform BM25 search over

NL Question: "Please list the countries of the gas stations with transactions taken place in June, 2013."

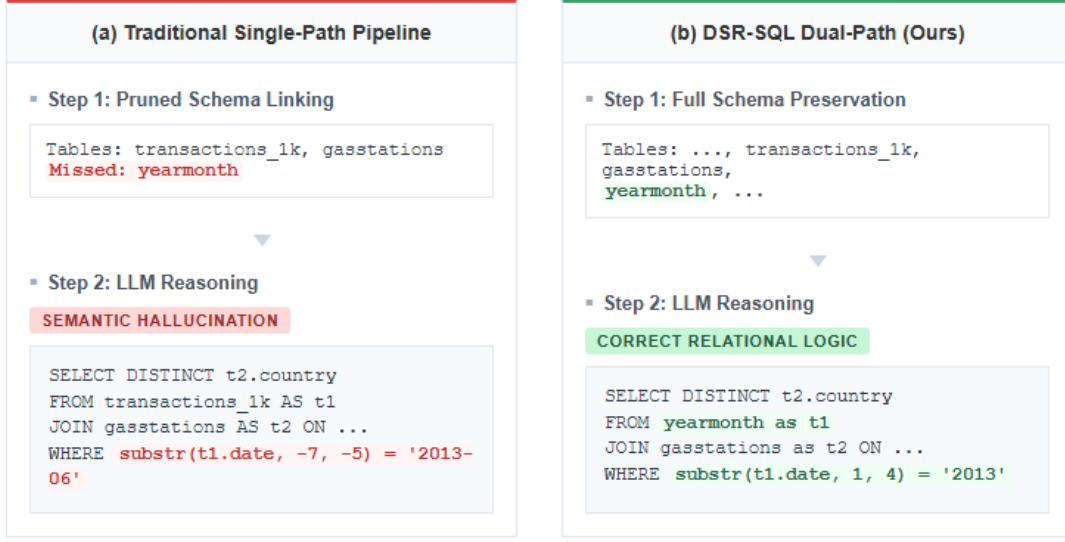


Figure 7: A motivating example illustrating the limitation of a single-schema pipeline and the potential benefit of dual-schema reasoning. (a) When relying only on an imperfect pruned schema, the model misses the critical yearmonth table and generates an incorrect workaround. (b) With an additional full-schema reasoning path, the model can recover the missed table and construct the correct relational logic.

the Lucene index and retrieve the top candidates:

$$C_i = \text{Top}_\rho^{\text{BM}^{25}}(k_i, \mathcal{D}), \quad (9)$$

where $\rho = 10$ in our implementation. The preliminary candidate pool is obtained by merging and deduplicating the retrieved results from all lexical queries:

$$C = \bigcup_{i=1}^m C_i. \quad (10)$$

To reduce noisy matches, we apply a substring-based verification step. For each candidate value $v \in C$, we compute a normalized longest-substring matching score between v and Q :

$$\mathcal{S}(v, Q) = \frac{\max_{s \in \text{Substr}(v)} \{|s| \mid s \text{ occurs in } Q\}}{|v|}, \quad (11)$$

where $\text{Substr}(v)$ denotes the set of contiguous substrings of v . This score measures the maximum proportion of the database value that can be continuously matched in the question context. We retain only candidates whose scores exceed a predefined threshold τ :

$$C_{\text{rel}} = \{v \in C \mid \mathcal{S}(v, Q) > \tau\}, \quad (12)$$

where $\tau = 0.85$ in our implementation.

Finally, the retained candidates are globally ranked according to their matching scores, value lengths, and retrieval order. We select the top κ values as the final retrieved value set:

$$V_{\text{ret}} = \text{Top}_\kappa(C_{\text{rel}}; \mathcal{S}(v, Q), |v|), \quad (13)$$

where $\kappa = 20$.

A.3 LoRA Fine-Tuning Details

Models and Training Data We perform LoRA-based fine-tuning separately for two 7B-parameter models: (1) the Table Selector, trained on $\mathcal{D}_{\text{sel}}$; (2) the SQL Generator-Refiner, trained on the unified multi-task dataset $\mathcal{D}_{\text{unified}} = \mathcal{D}_{\text{gen}} \cup \mathcal{D}_{\text{ref}}$.

Optimization and Precision For both models, we adopt LoRA (Hu et al., 2022) with BF16 mixed precision. Training is conducted for 3 epochs with batch size 16 and maximum sequence length 8,192 tokens.

Optimization uses learning rate $5e-5$, together with a cosine learning rate scheduler and a 1% warmup ratio (warmup_ratio = 0.01). Weight decay is set to 0.01.

LoRA Hyperparameters Key LoRA hyperparameters are set as follows: rank $r = 128$, scaling factor $\alpha = 64$, and dropout rate 0.1.

Compute Resources The LoRA fine-tuning for the SQL Generator-Refiner is completed using a single NVIDIA A800 (80GB) GPU. The overall wall-clock time is approximately 24 hours for the Qwen2.5-7B setting.

A.4 Fallback Selection Algorithm

Algorithm 1 Fallback SQL Selection Logic

▷ Helper function for executing SQL

```

1: function EXECUTE(sql)
2:   returns (success, result, time)
3: end function

4: Input: sqlfull, sqlfiltered
5: Output: sqlfinal
6: (successfull, resultfull, timefull) ← Execute(sqlfull)
7: (successfiltered, resultfiltered, timefiltered) ← Execute(sqlfiltered)
8: if successfull and not successfiltered then
9:   return sqlfull
10: else if not successfull and successfiltered then
11:   return sqlfiltered
12: else if successfull and successfiltered then
13:   if resultfull ≡ resultfiltered then
14:     return timefiltered < timefull ? sqlfiltered : sqlfull
15:   else
16:     return sqlfiltered
17:   end if
18: else
19:   return sqlfiltered
20: end if

```

When the intelligent merger fails to produce a valid result, our system employs a deterministic fallback strategy formalized in Algorithm 1. The algorithm prioritizes executable queries over non-executable ones. When both candidates are valid, it analyzes their execution metrics. If the results are identical, it selects the query with the shorter execution time. If results differ, it conservatively defaults to sql_1 (the precision-first path), as its generation context (the pruned schema S') is structurally more constrained and empirically less prone to hallucinations.

A.5 Cross-Model and Cross-Dataset Ablation

To assess the cross-backbone robustness of our architectural choices, we analyze progressive ablations across four base models in Table 8. Instead of

merely looking at absolute scores, focusing on the performance degradation (Δ) reveals how different models rely on specific components.

The results show that the full DSR-SQL framework achieves the best performance across the evaluated settings. However, the Table Selector exhibits model-dependent variability: while its removal causes notable drops for some models (*e.g.*, a 3.52-point drop for Gemma-7B on BIRD), Path2 (without the selector) occasionally outperforms Path1 in specific settings (*e.g.*, +0.7 points with DeepSeek-Coder on Spider test, and +0.07 points with Mistral on BIRD).

This variability illustrates a trade-off in schema linking: the Table Selector often improves precision but may exclude necessary context in ambiguous cases. Combining Path1’s precision with Path2’s completeness allows the Merger to adapt to the different error patterns observed across the evaluated backbones.

A.6 Case Study

To illustrate the behavior of our dual-path reasoning framework, we provide a qualitative analysis of two error scenarios caused by imperfect schema linking: Missing-Table error and Wrong-Table error. As shown in Figure 8, relying solely on a pruned schema can trigger the information masking effect, which DSR-SQL can mitigate through its full-schema path.

Case 1: The Missing-Table Scenario In the first case (Figure 8, top), the question asks for transaction records in June 2013. The schema linker fails to retrieve the critical yearmonth table. Forced to reason over this incomplete state, the pruned-schema path attempts a spurious semantic patch (refined_sql_1) by applying string matching (substr) directly on the unrelated transactions_1k.date column. This structure violates the intended relational design and fails. Conversely, the full-schema path has access to the global schema, allowing the LLM to successfully discover the omitted yearmonth table, formulate the correct JOIN condition, and accurately capture the temporal semantics. The DSR-SQL node evaluates both candidates and correctly routes the output to this semantically complete query (selected_sql).

Case 2: The Wrong-Table Scenario In the second case (Figure 8, bottom), the question queries a patient’s diagnosis. Here, the schema linker

Method (Progressive Ablation)	BIRD	SPIDER		SPIDER-SYN	SPIDER-DK
	dev	test	dev		
<i>Qwen2.5-Coder-7B-Instruct</i>					
DSR-SQL (Full)	62.26	85.5	86.5	82.8	70.1
→ w/o Merger (<i>reduces to Path1</i>)	60.82 (-1.44)	85.2 (-0.3)	86.1 (-0.4)	81.6 (-1.2)	69.9 (-0.2)
→ w/o Selector (<i>reduces to Path2</i>)	58.93 (-1.89)	84.6 (-0.6)	86.5 (+0.4)	81.2 (-0.4)	69.5 (-0.4)
→ w/o Refiner (<i>Baseline SFT</i>)	55.61 (-3.32)	83.0 (-1.6)	85.4 (-1.1)	78.6 (-2.6)	68.6 (-0.9)
<i>DeepSeek-Coder-6.7B-Instruct</i>					
DSR-SQL (Full)	61.86	85.1	85.1	80.2	—
→ w/o Merger (<i>reduces to Path1</i>)	61.02 (-0.84)	84.4 (-0.7)	85.0 (-0.1)	78.9 (-1.3)	—
→ w/o Selector (<i>reduces to Path2</i>)	58.41 (-2.61)	85.1 (+0.7)	82.9 (-2.1)	77.0 (-1.9)	—
→ w/o Refiner (<i>Baseline SFT</i>)	55.87 (-2.54)	83.9 (-1.2)	82.6 (-0.3)	76.2 (-0.8)	—
<i>Mistral-7B-Instruct</i>					
DSR-SQL (Full)	61.67	85.7	86.0	82.1	—
→ w/o Merger (<i>reduces to Path1</i>)	59.97 (-1.70)	85.3 (-0.4)	84.9 (-1.1)	80.6 (-1.5)	—
→ w/o Selector (<i>reduces to Path2</i>)	60.04 (+0.07)	83.5 (-1.8)	84.6 (-0.3)	80.9 (+0.3)	—
→ w/o Refiner (<i>Baseline SFT</i>)	58.21 (-1.83)	82.2 (-1.3)	83.7 (-0.9)	79.8 (-1.1)	—
<i>Gemma-7B</i>					
DSR-SQL (Full)	58.21	—	—	—	—
→ w/o Merger (<i>reduces to Path1</i>)	57.17 (-1.04)	—	—	—	—
→ w/o Selector (<i>reduces to Path2</i>)	53.65 (-3.52)	—	—	—	—
→ w/o Refiner (<i>Baseline SFT</i>)	51.50 (-2.15)	—	—	—	—

Table 8: Cross-model and cross-dataset ablation of the DSR-SQL framework (Execution Accuracy, EX). The values in parentheses represent the progressive performance drop (Δ) relative to the configuration in the row immediately above.

exhibits retrieval hallucination, mistakenly injecting the examination table instead of the correct patient table. Consequently, the pruned-schema path builds a fundamentally flawed query grounded in incorrect sources. Without a dual-path mechanism, this upstream error would be fatal. However, the full-schema path evaluates the entire relational structure natively, correctly identifying that diagnosis information resides in the patient table. The execution feedback mechanism detects the structural soundness of the full-schema query and gracefully discards the hallucinated generation.

These cases illustrate the core motivation of DSR-SQL: schema pruning can reduce noise but may remove necessary context, while the full-schema path can help recover such information. The dual-path design therefore provides a practical way to balance structural precision and contextual completeness.

Case 1: Missing-Table Error <i>"Please list the countries of the gas stations with transactions taken place in June, 2013."</i>	Path 1 (Pruned Schema): Missed 'yearmonth' <pre>SELECT DISTINCT t2.country FROM transactions_1k AS t1 JOIN gasstations AS t2 ON t1.customerid = t2.gasstationid WHERE substr(t1.`date`, -7, -5) = '2013-06'</pre> Path 2 (Full Schema): Recovered 'yearmonth' <pre>SELECT DISTINCT t2.country FROM yearmonth as t1 JOIN gasstations as t2 ON t1.gasstationid = t2.gasstationid WHERE substr(t1.date, 1, 4) = '2013' AND ...</pre>	MERGE OR SELECT ✓ Path 2 Output
Case 2: Wrong-Table Error <i>"What is the disease patient '30609' diagnosed with. List all the date of laboratory tests done."</i>	Path 1 (Pruned Schema): Hallucinated 'examination' <pre>SELECT DISTINCT t1.diagnosis, t2.date FROM examination AS t1 JOIN laboratory AS t2 ON t1.id = t2.id ...</pre> Path 2 (Full Schema): Corrected to 'patient' <pre>SELECT t2.diagnosis, t1.date FROM laboratory as t1 JOIN patient as t2 ON t1.id = t2.id ...</pre>	MERGE OR SELECT ✓ Path 2 Output

Figure 8: Qualitative comparison of SQL generation under single-path (Pruned Schema) and dual-path (DSR-SQL) settings. Top (Case 1): The pruned path suffers from a missing-table error, forcing the model into semantic hallucination; the full-schema path successfully recovers the yearmonth table. Bottom (Case 2): The pruned path hallucinates a wrong table (examination); the full-schema path correctly grounds the logic in the patient table. In both cases, DSR-SQL’s merge mechanism successfully selects the semantically complete query.