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

Anonymous ACL submission

Abstract

Small open-source LLMs for Text-to-SQL face a critical dilemma: providing the full schema introduces overwhelming noise that distracts the model, while pruning the schema risks irreversible information loss due to the masking effect from erroneous linking. To resolve this trade-off systematically, we propose DSR-SQL, a dual-path reasoning framework. It concurrently generates candidate SQLs from both full and pruned schemas to capture structural precision and semantic completeness, followed by an intelligent module that merges or selects the optimal query based on execution feedback, effectively balancing recall and precision. Furthermore, DSR-SQL introduces a Permutation-Invariant Minimal Cross-Entropy (PI-MCE) loss to resolve training bias caused by the disorder of table selection outputs, significantly improving schema linking accuracy. Extensive experiments show that DSR-SQL significantly outperforms comparable baselines and some GPT-4-based methods, demonstrating that refined architectural design combined with targeted fine-tuning enables small LLMs to rival proprietary giants, facilitating low-cost, efficient, and privacy-preserving natural language interfaces.

1 Introduction

In recent years, large language models (LLMs) have emerged as a new paradigm driving advances in Text-to-SQL, aiming to automatically convert natural language questions into executable SQL queries (Deng et al., 2022; Katsogiannis-Meimarakis and Koutrika, 2023). Although closed-source large models such as GPT-4 demonstrate outstanding performance, their high cost and data privacy risks limit widespread adoption in enterprise environments (Pourreza and Rafiei, 2024). Consequently, research has shifted toward fine-tuning more cost-effective and privacy-friendly open-source smaller LLMs (*e.g.*, 7B-parameter models), with the goal of achieving or even surpassing the performance of closed models on specific tasks—an increasingly promising direction (Pourreza and Rafiei, 2024; Li et al., 2024; Sheng and Xu, 2025).

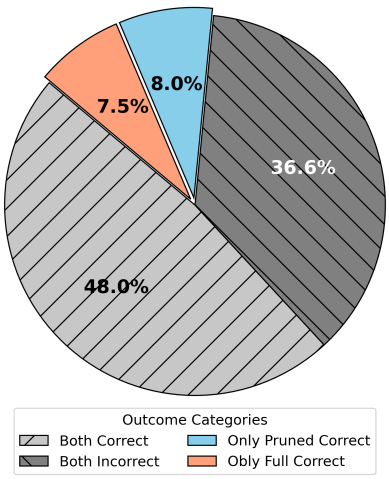

Figure 1: Performance distribution of Pruned vs. Full Schema strategies using a fine-tuned Qwen2.5-7B on the BIRD dev set.

Many approaches employ a **Pruned Schema input** strategy. This method pre-selects relevant schema elements to reduce input noise and focus the model's limited attention (Wang et al., 2025; Lee et al., 2025; Pourreza and Rafiei, 2024; Sheng and Xu, 2025). However, its performance is critically bottlenecked by the schema linker's accuracy. Any pre-selection error can cause an irreversible "Information Masking Effect", where the model lacks the necessary information to generate a correct query, thus limiting system robustness (Pourreza and Rafiei, 2024; Maamari et al., 2024).

Conversely, the **Full Schema input** strategy provides complete information, thus avoiding the masking problem. For smaller models, however, the sheer volume of irrelevant schema elements acts as significant noise, overwhelming their rea-

soning capabilities and leading to incorrect SQL generation. This creates a difficult trade-off, as the optimal strategy diverges based on model capability: the Full Schema approach, which is effective for powerful models like GPT-4 (Maamari et al., 2024), often fails for smaller ones.

Our experiment on the BIRD dev set using a fine-tuned Qwen2.5-7B model highlights a clear complementarity (Figure 1). While both Pruned and Full Schema strategies succeed in 48.0% of cases, they uniquely solve 8.0% and 7.5% respectively. This reveals a significant opportunity: an ideal method that could combine both paths would elevate accuracy to a potential of **63.4%**. We argue that a binary choice is unnecessary, as the high recall from the full schema and the efficient reasoning from the pruned schema are both indispensable.

To resolve this core contradiction, we propose the Dual-Schema Reasoning for Text-to-SQL (DSR-SQL) framework. While agnostic to the underlying model, this framework is particularly pivotal for smaller LLMs as it employs a dual-path cooperative mechanism to balance information completeness and reasoning precision. A key challenge in this design is ensuring the reliability of the precision path, which depends on the schema linker's accuracy. To bolster this critical component, we designed the Permutation-Invariant Minimum Cross-Entropy (PI-MCE) loss function, which substantially improves schema-linking accuracy by addressing training bias from output-order uncertainty. This enhancement makes the entire DSR-SQL cooperative mechanism more robust and effective. Our method achieves an execution accuracy (EX) of 67.41% on the BIRD test set, outperforming most methods that do not employ self-consistency, including those relying on closed-source large models. Our code is available in a GitHub repository[1] to replicate the results reported in this paper.

2 Related Work

Text-to-SQL research has progressed from early rule-based systems and manual templates (Zelle and Mooney, 1996) to neural sequence-to-sequence models (Sutskever et al., 2014). A key challenge has been effectively encoding complex database schemas and their relationships. To address this, approaches leveraging Graph Neural Networks (GNNs) and relation-aware mechanisms emerged. For instance, SADGA (Cai et al., 2021) and LGESQL (Cao et al., 2021) built graph representations of schemas, while RAT-SQL (Wang et al., 2019) and RASAT (Qi et al., 2022) introduced relation-aware self-attention to explicitly model schema linking and relationships within the transformer architecture.

The advent of Large Language Models (LLMs) marked a significant paradigm shift, enabling Text-to-SQL through In-Context Learning (ICL) by providing examples within the prompt (Rajkumar et al., 2022). To enhance the robustness and accuracy of LLM-generated SQL, advanced pipelines have been developed. These often incorporate Chain-of-Thought (CoT) reasoning (Wei et al., 2022; Pourreza and Rafiei, 2023; Wang et al., 2025) to generate explicit intermediate steps, Self-Consistency (SC) voting (Wang et al., 2022; Lee et al., 2025) to aggregate diverse reasoning paths, and iterative self-correction mechanisms (Dong et al., 2023; Talaei et al., 2024) that refine SQL queries based on execution feedback or semantic checks.

While powerful, proprietary LLMs pose challenges regarding cost, privacy, and deployment. Consequently, recent research has focused on adapting smaller, open-source models for Text-to-SQL. This involves supervised fine-tuning (SFT) on specialized datasets (Pourreza and Rafiei, 2024; Li et al., 2024; Domínguez et al., 2024) to imbue domain-specific knowledge. Synthetic data augmentation strategies (Yang et al., 2024; Li et al., 2025) are employed to overcome data scarcity for SFT. Furthermore, Reinforcement Learning (RL) techniques are being explored to further refine model performance, such as Group Relative Policy Optimization (GRPO) (Guo et al., 2025) which optimizes for execution accuracy without a value function (Sheng and Xu, 2025; Pourreza et al., 2025). Our work builds on this by introducing architectural innovations for efficient small-model reasoning, aiming to achieve competitive performance without relying on large, proprietary models.

3 Mehtod

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

As illustrated in Figure 2, DSR-SQL addresses the trade-off between "information completeness" and "reasoning precision" through a dual-path collaborative architecture. The process begins with a user question and a database schema, augmented

[1] https://anonymous.4open.science/r/DSR-SQL-C1C1

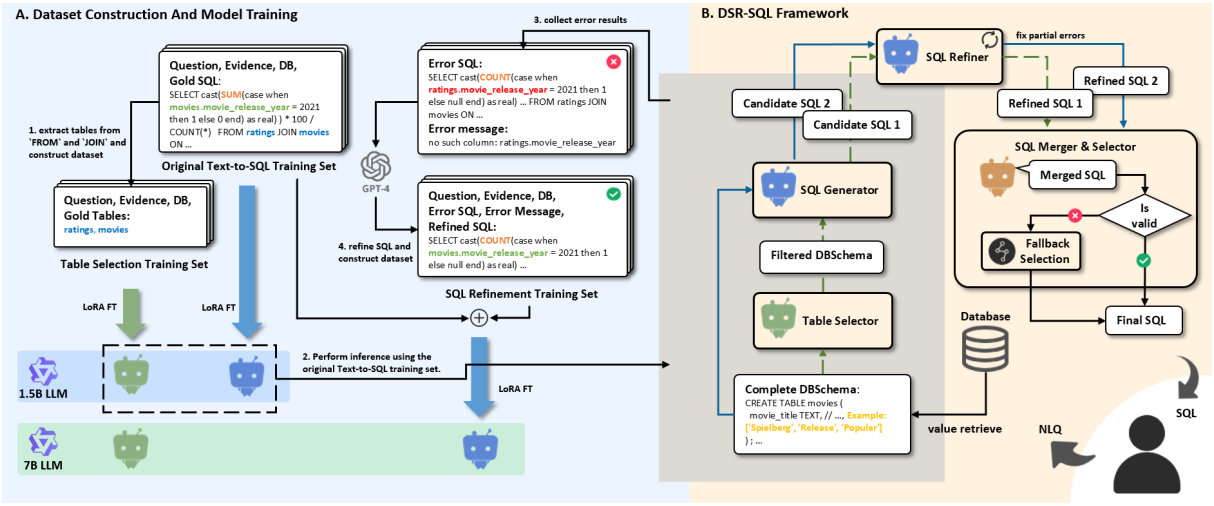

Figure 2: Part A details the methodology for constructing the training set used to train our specialized model. Part B outlines our DSR-SQL framework. Specifically, during the construction of the SQL Refinement training set, the fine-tuned 1.5B model is employed to perform inference and identify erroneous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

by value retrieval (details in Appendix A.1). Unlike binary choices, our framework pursues a 'best of both worlds' approach by processing the query concurrently via Path 1 (Precision-First) and Path 2 (Recall-First). The final output is synthesized by an intelligent Merger or a fallback selector. Specific training methodologies are detailed in Section 3.2.

Table Selector This module prunes the full schema S based on the question Q to output a filtered schema S', formalized as $S' = f_{\text{sel}}(Q, S|\theta_{\text{sel}})$. To achieve high accuracy, we fine-tune a dedicated language model using a Permutation-Invariant Minimum Cross-Entropy (PI-MCE) loss, specifically designed to handle the unordered nature of the output table set.

SQL Generator The generator operates concurrently along two paths to maximize robustness. **Path 1**: The Precision-First Path uses the filtered schema S' to generate $sql_1 = f_{\text{gen}}(Q, S'|\theta_{\text{gen}})$. By operating on a pruned schema, this path minimizes contextual noise and focuses on the core logic. Conversely, **Path 2**: The Recall-First Path operates directly on the complete schema S to produce $sql_2 = f_{\text{gen}}(Q, S|\theta_{\text{gen}})$. This ensures that no potentially relevant information is missed, safeguarding against the "information masking" effect caused by potential errors in the initial pruning stage.

SQL Refiner This module provides iterative self-correction. When an initial query (sql_1 or sql_2) fails to execute, the refiner uses the error feedback E to generate a corrected query: $sql_{\text{ref}} = f_{\text{ref}}(Q, S, sql_{\text{err}}, E|\theta_{\text{ref}})$.

SQL Merger & Selector This final module determines the optimal output sql_{final}. We directly leverage the trained discriminator from the CSC-SQL (Sheng and Xu, 2025) framework as our intelligent merger. It analyzes the candidate queries (sql_1, sql_2) and their execution results (r_1, r_2) to synthesize a superior query: $sql_{\text{mer}} = f_{\text{mer}}(Q, sql_1, sql_2, r_1, r_2|\theta_{\text{csc}})$. If the merger fails to produce a valid result, the system degrades to a deterministic fallback strategy (detailed in Appendix A.2), which prioritizes valid queries and defaults to Path 1 (sql_1) in case of conflict.

3.2 Training Set Construction and Model Fine-Tuning

To enable the practical application of our DSR-SQL framework, it is essential to construct a high-quality training dataset and fine-tune specialized models. The following section details the methodology for dataset construction and the fine-tuning strategy.

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, forming a "Gold Table Set." These sets were then paired with the original question and the full database schema to create

training samples formatted as `(Question, Full DB Schema)` → `Gold Table Set`.

SQL Generation Training Set For the fundamental SQL generation task, we utilized the standard training pairs provided by the benchmarks. These are formatted as `(Question, Schema)` → `Gold SQL`.

SQL Refinement Training Set To construct a dataset that effectively teaches correction, we adopted a comprehensive **"Generate-Execute-Refine"** methodology to synthesize high-quality debugging data. We began by fine-tuning preliminary models (based on Qwen2.5-1.5B) solely on the generation task. These models were then used to generate SQL queries for the training set, which were subsequently executed against the database. We specifically collected instances that failed, categorizing them into two types: *Execution Errors* (syntax issues or runtime exceptions) and *Logic Errors* (queries that executed successfully but yielded results inconsistent with the execution outcomes of the Gold SQL).

To generate effective corrections from these failures, we employed GPT-4 (Achiam et al., 2023) as a "reasoning teacher." A naive approach of directly pairing the `Error SQL` with the `Gold SQL` would be suboptimal, as the `Gold SQL` often differs significantly in structure from the model's generated SQL, making it an unsuitable target for learning specific debugging steps. Instead, we prompted GPT-4 with the `Question`, `Schema`, `Error SQL`, `Error Message`, and the `Gold SQL` (as a reference). GPT-4 was explicitly instructed to perform a **minimal, logical correction**: it had to fix the specific error identified by the database engine or the logic gap, while preserving the original structure of the `Error SQL` as much as possible. This process yielded a refined query, denoted as `Refined SQL`, which represents a step-by-step correction rather than a complete rewrite. The final training samples are thus formatted as `(Question, Schema, Error SQL, Error Message)` → `Refined SQL`.

These refinement samples were then merged with the generation samples to create a unified multi-task dataset.

Fine-Tuning Strategy We employed Low-Rank Adaptation (LoRA) to reduce computational costs and mitigate catastrophic forgetting (Hu et al., 2022). The models were trained for 3 epochs using BF16 mixed-precision, with a batch size of 16 and a maximum sequence length of 8192. We set the learning rate to 5e-5, utilizing a cosine learning rate scheduler with a warmup_ratio of 0.01, and applied a weight_decay of 0.01. Key LoRA hyperparameters included a rank (r) of 128, an alpha (α) of 64, and a dropout rate of 0.1. The entire fine-tuning process for the Qwen2.5-7B model was conducted on a single NVIDIA A800 80G GPU and completed in approximately one day.

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

In standard sequence-to-sequence tasks, the model is trained to generate a fixed target sequence. The standard token-averaged Cross-Entropy (CE) loss is highly effective for this purpose. Given an input X and a target sequence $Y = (y_1, ..., y_T)$, the CE loss is calculated as:

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

where θ represents the model parameters and $P(y_t|y_{<t}, X;\theta)$ is the predicted probability of the next token.

However, this approach is ill-suited for tasks where the output represents an unordered set. In our table selection task, for instance, the goal is to generate a set of relevant table names. The sequences "Teacher Table, Student Table" and "Student Table, Teacher Table" are textually different but semantically identical, as they refer to the same set. Using standard CE with a single fixed label would unfairly penalize the model for generating a valid but differently ordered sequence, forcing the model to overfit to an arbitrary, semantically meaningless order, which unnecessarily penalizes valid predictions.

To overcome this, we propose the Permutation-Invariant Minimum Cross-Entropy (PI-MCE) loss. The core idea is to treat all valid permutations of the target set as acceptable. For a given ground-truth set T_{gold} with k items, we generate all $k!$ permutations, forming a set of target sequences $\Pi_{seq}(T_{gold})$. We then compute the CE loss for each permutation $Y_\pi \in \Pi_{seq}(T_{gold})$:

$$\mathcal{L}_{CE}(Y_\pi|X;\theta) = -\frac{1}{|Y_\pi|}\sum_{t=1}^{|Y_\pi|}\log P(y_t^\pi|y_{<t}^\pi, X;\theta) \tag{2}$$

The final PI-MCE loss is defined as the minimum loss over all these permutations, which are gener-

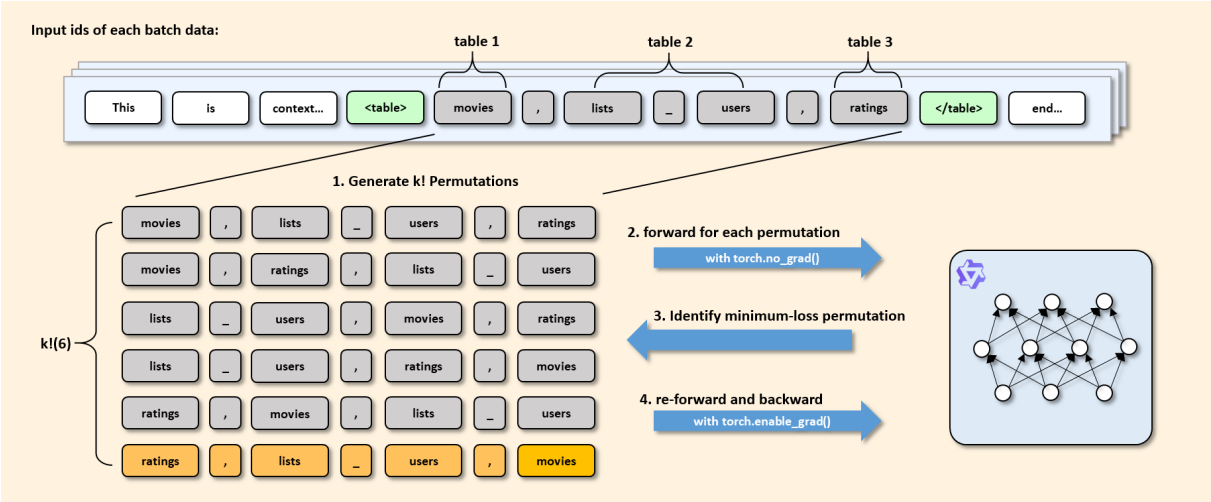

Figure 3: The implementation pipeline for the PI-MCE loss. The process identifies the optimal permutation with minimal loss in a gradient-free pass before performing a single, gradient-enabled backward pass.

ated from the ground-truth set T^Y_{gold} corresponding to the given target Y. This allows the model to match the target sequence that is closest to its own prediction distribution.

$$\begin{aligned}&\mathcal{L}_{PI-MCE}(Y|X;\theta)\\&= \min_{Y_\pi \in \Pi_{seq}(T^Y_{gold})} \mathcal{L}_{CE}(Y_\pi|X;\theta)\\&= \min_{Y_\pi \in \Pi_{seq}(T^Y_{gold})} \left[-\frac{1}{|Y_\pi|} \sum_{t=1}^{|Y_\pi|} \log P(y^\pi_t | y^\pi_{<t}, X;\theta) \right]\end{aligned} \tag{3}$$

By design, PI-MCE aligns the model's sequential generation with the set-based objective, encouraging it to focus on *what* to select rather than *how* to order it.

As illustrated in Figure 3, the implementation of PI-MCE follows an efficient two-stage procedure. First, all $k!$ permutations of the ground-truth set are evaluated in a gradient-free forward pass to identify the one with the minimum cross-entropy loss; during this step, no gradient information is computed or stored. Subsequently, a second forward pass is performed exclusively on this minimum-loss permutation, this time with gradient computation enabled. The model parameters are then updated via backpropagation using the loss from only this single, optimal path, thus maintaining computational tractability.

4 Experiments

4.1 Experimental Setup

Models This study trains and evaluates all modules, including the table selector, SQL generator, and refiner, on open-source Large Language Models (LLMs) with approximately 7 billion parameters. Our primary results are based on Qwen2.5-7B (Hui et al., 2024), with DeepSeek-6.7B (Guo et al., 2024), Mistral-7B, and Gemma-7B used for comparative analysis to validate our approach's robustness.

Datasets To evaluate generalization and robustness, we employ several key Text-to-SQL benchmarks. Our primary benchmark is BIRD (Li et al., 2023), which features realistic Business Intelligence (BI) scenarios with large-scale databases, dirty values, and complex operations. It comprises 9,428 training samples, 1,534 development samples, and 1,789 test samples across 95 databases in 37 professional domains. We also utilize Spider (Yu et al., 2018) to assess generalization to unseen schemas. Spider contains 7,000 training samples, 1,034 development samples, and 2,147 test samples covering 200 databases across 138 diverse domains. Additionally, we include its variants Spider-SYN (Gan et al., 2021a) and Spider-DK (Gan et al., 2021b) to evaluate robustness against lexical variations and the ability to integrate domain knowledge, respectively.

Metrics Our primary metric is Execution Accuracy (EX), which assesses functional correctness by executing the generated SQL and is widely considered the most reliable standard for evaluating practical utility. Furthermore, to specifically evaluate the intermediate table selection module, we report Exact Accuracy, Precision, Recall, and Filtered Accuracy. Notably, **Filtered Accuracy** assesses

Model	test	dev
Open-source LLMs (∼7B)		
DSR-SQL+Qwen2.5-7B (Ours)	**67.41**	**62.26**
DTS-SQL+DeepSeek-6.7B (Pourreza and Rafiei, 2024)	60.31	55.8
SFTCodeS-7B (Li et al., 2024)	59.25	57.17
Closed-source LLMs (unspecified size)		
MAC-SQL+GPT-4 (Wang et al., 2025)	59.59	59.39
DAIL-SQL+GPT-4 (Gao et al., 2024)	57.41	54.76
DIN-SQL+GPT-4 (Pourreza and Rafiei, 2023)	55.90	50.72

Table 1: Comparison of execution accuracy (EX) on BIRD's held-out test set and open development set.

Model	test	dev
CHASE-SQL+Gemini1.5 (Pourreza et al., 2024)	87.6	-
DAIL-SQL+GPT-4 (Gao et al., 2024)	86.6	84.4
DSR-SQL+Qwen2.5-7B (Ours)	**85.5**	**86.5**
DTS-SQL+DeepSeek-6.7B (Pourreza and Rafiei, 2024)	85.3	85.5
DIN-SQL+GPT-4 (Pourreza and Rafiei, 2023)	85.3	82.8
MAC-SQL+GPT-4 (Wang et al., 2025)	82.8	86.75

Table 2: Comparison of execution accuracy (EX) on the Spider test set and development set.

whether the predicted table set fully encompasses all ground-truth tables. Unlike Exact Accuracy, it tolerates the inclusion of redundant tables, prioritizing the complete coverage of necessary information (*i.e.*, Recall) over strict set matching.

Implementation Details During inference, we consistently apply greedy decoding across all stages. This choice reflects realistic, resource-limited deployment scenarios and avoids the high variance introduced by other sampling strategies. To reduce parameter overhead and simplify deployment, the SQL generator and SQL refiner share a single fine-tuned LLM. When necessary, the refiner receives feedback from the executor and performs corrections, with the maximum number of correction iterations set to three.

4.2 Main Results

We evaluated DSR-SQL on the BIRD and Spider benchmarks, with comparative results shown in Table 1 and Table 2. Our primary focus is the challenging BIRD benchmark, which demands strong semantic reasoning and database value understanding. In the realm of open-source models (∼7B), DSR-SQL establishes a new standard, achieving an execution accuracy (EX) of 67.41% on the test set and 62.26% on the development set. This represents a substantial improvement of **7.1** percentage points on the test set over DTS-SQL (60.31%), a strong multi-model baseline, underscoring the superiority of our dual-path collaborative mechanism. Notably, on the development set, DSR-SQL's performance rivals proprietary giants, outperforming GPT-4-based methods like MAC-SQL (59.39%) and DAIL-SQL (54.76%). In parallel, we assessed the model on Spider, a benchmark focused on structural query complexity. DSR-SQL achieves a competitive EX of 85.5% on the test set, outperforming the comparable DTS-SQL. The gap with the state-of-the-art CHASE-SQL (87.6%) is primarily attributed to its use of computationally expensive techniques like Self-Consistency (Wang et al., 2022) with large-scale proprietary models. In contrast, DSR-SQL offers a superior balance between performance and deployment feasibility, delivering strong results with efficient greedy decoding on a 7B model.

4.3 Robustness Analysis

The DSR-SQL framework demonstrates strong robustness, evidenced by its consistent high performance across diverse datasets and base models, as shown in Figure 4. We first evaluated DSR-SQL on four benchmarks covering a wide spectrum of schema complexities and query types (BIRD, Spider, Spider-SYN, and Spider-DK) using Qwen2.5-7B. As illustrated in Figure 4(a), our framework consistently achieves the best performance among the compared methods, showcasing its reliability across varied database environments. Furthermore, we assessed its adaptability by applying it to four different large language models (Qwen2.5-7B, DeepSeek-6.7B, Mistral-7B, and Gemma-7B) on the challenging BIRD dev dataset. The results

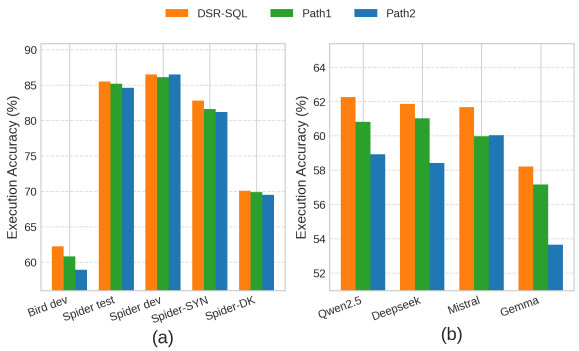

Figure 4: Robustness analysis of the DSR-SQL framework. **(a)** Execution accuracy across five different datasets, using Qwen2.5-7B as the base model, compared against two baselines (Path1, Path2). **(b)** Performance when DSR-SQL is applied to four different ∼7B language models on the BIRD dev dataset.

in Figure 4(b) show that DSR-SQL delivers the most significant performance improvement for every model. This uplift is observed across models with different architectures and initial capabilities, proving our method's benefits are not tied to a specific model's strengths. This dual consistency confirms the framework's high robustness and its practical value as a model-agnostic, plug-and-play enhancement.

4.4 Effectiveness of PI-MCE Loss

We validated the effectiveness of the PI-MCE loss, designed to handle the order-agnostic nature of table selection. Compared to standard token-averaged Cross-Entropy (CE), PI-MCE demonstrates clear advantages. As shown in Table 3, Exact Accuracy rises from 83.57% to 85.33%, and notably, Recall improves from 93.92% to 94.68%.

The improvement in Recall provides direct evidence that PI-MCE reduces false negatives. Under the standard CE loss, the model is penalized for sequence mismatches even if the correct tables are selected, which can suppress the model's confidence in predicting potential candidates. By removing this order constraint, PI-MCE encourages the model to retrieve all plausible tables without the risk of sequence-based penalties. The higher Recall confirms that fewer ground-truth tables are missed, effectively minimizing the risk of information loss before the SQL generation stage.

4.5 Ablation Study

To systematically evaluate the contribution of each component, we conduct a comprehensive ablation study on the BIRD development set, with results detailed in Table 4.

Loss	EX	F_Acc	Pre	Rec
CE	0.8357	0.867	0.9628	0.9392
PI-MCE	0.8533	0.882	0.9659	0.9468

Table 3: Comparison of table selection performance on the BIRD development set. Qwen2.5-7B was fine-tuned with identical training parameters and data, utilizing both cross-entropy (CE) loss and PI-MCE loss (EX = Exact Accuracy; F_Acc = Filtered Accuracy; Pre = Precision; Rec = Recall).

Ablation of DSR-SQL Framework We first analyze the architectural components of DSR-SQL. The full framework achieves an execution accuracy of 62.26%. By removing the SQL Merger & Selector, the model degrades to Path1, with performance dropping to 60.82%. This 1.44-point decrease highlights the effectiveness of the merger-selector mechanism in intelligently combining the outputs from the dual paths to produce a superior final query.

Path1 outperforms Path2 (58.93%), which corresponds to the removal of the table selector from Path1. This demonstrates that the Table Selector in Path1 generally succeeds in improving precision by filtering out irrelevant schema information. However, in more extensive experiments, we observe that Path1 does not consistently outperform Path2 (details in Appendix A.3). This inconsistency highlights the risk of 'information masking,' where an imperfect Table Selector might erroneously discard crucial tables or columns. This validates our dual-path design, which balances the precision of Path1 with the information completeness of Path2.

Finally, removing the SQL Refiner (`w/o sql refiner`) causes a severe performance drop across the board—by 2.61, 4.17, and 3.32 points for DSR-SQL, Path1, and Path2, respectively—confirming its critical role in correcting and enhancing generated queries.

Ablation of PI-MCE Loss We then evaluate the impact of our proposed PI-MCE loss function, which is designed to optimize the Table Selector in Path1. As shown in Table 4, replacing PI-MCE with a standard Cross-Entropy loss (`w/o PI-MCE`) reduces the accuracy of the full DSR-SQL framework from 62.26% to 61.67% and that of Path1 from 60.82% to 59.97%. This confirms that PI-

	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/ UPPER	64.08 (+1.82)	62.91 (+2.09)	-

Table 4: Ablation study results on the BIRD development set using the Qwen2.5-7B model. All scores are Execution Accuracy (%). `w/o` denotes removing a component, and `UPPER` represents the upper-bound performance with a perfect table selector. Changes relative to the `base` method are shown in parentheses.

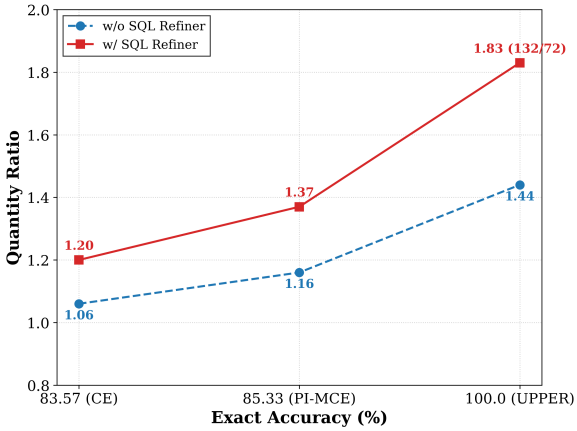

Figure 5: Evolution of the Quantity Ratio (number of problems solved correctly by Path1 but not Path2, divided by the reverse) as Table Selector accuracy increases. The gap between the curves demonstrates the refiner's effectiveness, while the finite ratio at the UPPER bound confirms the necessity of the dual-path framework.

MCE provides a tangible benefit to the schema linking stage, which propagates to the final system performance.

To quantify the remaining potential, we conducted an upper-bound experiment (`UPPER`) by providing the model with a perfect table selection. This oracle setup reveals a performance ceiling of 64.08% for DSR-SQL and 62.91% for Path1. The gap between our PI-MCE-trained model and this upper bound is 1.82 points for DSR-SQL and 2.09 points for Path1. These results indicate that while PI-MCE significantly closes the performance gap, schema linking remains a key bottleneck with substantial room for future improvement.

5 Discussion

To investigate the necessity of our dual-path mechanism, we analyze the *Quantity Ratio*, which compares the problems solved exclusively by each path. Figure 5 illustrates this ratio's evolution across three Table Selector accuracy levels: 83.57% (CE), 85.33% (PI-MCE), and 100.0% (UPPER), comparing performance with and without the SQL Refiner.

Activating the SQL Refiner consistently elevates the Quantity Ratio (Figure 5), indicating it provides a disproportionately larger benefit to Path1 despite processing candidates from both paths. An explanation is that Path1's precision-focused nature generates SQLs that are structurally sounder with minor, correctable flaws, which the refiner is adept at fixing. Conversely, Path2's recall-oriented candidates may contain more fundamental structural errors, less amenable to refinement. This effect is amplified as schema linking improves, with the ratio climbing from 1.20 to 1.83, confirming that high-quality schema and refinement work in synergy to maximize Path1's effectiveness.

Crucially, this analysis also reveals the irreplaceability of Path2. Even under ideal conditions—perfect table selection for Path1 and a system-wide SQL Refiner enhancing both paths—the ratio remains finite at 1.83. At this upper bound, 72 instances are still solved exclusively by Path2, compared to 132 by Path1. This phenomenon demonstrates that the "redundant" schema available to Path2 provides essential, implicit context for certain complex queries that even a perfectly refined, precision-filtered approach cannot handle. This finding strongly justifies our dual-path architecture, proving Path2 is an essential complementary solver.

6 Conclusion

In conclusion, we propose DSR-SQL, a novel dual-path framework designed to empower small open-source language models to achieve high performance in Text-to-SQL tasks. By reasoning over both full and pruned schemas in parallel, our approach effectively balances the high recall derived from comprehensive information with the high precision of a focused view, thereby mitigating the risk of information masking inherent in single-path methods. Furthermore, this process is optimized by our proposed PI-MCE loss, which aligns the training objective with the set-like nature of schema linking to significantly reduce false negatives. Experimental results on the challenging BIRD benchmark show that our model achieves 67.41%, outperforming most current open-source approaches.

Limitations

Despite its strong performance, DSR-SQL has several limitations that warrant future research. Firstly, the PI-MCE loss function exhibits factorial computational complexity ($O(k!)$), restricting its practical application to tasks involving the selection of a small number of items. Secondly, our current dual-schema strategy is limited to complete and pruned schema representations, which may not cover all complex schema linking scenarios; future work could explore a more dynamic framework incorporating additional schema granularities, such as schemas generated based on column selection, to further enhance robustness. Finally, the framework's effectiveness has primarily been validated on 7-billion-parameter scale models, and its scalability to significantly larger or smaller models remains to be systematically investigated.

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*.

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. *arXiv preprint arXiv:2106.01093*.

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 Patricio Daher. 2024. Blar-sql: Faster, stronger, smaller nl2sql. *arXiv preprint arXiv:2401.02997*.

Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Jinshu Lin, Dongfang Lou, and 1 others. 2023. C3: Zero-shot text-to-sql with chatgpt. *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. *arXiv preprint arXiv:2109.05157*.

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.*

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, and 1 others. 2024. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 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, Tieying 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):3672–3685.

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, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, and 1 others. 2023. 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.

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*.

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 *EMNLP (Findings)*.

Mohammadreza Pourreza, Shayan Talaei, Ruoxi Sun, Xingchen Wan, Hailong Li, Azalia Mirhoseini, Amin Saberi, and 1 others. 2025. Reasoning-sql: Reinforcement learning with sql tailored partial rewards for reasoning-enhanced text-to-sql. *CoRR*.

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*.

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

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.

Jiaxi Yang, Binyuan Hui, Min Yang, Jian Yang, Junyang Lin, and Chang Zhou. 2024. Synthesizing text-to-sql data from weak and strong llms. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7864–7875.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, and 1 others. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

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 Value Retrieve

Databases frequently contain an extensive volume of data, yet for any given natural language question, only a small subset of values is typically relevant. The accurate retrieval of these pertinent values is critical, as they are often integral components of SQL clauses such as `WHERE` and `HAVING`.

To address this challenge, our value retrieval module adapts the methodology proposed in CHASE-SQL (Pourreza et al., 2024). The process commences by extracting key terms from the input question using a Large Language Model (LLM) prompted with few-shot examples. Subsequently, for each extracted keyword, we retrieve a set of candidate values from the database that are most syntactically and semantically similar. These candidates are then re-ranked to ensure high relevance. This approach is designed to be robust to typographical errors in the user's question and effectively considers the semantics of keywords during retrieval, leading to precise and relevant value identification.

Method	Bird	Spider		Spider-SYN	Spider-DK
	dev	test	dev		
Qwen2.5-Coder-7B-Instruct					
MSR-SQL	62.39	85.5	86.5	82.8	70.1
w/o sql merger&selector	60.82 (-1.44)	85.2 (-0.3)	86.1 (-0.4)	81.6 (-1.2)	69.9 (-0.2)
w/o sql merger&selector + table selector	58.93 (-1.89)	84.6 (-0.6)	86.5 (+0.4)	81.2 (-0.4)	69.5 (-0.4)
w/o sql merger&selector + table selector + sql refiner	55.61 (-3.32)	83.0 (-1.6)	85.4 (-1.1)	78.6 (-2.6)	68.6 (-0.9)
deepseek-coder-6.7b-instruct					
MSR-SQL	61.86	85.1	85.1	80.2	–
w/o sql merger&selector	61.02 (-0.84)	84.4 (-0.7)	85.0 (-0.1)	78.9 (-1.3)	–
w/o sql merger&selector + table selector	58.41 (-2.61)	85.1 (+0.7)	82.9 (-2.1)	77.0 (-1.9)	–
w/o sql merger&selector + table selector + sql refiner	55.87 (-2.54)	83.9 (-1.2)	82.6 (-0.3)	76.2 (-0.8)	–
Mistral-7B-Instruct					
MSR-SQL	61.67	85.7	86.0	82.1	–
w/o sql merger&selector	59.97 (-1.70)	85.3 (-0.4)	84.9 (-1.1)	80.6 (-1.3)	–
w/o sql merger&selector + table selector	60.04 (+0.07)	83.5 (-1.8)	84.6 (-0.3)	80.9 (+0.3)	–
w/o sql merger&selector + table selector + sql refiner	58.21 (-1.83)	82.2 (-1.3)	83.7 (-0.9)	79.8 (-1.1)	–
Gemma-7B					
MSR-SQL	58.21	–	–	–	–
w/o sql merger&selector	57.17 (-1.04)	–	–	–	–
w/o sql merger&selector + table selector	53.65 (-3.52)	–	–	–	–
w/o sql merger&selector + table selector + sql refiner	51.50 (-2.15)	–	–	–	–

Table 5: Ablation study results of the MSR-SQL framework on different benchmark datasets and base models (Execution Accuracy, EX). The values in parentheses represent the percentage change in performance relative to the configuration in the row above.

A.2 Fallback Selection Algorithm

This appendix details the deterministic, rule-based fallback selection algorithm. This algorithm is invoked as a safety net when the primary *Intelligent Merger* component fails to produce a valid query. The procedure's goal is to select the most reliable and efficient query from the two original candidates: sql_1 (from the precision-first path, using a filtered schema) and sql_2 (from the recall-first path, using the full schema).

The algorithm's logic, formalized in Algorithm 1, is comprehensive. It prioritizes executable queries over non-executable ones. When both candidates are valid, it further analyzes their execution outcomes. If the results are identical, it selects the query with the shorter execution time, optimizing for performance. If the results differ, it conservatively defaults to sql_1, which is generated from a more constrained schema and is thus considered more focused and less likely to contain hallucinations. In the event that both queries fail, it also defaults to sql_1. This multi-faceted approach ensures a robust, efficient, and reliable final selection under all circumstances.

A.3 Detailed Ablation Study Results on Different Base Models

Table 5 presents the ablation study results of the MSR-SQL framework across various benchmark datasets (Bird, Spider, Spider-SYN, and Spider-DK) and base models (Qwen2.5, DeepSeek-Coder, Mistral, and Gemma). The evaluation metric is Execution Accuracy (EX), with values in parentheses indicating the percentage change in performance relative to the preceding configuration. The study systematically removes components—specifically the SQL merger & selector, the table selector, and the SQL refiner—to assess their individual contributions.

Overall, the results demonstrate that the full MSR-SQL framework generally achieves the highest performance, and accuracy tends to decline as components are removed, validating the effectiveness of the proposed modules. However, the impact of the *table selector* varies across different settings. While removing the table selector leads to a significant performance drop in most cases (e.g., a 3.52% decrease on the Bird dataset with Gemma-7B), there are specific instances where performance improves without it. For example, with the *deepseek-coder-6.7b-instruct* model on the Spider test set, accuracy increases by 0.7% after

removing the table selector. Similarly, slight improvements are observed with *Mistral-7B-Instruct* on the Bird dev (+0.07%) and Spider-SYN (+0.3%) datasets. This suggests that while the table selector is generally beneficial for filtering noise, it may occasionally exclude necessary schema information or introduce errors in certain contexts.

Algorithm 1 Fallback SQL Selection Logic with Execution Semantics

▷ Helper function for executing SQL
function EXECUTE(sql)
    ▷ Executes the input SQL query and captures its outcome.
    returns a tuple $(\text{success}, \text{result}, \text{time})$ where:
        success is a boolean indicating if the query ran without errors.
        result contains the data returned by the query, or is null on failure.
        time is the measured execution duration.
end function

▷ Main selection logic
Input: sql_{full}, $\text{sql}_{\text{filtered}}$
Output: $\text{sql}_{\text{final}}$
$(\text{success}_{\text{full}}, \text{result}_{\text{full}}, \text{time}_{\text{full}}) \leftarrow \text{Execute}(\text{sql}_{\text{full}})$
$(\text{success}_{\text{filtered}}, \text{result}_{\text{filtered}}, \text{time}_{\text{filtered}}) \leftarrow \text{Execute}(\text{sql}_{\text{filtered}})$
if $\text{success}_{\text{full}}$ **and not** $\text{success}_{\text{filtered}}$ **then**
    return sql_{full}
else if not $\text{success}_{\text{full}}$ **and** $\text{success}_{\text{filtered}}$ **then**
    return $\text{sql}_{\text{filtered}}$
else if $\text{success}_{\text{full}}$ **and** $\text{success}_{\text{filtered}}$ **then**
    if $\text{result}_{\text{full}} \equiv \text{result}_{\text{filtered}}$ **then**
        if $\text{time}_{\text{filtered}} < \text{time}_{\text{full}}$ **then**
            return $\text{sql}_{\text{filtered}}$
        else
            return sql_{full}
        end if
    else ▷ Results differ, prefer the more focused query
        return $\text{sql}_{\text{filtered}}$
    end if
else ▷ Default case (*e.g.*, both failed)
    return $\text{sql}_{\text{filtered}}$
end if