

E³-Rewrite: Learning to Rewrite SQL for Executability, Equivalence, and Efficiency

Dongjie Xu^{1*}, Weijie Shi², Yue Cui³, Qingzhi Ma⁴, Wenwen Xia¹
Ruiyuan Zhang⁵, Shimin Di⁶, Jia Zhu⁷, Kai Zheng⁸, Jiajie Xu^{1†}

¹Soochow University ²The Hong Kong University of Science and Technology
³Alibaba Group ⁴Huatai Securities ⁵Hong Kong Generative AI R&D Center
⁶Southeast University ⁷Zhejiang Normal University
⁸University of Electronic Science and Technology of China

Abstract

SQL query rewriting reformulates queries into more efficient forms while preserving equivalence. Most existing methods rely on predefined rules that struggle to generalize beyond fixed query patterns and cannot express complex rewriting strategies. Large language models (LLMs) can reason over complex structures and generalize to diverse patterns, but their limited execution awareness often results in non-equivalent or inefficient rewrites. To this end, we propose **E³-Rewrite**, an LLM-based framework that generates executable, equivalent, and efficient SQL rewrites. It features two key designs: (1) an execution-aware context construction that integrates query plans and retrieved examples to expose structural bottlenecks; and (2) a reinforcement learning optimization with executability, equivalence, and efficiency rewards, trained under a correctness-to-efficiency curriculum for stable convergence. Experiments on multiple benchmarks show that E³-Rewrite reduces execution time by up to 25.6% over leading baselines and delivers 7.3% more equivalent rewrites, extending SQL rewriting to query patterns previously beyond rule-based methods.

1 Introduction

Efficient query processing is a central goal of modern database systems, and SQL query rewriting plays a key role in improving performance (Li, 2019; Zhou et al., 2022b). The goal is to reformulate a query into an equivalent but more efficient form (Wang et al., 2022). To be practically useful, a rewritten query must satisfy three criteria: (1) **Executability**: the rewritten query must conform to SQL syntax and execute correctly within a target DBMS; (2) **Equivalence**: the rewritten query must preserve the original query’s semantics and

produce identical results; (3) **Efficiency**: the rewritten query should reduce execution time, and the overhead of the rewriting process itself should be justified by the resulting performance gains.

Most SQL rewriting methods follow a rule-based paradigm: they apply predefined rewrite rules to convert an input query into an equivalent but more efficient form (Bai et al., 2023; Begoli et al., 2018). Early systems rely on manually crafted rules and heuristics, which generalize poorly to diverse query patterns (Liu and Mozafari, 2024; Dong et al., 2023). To improve flexibility, later methods frame rewriting as a rule selection task. For example, LearnedRewrite (Zhou et al., 2021) uses Monte Carlo Tree Search guided by cost estimators to explore rule combinations. Recently, large language model (LLM)-based systems such as LLM-R² (Li et al., 2024) and R-Bot (Sun et al., 2025) leverage LLM to model queries and suggest appropriate rules via prompting or retrieval.

While these approaches reduce manual effort, they remain constrained by the fixed coverage of rule sets. Fundamentally, rule-based methods remain limited in expressiveness, brittle in their dependencies, and poorly adaptable to unseen query structures (Leis et al., 2015; Wang et al., 2022). Figure 1 illustrates such limitations: the original query with a correlated aggregate takes over 300 seconds to execute, a rule-based rewrite reduces runtime to 94.3 seconds but still incurs heavy joins, while our rewrite with a pre-aggregated Common Table Expression (CTE) reduces execution time to only 3.26 seconds with full equivalence.

To overcome these limitations, we propose training an LLM to directly generate equivalent and efficient SQL rewrites without explicit rules. Although supervised fine-tuning (SFT) can in principle be applied, it is ill-suited for SQL rewriting because the task lacks definitive ground-truth labels, as there is no uniquely optimal rewrite and the quality of each candidate can only be verified through execution,

*Email: djxu@stu.suda.edu.cn

†Corresponding author

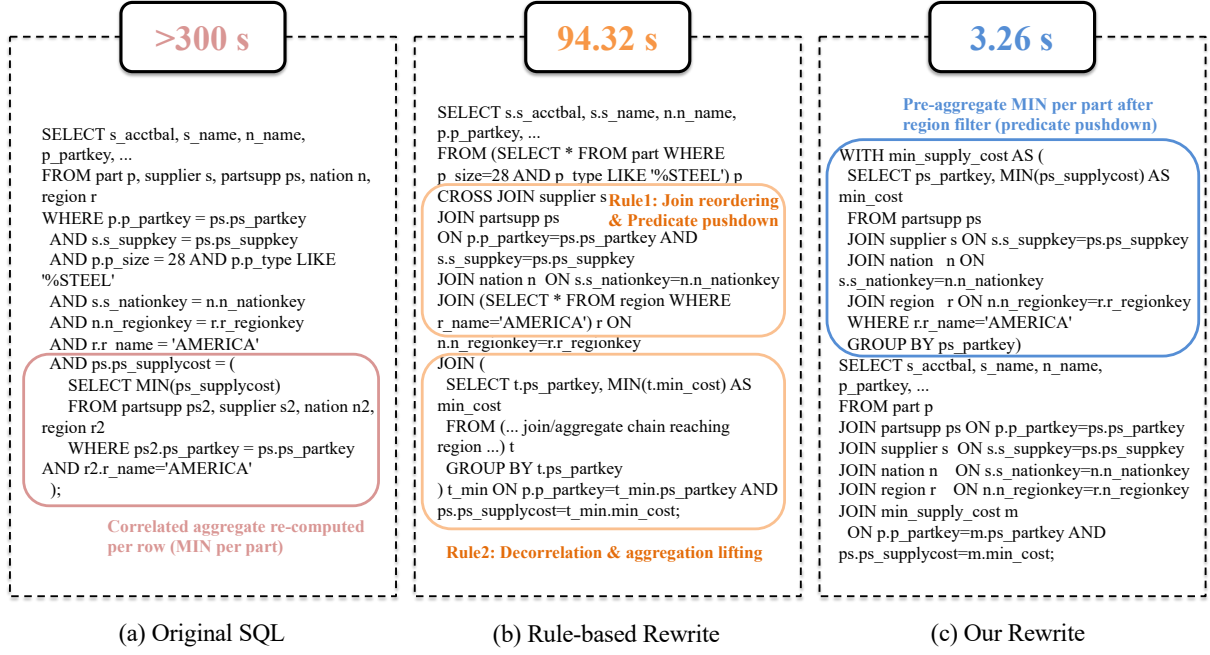


Figure 1: An illustrative example comparing the original SQL (a), a rule-based rewrite (b), and our E^3 -Rewrite output (c). By pre-aggregating the correlated subquery into a CTE and pushing down predicates, our method reduces query latency from over 300s to 3.26s.

providing no stable supervision signal for learning. We therefore adopt reinforcement learning (RL) to optimize SQL rewriting through execution feedback, allowing the model to refine its rewrites autonomously. However, two major challenges remain: (1) Due to the lack of execution awareness, LLMs struggle to generate SQL rewrites that are executable, equivalent, and efficient; (2) The optimization process lacks stable and interpretable signals that connect equivalence with execution performance, making stable learning difficult.

To address the above challenges, we introduce **E^3 -Rewrite**, an LLM-based SQL rewriting framework optimized for executability, equivalence and efficiency. To overcome the lack of execution awareness and semantic grounding, we incorporate query execution plans into the input context to expose logical structures and performance bottlenecks. Furthermore, to improve structural understanding and generalization, we retrieve analogous rewrites from a hybrid structure–semantic demonstration library during inference. To mitigate unstable optimization and the absence of direct supervision, we design an RL framework guided by execution-driven rewards. These rewards jointly capture executability, equivalence, and efficiency, allowing the model to improve through feedback rather than fixed labels. A curriculum-based train-

ing strategy further stabilizes learning by first emphasizing executability and equivalence, and later incorporating latency as an optimization signal.

Our contributions are summarized as follows:

- We present E^3 -Rewrite, to our knowledge the first end-to-end LLM-based SQL rewriting framework that handles complex query structures and extends rewriting beyond the scope of rule-based methods.
- We introduce two key technical components: (i) an execution-aware context construction module that integrates query plans and retrieved demonstrations to expose performance bottlenecks and enhance structural reasoning; and (ii) a reinforcement-learning framework with execution-driven rewards and a correctness-to-efficiency curriculum, enabling stable training across executability, equivalence, and efficiency.
- We evaluate E^3 -Rewrite on multiple SQL benchmarks, achieving up to 25.6% lower execution time than strong baselines and 7.3% more equivalent rewrites.

2 Related Work

SQL Rewriting. Existing query rewriting methods have evolved from rule-based systems to learning-based and LLM-augmented approaches. (1) *Heuristic Rule-based Systems.* Early systems

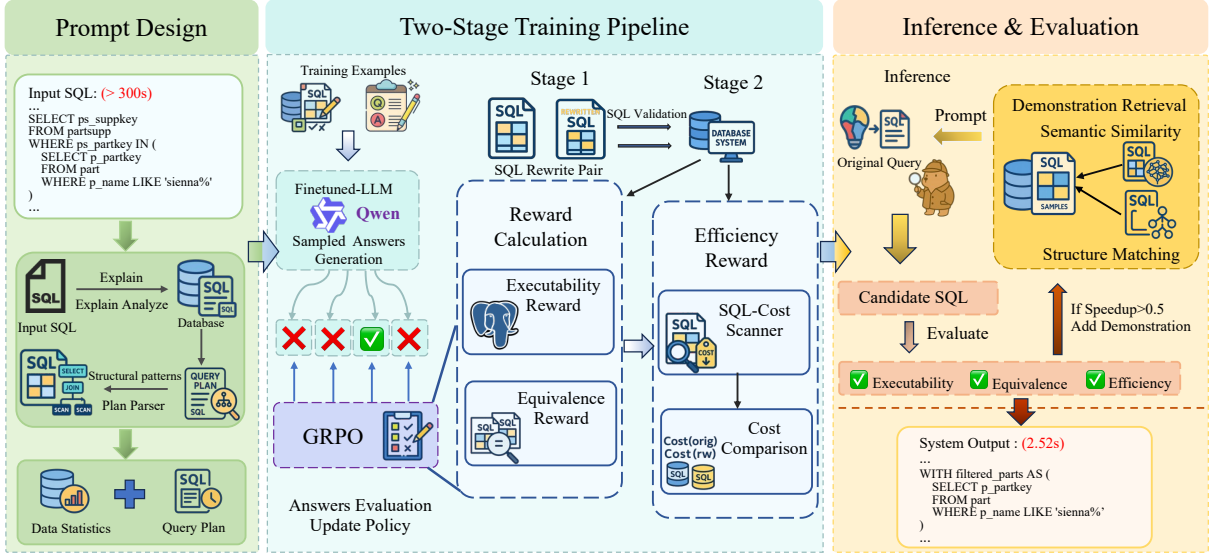


Figure 2: Overview of our E³-Rewrite. The system introduces execution hints into the prompt using parsed query plans, then optimizes a finetuned LLM with two-stage GRPO training that incorporates executability, equivalence, and efficiency rewards. During inference, E³-Rewrite retrieves hybrid demonstrations based on structure and semantics to guide rewriting, and updates its memory pool with rewrites that yield substantial execution gains.

like PostgreSQL (PostgreSQL, 2025) and Volcano (Graefe and McKenna, 1993) rely on fixed or heuristic rule sequences. These approaches often neglect inter-rule dependencies (Graefe, 1995), leading to blind or inefficient search. (2) *Learning-based Rule Selection*. LearnedRewrite (Zhou et al., 2021) treats rewriting as a rule search problem and uses MCTS with learned cost models. However, generalization across schemas remains limited due to reliance on schema-specific reward estimators. (3) *LLM-Augmented Rewriting*. LLM-R² (Li et al., 2024) and R-Bot (Sun et al., 2025) leverage LLMs for rule retrieval or recommendation via prompting. QUITE (Song et al., 2025) adopts a training-free, feedback-aware multi-agent pipeline that combines a structured knowledge base, hybrid SQL corrector, and hint injection to iteratively refine rewrites. While promising, existing approaches remain constrained by limited generalization and execution awareness (Liu and Mozafari, 2024), motivating the need for a fully learnable, feedback-driven rewriting framework beyond rule enumeration.

LLM-Based SQL Generation. Parallel to rewriting, LLMs have achieved remarkable performance in NL2SQL tasks, including DIN-SQL (Poureza and Rafiei, 2023), MAC-SQL (Wang et al., 2025), and COGSQ (Yuan et al., 2025). These systems adopt prompt engineering, chain-of-thought reasoning, or multi-agent collaboration to enhance generation quality. However, most focus solely on produc-

ing executable queries, and evaluate correctness via execution consistency (Yu et al., 2018). They do not ensure strict semantic equivalence (Chu et al., 2017; Zhou et al., 2022a) with a reference query, an essential aspect in SQL rewriting where input-output fidelity is critical.

Beyond SQL, related work has explored reinforcement learning with compiler and symbolic execution feedback for code translation (Jana et al., 2024) and few-shot example selection for LLM-based refactoring (Shirafuji et al., 2023). Unlike these tasks, SQL performance rewriting jointly considers executability, semantic equivalence, and execution efficiency.

3 Problem Formulation

Unlike text-to-SQL, which generates SQL from natural-language questions and is typically evaluated by execution correctness, we focus on **SQL-to-SQL performance rewriting**: rewriting a given SQL query q into an equivalent yet more efficient form. Formally, the rewritten query q' should satisfy:

- **Executability:** q' conforms to SQL syntax and executes successfully on the target DBMS.
- **Equivalence:** $q \equiv q'$, i.e., for any database instance D , the result sets $q(D)$ and $q'(D)$ are identical, ensuring query equivalence.
- **Efficiency:** Under the same execution environ-

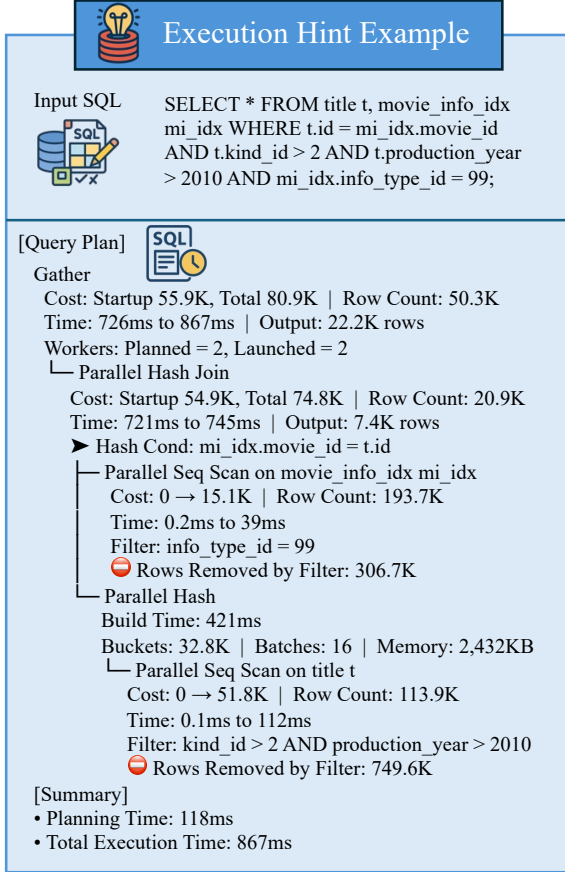


Figure 3: Execution hint example.

ment, the runtime of q' is lower than that of q , i.e., $T(q') < T(q)$.

4 Method

4.1 The Overview of E³-Rewrite

E³-Rewrite is an end-to-end SQL rewriting framework that combines large language models with reinforcement learning and demonstration reuse to produce executable, equivalent, and efficient rewrites. As shown in Figure 2, the system consists of three components: (i) **Prompt Design**, which extracts query execution plans via EXPLAIN/ANALYZE, linearizes them into text sequences, and prepends them to queries to highlight structural bottlenecks; (ii) **Two-Stage Reinforcement Learning**, which fine-tunes the model with GRPO and a curriculum that first emphasizes correctness (executability and equivalence) and then efficiency; and (iii) **Inference & Evaluation**, where a hybrid retriever selects past successful rewrites for guidance and new effective rewrites are added back to the demonstration pool for reuse.

4.2 Execution Hint Injection

Although large language models exhibit strong reasoning ability, limited execution awareness often causes them to generate SQL rewrites that are inefficient or non-equivalent. To address this, we treat query execution plans as *execution hints* that reflect how the DBMS interprets and optimizes a query. These plans reveal operator choices, join orders, and runtime bottlenecks, offering richer optimization cues than raw SQL. We obtain them using standard EXPLAIN for cost estimates and EXPLAIN ANALYZE for runtime traces (PostgreSQL, 2025); the latter is used during training to capture realistic performance data, while the former is used at inference to reduce latency.

To make hints model-consumable, we linearize the hierarchical plan into an indented text sequence that preserves operator nesting and prepend it to the SQL query. This execution-aware context enables the model to detect inefficiencies such as full scans, unindexed joins, or redundant filters, and to generate more efficient rewrites. As shown in Figure 3, the hint highlights bottlenecks like rows discarded by post-filtering or sequential scans without indexes, thereby offering structural cues for rewriting.

4.3 Reinforcement Learning for SQL Rewriting

To enable self-improving SQL rewriting, we fine-tune foundation LLMs with RL guided by execution-aware reward signals. Traditional methods like PPO (Schulman et al., 2017) struggle with discrete, sparse, and conflicting rewards, which are common in SQL rewriting.

We therefore adopt the Group-Relative Policy Optimization (GRPO) algorithm (Guo et al., 2025), which eliminates the need for a value network and stabilizes learning via intra-group comparisons of sampled rewrites. For each input query q , the model samples N candidate rewrites, evaluates them with scalar rewards $\{r_i\}$, and computes relative advantages A_i by normalizing within the group. GRPO then applies a clipped policy-gradient update with KL regularization, where the clipping threshold ϵ and the regularization coefficient β control training stability. This design makes GRPO well-suited for our setting, where rewrite quality depends on non-differentiable and delayed execution feedback.

Execution-Aware Reward Function. To guide

learning with meaningful execution feedback, we define a composite reward that combines three objectives:

$$R_{\text{total}} = \lambda_{\text{eq}} R_{\text{eq}} + \lambda_{\text{exec}} R_{\text{exec}} + \lambda_{\text{perf}} R_{\text{perf}}. \quad (1)$$

- **Executability** (R_{exec}): We use EXPLAIN to pre-execute the query in the target DBMS. If the query passes both parsing and semantic analysis, we set $R_{\text{exec}} = 1$; otherwise, $R_{\text{exec}} = 0$.

- **Equivalence** (R_{eq}): We adopt a three-stage verification pipeline to determine whether the rewritten query is equivalent to the original query. First, we employ two complementary formal SQL equivalence checkers, SQLSolver (Ding et al., 2023) and QED (Wang et al., 2024), to determine equivalence. If the result is positive, we set $R_{\text{eq}} = 1$; if explicitly negative, we set $R_{\text{eq}} = 0$. If the solver returns an unknown verdict (e.g., timeout or unsupported constructs), we invoke a lightweight LLM-based semantic judgment to assess equivalence (Zhao et al., 2023a; Singh and Bedathur, 2024). For cases where the LLM output is inconclusive, we fall back to executing both queries and compare their outputs over sampled database instances. Only when the execution results match exactly, we set $R_{\text{eq}} = 1$; else, we set $R_{\text{eq}} = 0$.

- **Performance Improvement** (R_{perf}): We extract the estimated execution cost from EXPLAIN and define:

$$R_{\text{perf}} = \max \left(0, \frac{T(q) - T(q')}{T(q)} \right), \quad (2)$$

where $T(\cdot)$ denotes the optimizer-estimated cost. This reward encourages performance improvement without compromising executability or equivalence.

Curriculum-Guided Training. To stabilize learning and mitigate reward conflicts between executability, equivalence, and efficiency, we adopt a curriculum RL strategy (Bengio et al., 2009; Feng et al., 2025; Huang et al., 2025). The training process is split into two stages:

- **Stage 1: Correctness-first Learning.** We activate only R_{eq} and R_{exec} , guiding the model to produce SQL rewrites that are executable and equivalent to the original query.

- **Stage 2: Optimization-aware Refinement.** Once the model consistently satisfies correctness constraints, we enable R_{perf} to encourage performance-oriented rewrites. Training proceeds with the full reward function, jointly optimizing for equivalence and efficiency.

- **(Optional) Rehearsal.** To prevent degradation in equivalence or executability, some Stage 1 examples are periodically revisited in Stage 2.

This progressive reward schedule allows the model to first acquire a robust understanding of SQL executability and equivalence, before shifting attention to optimization focused on reducing execution latency. Combined with GRPO’s group level feedback mechanism, this design yields a stable rewriting policy informed by execution behavior, capable of generalizing to diverse and complex SQL workloads.

4.4 Hybrid Demonstration Retrieval

To enable effective demonstration-based rewriting, we design a retrieval module that selects relevant SQL rewrites from a demonstration pool using a hybrid structural–semantic similarity metric.

Let q denote the input SQL query and $\mathcal{C} = \{q_i\}_{i=1}^N$ be the demonstration pool. For each $q_i \in \mathcal{C}$, we compute the overall similarity $\text{Sim}(q, q_i)$ as:

$$\text{Sim}(q, q_i) = \alpha \cdot \text{Sim}_{\text{struct}}(q, q_i) + (1 - \alpha) \cdot \text{Sim}_{\text{sem}}(q, q_i). \quad (3)$$

where $\alpha \in [0, 1]$ is a balancing coefficient between the structural and semantic components.

Structural Similarity. Following prior work on tree-based evaluation for SQL queries (Song et al., 2024), we define structural similarity based on the abstract syntax tree (AST) edit distance. Each SQL query is parsed into an operator-level AST $T(q)$, and the similarity is computed as:

$$\text{Sim}_{\text{struct}}(q, q_i) = 1 - \frac{\text{TED}(T(q), T(q_i))}{\max(|T(q)|, |T(q_i)|)}, \quad (4)$$

Here, $\text{TED}(\cdot, \cdot)$ denotes the tree edit distance computed via the APTED algorithm, and $|T(\cdot)|$ is the number of AST nodes. The score is normalized to $[0, 1]$, with higher values indicating stronger structural resemblance.

Semantic Similarity. To measure the semantic alignment between queries, we compute the cosine similarity of their contextual embeddings:

$$\text{Sim}_{\text{sem}}(q, q_i) = \cos(\mathbf{e}_q, \mathbf{e}_{q_i}), \quad (5)$$

where $\mathbf{e}_q, \mathbf{e}_{q_i} \in \mathbb{R}^d$ are dense representations derived from a pretrained transformer encoder (Reimers and Gurevych, 2019). This component captures query-level intent beyond syntactic forms (Zhao et al., 2023b, 2022).

Query Latency (s)	TPC-H			IMDB			DSB		
	Average	Median	p90	Average	Median	p90	Average	Median	p90
Origin	78.81	10.75	300.00	5.74	1.54	18.53	38.83	3.53	300.00
LearnedRewrite	41.34	9.73	103.41	5.73	1.54	18.51	32.61	3.19	98.67
LLM-R2	54.76	10.02	300	5.32	1.47	17.44	20.12	2.96	48.67
R-Bot	39.89	9.15	84.27	5.54	1.52	18.07	27.75	3.12	89.96
E³-Rewrite (Qwen)	29.67	8.63	51.37	5.01	1.44	16.71	16.93	2.46	38.47

Table 1: Different Methods on Query Latency (s)

Demonstration Retrieval and Pool Update. For an input query q , we retrieve the top- k most similar examples from $\mathcal{C}$ as in-context demonstrations. The pool is dynamically updated: if a new rewrite is equivalent and yields significant speedup (e.g., $> 0.5\times$), the original–rewrite pair is added. This continual update keeps the pool evolving with high-quality, generalizable examples.

5 Experiment

5.1 Experiment Setting

Dataset. We evaluate our system on three widely used SQL rewriting benchmarks: (1) TPC-H (TPC-H Benchmark, 2025), a standard decision support benchmark consisting of 22 parameterized query templates. We generate approximately 2,000 queries over a 10GB database using the official toolkit. (2) IMDB (JOB) (Maas et al., 2011; Leis et al., 2015), a benchmark derived from the Join Order Benchmark, containing 2,000 analytical queries over the IMDB schema with diverse join patterns. (3) DSB (Ding et al., 2021), which extends TPC-DS to model modern decision support scenarios. It includes 2,000 complex queries featuring nested structures, aggregations, and patterns that are particularly challenging for query optimizers.

LLM. For our method, we fine-tune open-source models from the Qwen3 (Yang et al., 2025) families. We also include GPT-4o, a commercial closed-source model that serves as the backbone for several LLM-based baselines, including LLM-R², and R-Bot.

Baselines. We compare our method with the following representative SQL rewriting systems: (1) LearnedRewrite (LR) (Zhou et al., 2021), which uses MCTS guided by a learned cost model to search for the best rule sequences; (2) LLM-R² (Li et al., 2024), which prompts with demonstrations and applies rewrite rules via Apache Calcite; (3) R-

Bot (Sun et al., 2025), a retrieval-augmented LLM system that performs step-by-step rule selection.

Evaluation Metrics. We evaluate the quality of rewritten queries using the following metrics: (1) Equivalence Rate, defined as the proportion of generated rewrites that are executable and produce the same output tuples as the original queries on the same database instance. (2) Query Latency, which measures the execution time of executable and equivalent rewrites; rewrites failing execution or equivalence checks are excluded to avoid conflating efficiency gains with correctness failures. (3) Improved Queries, which counts executable and equivalent rewrites that reduce execution time by at least 10% relative to the original query. For latency measurement, each query is executed five times, with the highest and lowest runs discarded before averaging. We report the mean, median, and 90th percentile (p90) latency under a uniform 300-second timeout, where timed-out queries are recorded as 300 seconds.

System Environment. All queries are evaluated on PostgreSQL v14. Experiments are conducted on a server with a 24-core Intel Xeon @ 3.0GHz, 512 GB RAM, and 8×A100 GPUs (80 GB). Training requires approximately 60 GPU hours, including reinforcement learning and curriculum-based finetuning. Additional implementation details and hyperparameters are provided in Appendix A.

5.2 Execution Efficiency Evaluation

We evaluate the execution efficiency of different rewriting methods using query latency as the primary metric, reflecting runtime cost and overall performance. Each query is executed five times under identical hardware and database settings, and the average latency is computed after removing the highest and lowest values to reduce caching noise. Table 1 summarizes results on TPC-H, IMDB, and DSB. E³-Rewrite achieves the lowest latency

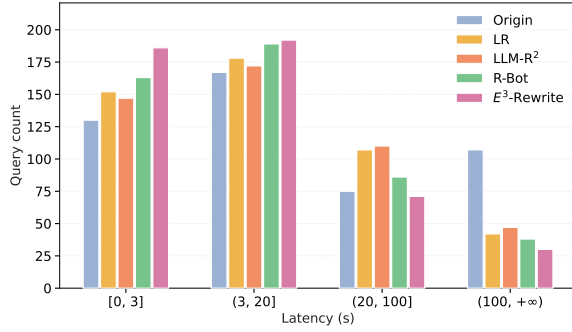


Figure 4: Latency distribution across four log-spaced buckets.

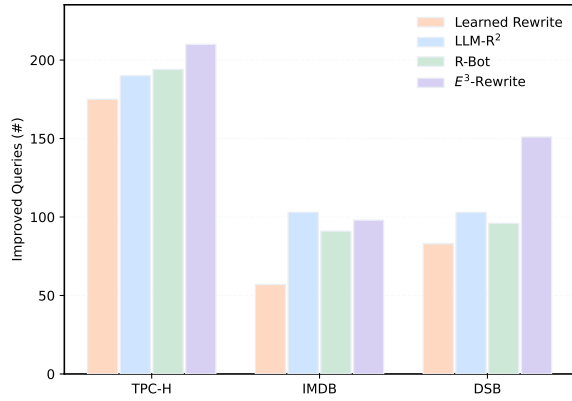


Figure 5: Improvement Counts of SQL Rewrites across Different Benchmarks.

across all datasets, reducing TPC-H average from 78.81s (Original) to 29.67s and p90 latency from 300.00s to 51.37s. LearnedRewrite provides moderate improvement but remains constrained by its fixed rule set. LLM-R² lowers mean latency yet struggles with high-latency outliers, showing instability under complex workloads. R-Bot performs well on IMDB but degrades on DSB with deeper analytical joins. In contrast, E³-Rewrite consistently improves both average and tail latency, demonstrating robust adaptation across diverse queries.

Figure 4 shows latency distributions across four log-spaced ranges. E³-Rewrite clearly shifts query execution toward lower-latency regions, increasing fast queries (<3s) from 130 to 186 and reducing slow ones (>100s) from 107 to 30. This improvement stems from the interplay of three core components: execution hints that expose runtime inefficiencies (e.g., full scans, nested loops), GRPO-based reinforcement learning that stabilizes training while balancing executability, equivalence, and efficiency, and hybrid demonstration retrieval that enhances structural generalization by reusing ef-

Method	TPC-H	IMDB	DSB
LearnedRewrite	88.8	100.0	76.4
LLM-R ²	91.2	99.2	86.4
R-Bot	96.8	98.6	95.4
E ³ -Rewrite	99.6	99.4	98.8

Table 2: Equivalence ratio (%) of SQL rewrites across different benchmarks.

Method	Avg Latency Improvement (s)	Improved Queries	Equiv. Ratio
E ³ -Rewrite (Qwen 14B)	78.81 → 35.58	158	94.8%
E ³ -Rewrite (Qwen 32B)	78.81 → 29.67	210	99.6%

Table 3: Rewrite quality of E³-Rewrite under different model scales on TPC-H.

fective rewrites. Together, these components yield stronger runtime stability and mitigate heavy-tail latency cases by steering the model away from inefficient join orders and redundant scans.

5.3 Rewriting Quality and Model Behavior Analysis

We evaluate rewrite quality using three metrics: average latency, improved query count, and equivalence ratio. Figure 5 and Table 2 compare results on TPC-H, IMDB, and DSB, showing the number of improved queries and the percentage of equivalent rewrites. E³-Rewrite consistently outperforms all baselines, achieving the highest equivalence ratios and improvement counts on every dataset. On TPC-H, it produces 210 improved rewrites with a 99.6% equivalence ratio, surpassing LLM-R² and R-Bot. Similar trends appear on IMDB and DSB, where equivalence rates reach 99.4% and 98.8%.

This strong performance stems from two core design choices. First, incorporating execution plan context enables safer, more targeted rewrites by revealing operator-level bottlenecks and structural dependencies. Second, the reward-based fine-tuning jointly optimizes execution efficiency and semantic fidelity, helping preserve the intent of the original query. Table 3 further analyzes model scaling: E³-Rewrite with Qwen 14B achieves 158 improved queries and a 94.8% equivalence ratio on TPC-H, confirming effectiveness with moderate capacity, while scaling to Qwen 32B yields further gains, demonstrating strong scalability and efficiency.

Method	TPC-H 1G			TPC-H 5G			TPC-H 10G		
	Avg	Median	p90	Avg	Median	p90	Avg	Median	p90
Original	52.58	0.42	300.00	54.49	1.54	300.00	78.81	10.75	300.00
LearnedRewrite	16.98	0.38	18.31	21.02	1.52	26.43	41.34	9.73	103.41
LLM-R2	19.48	0.41	25.06	25.79	1.50	32.38	54.76	10.02	300
R-Bot	14.64	0.40	17.45	19.68	1.41	24.21	39.89	9.15	84.27
E ³ -Rewrite	12.39	0.41	9.47	16.14	1.43	16.28	29.67	8.63	51.37

Table 4: Query Latency Across Different TPC-H Scales.

5.4 Robustness Evaluation

We assess the robustness of E³-Rewrite under varying data scales by evaluating its performance on the TPC-H benchmark at 1GB, 5GB, and 10GB. As shown in Table 4, our method consistently achieves the lowest average and 90th percentile (p90) latencies across all scales. Specifically, p90 latency increases moderately from 9.47s at 1GB to 51.37s at 10GB, indicating strong scalability. In contrast, LLM-R² shows significant performance degradation, with p90 reaching the 300s timeout at the 10GB scale. These results suggest that E³-Rewrite generalizes effectively to larger datasets and more complex query plans.

We attribute this robustness to two key factors: (1) the use of structured, execution-aware prompts that expose inefficiencies in query plans and help the model learn transferable optimization patterns, and (2) reward-guided optimization that aligns model behavior with performance-oriented rewriting while preserving correctness. Such scalability and stability are crucial for real-world deployment, where query complexity, data scale, and workload characteristics vary widely.

5.5 Ablation Study

To better understand the role of each core component in E³-Rewrite, we conduct an ablation study on the TPC-H benchmark. As shown in Table 5, we evaluate the effect of removing three key modules individually—RL, execution plan hints, and demonstration retrieval, while keeping the remaining ones intact to isolate their impact. Specifically, removing RL skips policy optimization and uses the base LLM with plan hints and retrieval. Removing execution plan hints excludes plan-derived context from the prompt, while still applying RL and retrieval. Removing retrieval disables example selection and uses a static prompt template, while

Method	Avg Latency (s)	Improved Queries	Equiv. Ratio
E ³ -Rewrite(full)	29.67	210	99.6%
E ³ -Rewrite (w/o Execution Hint)	39.56	163	98.4%
E ³ -Rewrite(w/o RL)	32.29	194	90.2%
E ³ -Rewrite (w/o Demonstration)	35.39	177	96.4%
E ³ -Rewrite(vanilla)	56.71	125	84.8%

Table 5: Ablation Study of E³-Rewrite on TPC-H.

keeping plan hints and RL active. We also include a vanilla baseline that applies the base LLM without any fine-tuning or structured input.

Disabling RL causes the largest drop in equivalence (99.6%→90.2%), highlighting the necessity of reward-guided optimization. Excluding plan hints yields the highest latency increase (29.67s→39.56s), showing their role in exposing structural inefficiencies (e.g., nested loops, full scans). Removing retrieval reduces the number of improved queries (210→177), indicating that relevant examples enhance generalization. The vanilla variant performs worst (56.71s latency, 84.8% equivalence), confirming the importance of both training and structural guidance. Overall, the results demonstrate distinct contributions: RL enforces equivalence and efficiency, plan hints provide structural awareness, and retrieval supports pattern reuse across queries.

6 Conclusion

We presented E³-Rewrite, an LLM-based framework for executable, equivalent, and efficient SQL rewriting that combines execution-guided context, reinforcement learning with execution-driven rewards, curriculum training, and hybrid demonstration retrieval. Grounded in execution signals

and optimized with multi-objective rewards, E³-Rewrite improves rewrite reliability and performance. Experiments on three benchmarks show efficiency gains and coverage over baselines.

Limitations

While E³-Rewrite achieves consistent gains across multiple SQL benchmarks, several limitations remain. First, the framework currently relies on PostgreSQL for execution feedback and cost estimation, and extending it to other database systems may require adapting plan extraction and cost modeling. Second, equivalence verification and reward computation involve repeated query executions, increasing training overhead. Finally, reward weights and retrieval balance are manually tuned, and the fixed combination of structural and semantic similarity may limit adaptability across workloads.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (Grant No. 62572336, 62272334, 62577050). We acknowledge the support of the "Pioneer" and "Leading Goose" R&D Program of Zhejiang (No. 2026C02A1236).

References

- Qiushi Bai, Sadeem Alsudaïs, and Chen Li. 2023. [Querybooster: Improving SQL performance using middleware services for human-centered query rewriting](#). *Proc. VLDB Endow.*, 16(11):2911–2924.
- Edmon Begoli, Jesús Camacho-Rodríguez, Julian Hyde, Michael J. Mior, and Daniel Lemire. 2018. [Apache calcite: A foundational framework for optimized query processing over heterogeneous data sources](#). In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 221–230. ACM.
- Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. [Curriculum learning](#). In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, volume 382 of *ACM International Conference Proceeding Series*, pages 41–48. ACM.
- Shumo Chu, Chenglong Wang, Konstantin Weitz, and Alvin Cheung. 2017. [Cosette: An automated prover for SQL](#). In *8th Biennial Conference on Innovative Data Systems Research, CIDR 2017, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*. www.cidrdb.org.
- Bailu Ding, Surajit Chaudhuri, Johannes Gehrke, and Vivek R. Narasayya. 2021. [DSB: A decision support benchmark for workload-driven and traditional database systems](#). *Proc. VLDB Endow.*, 14(13):3376–3388.
- Haoran Ding, Zhaoguo Wang, Yicun Yang, Dexin Zhang, Zhenglin Xu, Haibo Chen, Ruzica Piskac, and Jinyang Li. 2023. [Proving query equivalence using linear integer arithmetic](#). *Proceedings of the ACM on Management of Data*, 1(4):1–26.
- Rui Dong, Jie Liu, Yuxuan Zhu, Cong Yan, Barzan Mozafari, and Xinyu Wang. 2023. [Slabcity: Whole-query optimization using program synthesis](#). *Proc. VLDB Endow.*, 16(11):3151–3164.
- Zihao Feng, Xiaoxue Wang, Ziwei Bai, Donghang Su, Bowen Wu, Qun Yu, and Baoxun Wang. 2025. [Improving generalization in intent detection: GRPO with reward-based curriculum sampling](#). *CoRR*, abs/2504.13592.
- Goetz Graefe. 1995. [The cascades framework for query optimization](#). *IEEE Data Eng. Bull.*, 18(3):19–29.
- Goetz Graefe and William J. McKenna. 1993. [The volcano optimizer generator: Extensibility and efficient search](#). In *Proceedings of the Ninth International Conference on Data Engineering, April 19-23, 1993, Vienna, Austria*, pages 209–218. IEEE Computer Society.
- 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, Zhushu Li, Ziyi Gao, Aixin Liu, and 175 others. 2025. [Deepseek-r1 incentivizes reasoning in llms through reinforcement learning](#). *Nature*, 645(8081):633–638.
- Jerry Huang, Siddarth Madala, Risham Sidhu, Cheng Niu, Julia Hockenmaier, and Tong Zhang. 2025. [RAG-RL: advancing retrieval-augmented generation via RL and curriculum learning](#). *CoRR*, abs/2503.12759.
- Prithwish Jana, Piyush Jha, Haoyang Ju, Gautham Kishore, Aryan Mahajan, and Vijay Ganesh. 2024. [Cotran: An llm-based code translator using reinforcement learning with feedback from compiler and symbolic execution](#). In *ECAI 2024: 27th European Conference on Artificial Intelligence, 19–24 October 2024, Santiago de Compostela, Spain—including 13th Conference on Prestigious Applications of Intelligent Systems (PAIS 2024)*, pages 4011–4018. SAGE Publications Pvt. Ltd 1 Oliver’s Yard, 55 City Road, London, EC1Y 1SP.
- Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter A. Boncz, Alfons Kemper, and Thomas Neumann. 2015. [How good are query optimizers, really?](#) *Proc. VLDB Endow.*, 9(3):204–215.
- Feifei Li. 2019. [Cloud native database systems at alibaba: Opportunities and challenges](#). *Proc. VLDB Endow.*, 12(12):2263–2272.

- Zhaodonghui Li, Haitao Yuan, Huiming Wang, Gao Cong, and Lidong Bing. 2024. **LLM-R2: A large language model enhanced rule-based rewrite system for boosting query efficiency**. *Proc. VLDB Endow.*, 18(1):53–65.
- Jie Liu and Barzan Mozafari. 2024. **Query rewriting via large language models**. *CoRR*, abs/2403.09060.
- Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. **Learning word vectors for sentiment analysis**. In *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA*, pages 142–150. The Association for Computer Linguistics.
- PostgreSQL. 2025. PostgreSQL: The world’s most advanced open source database. Retrieved July 1, 2025 from <https://www.postgresql.org>.
- Mohammadreza Pourreza and Davood Rafiei. 2023. **DIN-SQL: decomposed in-context learning of text-to-sql with self-correction**. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.
- Nils Reimers and Iryna Gurevych. 2019. **Sentence-bert: Sentence embeddings using siamese bert-networks**. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 3980–3990. Association for Computational Linguistics.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. **Proximal policy optimization algorithms**. *CoRR*, abs/1707.06347.
- Atsushi Shirafuji, Yusuke Oda, Jun Suzuki, Makoto Morishita, and Yutaka Watanobe. 2023. Refactoring programs using large language models with few-shot examples. In *2023 30th Asia-Pacific Software Engineering Conference (APSEC)*, pages 151–160. IEEE.
- Rajat Singh and Srikanta Bedathur. 2024. **Exploring the use of llms for SQL equivalence checking**. *CoRR*, abs/2412.05561.
- Yewei Song, Saad Ezzini, Xunzhu Tang, Cedric Lothritz, Jacques Klein, Tegawendé F. Bissyandé, Andrey Boytsov, Ulrick Ble, and Anne Goujon. 2024. **Enhancing text-to-sql translation for financial system design**. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP 2024, Lisbon, Portugal, April 14-20, 2024*, pages 252–262. ACM.
- Yuyang Song, Hanxu Yan, Jiale Lao, Yibo Wang, Yufei Li, Yuanchun Zhou, Jianguo Wang, and Mingjie Tang. 2025. **QUITE: A query rewrite system beyond rules with LLM agents**. *CoRR*, abs/2506.07675.
- Zhaoyan Sun, Xuanhe Zhou, Guoliang Li, Xiang Yu, Jianhua Feng, and Yong Zhang. 2025. **R-bot: An llm-based query rewrite system**. *Proc. VLDB Endow.*, 18(12):5031–5044.
- TPC-H Benchmark. 2025. Tpc-h toolkit. https://www.tpc.org/tpc_documents_current_versions/current_specifications5.asp. Accessed: 2025-05-10.
- Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2025. **MAC-SQL: A multi-agent collaborative framework for text-to-sql**. In *Proceedings of the 31st International Conference on Computational Linguistics, COLING 2025, Abu Dhabi, UAE, January 19-24, 2025*, pages 540–557. Association for Computational Linguistics.
- Shuxian Wang, Sicheng Pan, and Alvin Cheung. 2024. **QED: A powerful query equivalence decider for SQL**. *Proc. VLDB Endow.*, 17(11):3602–3614.
- Zhaoguo Wang, Zhou Zhou, Yicun Yang, Haoran Ding, Gansen Hu, Ding Ding, Chuzhe Tang, Haibo Chen, and Jinyang Li. 2022. **Wetune: Automatic discovery and verification of query rewrite rules**. In *SIGMOD ’22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, pages 94–107. ACM.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. **Qwen3 technical report**. *Preprint*, arXiv:2505.09388.
- Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. 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, Brussels, Belgium, October 31 - November 4, 2018*, pages 3911–3921. Association for Computational Linguistics.
- Hongwei Yuan, Xiu Tang, Ke Chen, Lidan Shou, Gang Chen, and Huan Li. 2025. **Cogsql: A cognitive framework for enhancing large language models in text-to-sql translation**. In *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, pages 25778–25786. AAAI Press.
- Fuheng Zhao, Lawrence Lim, Ishtiyaque Ahmad, Divyakant Agrawal, and Amr El Abbadi. 2023a. **Llm-sql-solver: Can llms determine SQL equivalence?** *CoRR*, abs/2312.10321.
- Yue Zhao, Gao Cong, Jiachen Shi, and Chunyan Miao. 2022. **Queryformer: A tree transformer model for query plan representation**. *Proc. VLDB Endow.*, 15(8):1658–1670.

Yue Zhao, Zhaodonghui Li, and Gao Cong. 2023b. [A comparative study and component analysis of query plan representation techniques in ML4DB studies](#). *Proc. VLDB Endow.*, 17(4):823–835.

Qi Zhou, Joy Arulraj, Shamkant B. Navathe, William Harris, and Jinpeng Wu. 2022a. [SPES: A symbolic approach to proving query equivalence under bag semantics](#). In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*, pages 2735–2748. IEEE.

Xuanhe Zhou, Chengliang Chai, Guoliang Li, and Ji Sun. 2022b. [Database meets artificial intelligence: A survey](#). *IEEE Trans. Knowl. Data Eng.*, 34(3):1096–1116.

Xuanhe Zhou, Guoliang Li, Chengliang Chai, and Jianhua Feng. 2021. [A learned query rewrite system using monte carlo tree search](#). *Proc. VLDB Endow.*, 15(1):46–58.

A Experimental Setup and Implementation Details

A.1 Training and Test Data Description

We construct the RL training corpus from the training splits of TPC-H, IMDB (JOB), and DSB, comprising approximately 8,000 source queries in total. For TPC-H and DSB, additional training queries are instantiated from parameterized templates with distinct parameter configurations, while IMDB provides analytical queries with diverse join structures. All training queries are validated to execute successfully on PostgreSQL v14, and their EXPLAIN and EXPLAIN ANALYZE outputs are collected to provide execution feedback for GRPO training.

For each benchmark, the test set contains 500 SQL queries that are strictly disjoint from the training corpus. To prevent data leakage, we eliminate overlap using template IDs for TPC-H and DSB and AST-based comparison for IMDB. All training resources are constructed exclusively from the training split, ensuring that no test query is used during training.

A.2 GRPO Parameter Settings and Reward Sensitivity Analysis

We fine-tune our SQL rewriting model using Group-Relative Policy Optimization (GRPO) with Qwen3-14B and Qwen3-32B as the base models. Training is conducted for a single epoch on approximately 8,000 source SQL queries with a batch size of 256. For each input query, GRPO samples 4 or 8 candidate rewrites and computes group-normalized advantages based on their rewards.

α	Retrieval Time (s)	Improved Queries
0.0	0.07	64
0.3	0.76	71
0.5	0.72	81
0.7	0.73	76
1.0	1.04	78

Table 6: Sensitivity to the retrieval balance coefficient α on the TPC-H benchmark ($k=1$; fixed checkpoint, plan prompting, and demo-pool snapshot).

Reward formulation. The total reward in GRPO training is defined as

$$R_{\text{total}} = \lambda_{\text{eq}} R_{\text{eq}} + \lambda_{\text{exec}} R_{\text{exec}} + \lambda_{\text{perf}} R_{\text{perf}},$$

where R_{eq} evaluates semantic equivalence using our three-stage verification pipeline, R_{exec} indicates whether the rewritten SQL is valid and executable in PostgreSQL, and R_{perf} measures relative performance improvement based on optimizer-estimated cost reduction. In Stage 1, only R_{eq} and R_{exec} are activated to establish correctness, while R_{perf} is introduced in Stage 2 to encourage efficient rewrites. The reward weights are selected through sensitivity analysis to maintain stable training across benchmarks.

Two-stage training strategy. To stabilize GRPO training and avoid premature optimization toward efficiency, we employ a two-stage curriculum. In Stage 1, only the correctness-oriented rewards R_{eq} and R_{exec} are activated (i.e., $\lambda_{\text{perf}}=0$), guiding the model to generate executable and semantically equivalent rewrites. Once correctness becomes stable, we introduce the performance reward R_{perf} in Stage 2 to further optimize query efficiency. This staged optimization allows the model to establish reliable correctness before introducing performance-oriented feedback, reducing conflicts among reward objectives during training.

A.3 Sensitivity of the Demonstration Retrieval Parameter

We further examine how the retrieval balance coefficient α affects the retriever’s efficiency and the quality of retrieved demonstrations. Table 6 reports the sensitivity of E³-Rewrite to the retrieval balance coefficient α on the TPC-H benchmark. As the table shows, retrieval latency remains stable for intermediate values of α (0.3–0.7), while purely

semantic retrieval ($\alpha=0$) is fastest since it only requires embedding-based similarity, and purely structural retrieval ($\alpha=1.0$) is the slowest due to costly tree-edit distance computations.

The number of improved queries peaks around $\alpha=0.5$, confirming that a balanced combination of structural and semantic similarity provides the most effective demonstrations for rewriting. It is worth noting that both TPC-H and DSB queries are template-generated, leading to high structural redundancy where structural cues dominate; in contrast, IMDB queries are manually written and structurally diverse, where semantic similarity contributes more substantially to retrieval quality.

B Further Analysis

To further assess the generalization capability of E³-Rewrite, we evaluate its performance on the DSB benchmark using models trained on a different dataset. Specifically, we compare two variants: one trained directly on DSB (in-domain), and one trained on the TPC-H dataset without any additional fine-tuning. As shown in Table 9, the model trained on TPC-H still achieves strong performance on DSB, with only a modest drop in average latency (from 16.93s to 17.44s), improved queries (151 to 146), and equivalence ratio (98.8% to 97.4%).

B.1 Rewriting Latency Analysis

Method	TPC-H	IMDB	DSB
LearnedRewrite	0.99	0.14	1.23
LLM-R2	6.36	2.97	8.71
R-Bot	3.75	1.51	5.94
E ³ -Rewrite	3.91	1.61	5.87

Table 8: Average rewriting latency per query (seconds).

We compare the average rewriting latency of different methods across TPC-H, IMDB, and DSB in Table 8. E³-Rewrite achieves average latencies of 3.91s, 1.61s, and 5.87s on TPC-H, IMDB, and DSB, respectively. It consistently outperforms LLM-R2 in latency and remains comparable to R-Bot across all three benchmarks. LearnedRewrite is the fastest method, with latencies of 0.99s, 0.14s, and 1.23s, owing to its rule-based search paradigm. Overall, E³-Rewrite maintains practical rewriting efficiency while providing the flexibility of direct

query generation and execution-aware optimization.

B.2 Cross-Dataset Robustness

Training Dataset	Avg Latency (s)	Improved Queries	Equiv. Ratio
DSB (in-domain)	16.93	151	98.8%
TPC-H (cross-domain)	17.44	146	97.4%

Table 9: Cross-dataset robustness on DSB: performance of E³-Rewrite trained on TPC-H vs. trained on DSB.

These results demonstrate that E³-Rewrite learns transferable rewriting behaviors that generalize across benchmarks with different query styles and schema characteristics. This cross-dataset robustness highlights the framework’s potential for deployment in unseen workloads without retraining.

B.3 Model Scale Analysis on DSB

Model	Avg Latency improvement(s)	Improved Queries	Equiv. Ratio
E ³ -Rewrite (Qwen 14B)	38.83 → 28.39	86	89.4%
E ³ -Rewrite (Qwen 32B)	38.83 → 16.93	151	98.8%

Table 10: Rewrite quality of E³-Rewrite under different model scales on DSB.

We further evaluate the impact of model scale on the DSB benchmark, which contains highly diverse, deeply nested decision-support queries. As shown in Table 10, E³-Rewrite with Qwen 14B yields 86 improved rewrites and an 89.4% equivalence ratio. While this confirms that our method remains functional at moderate scale, the performance gap to larger models is pronounced on DSB.

Scaling up to Qwen 32B significantly enhances all metrics: the number of improved rewrites nearly doubles (151 vs. 86), equivalence ratio rises to 98.8%, and average latency drops from 28.39s to 16.93s. We attribute these gains not only to increased model capacity, but also to DSB’s inherent structural complexity: many queries involve multi-hop joins, correlated subqueries, and domain-specific aggregation logic that challenge smaller models’ planning and generalization abilities.

Overall, these results suggest that while E³-Rewrite is robust even under moderate model sizes,

Dataset	Total	Equiv. Tool		LLM Check		Exec Compare	
		#Finalized (%)	Avg (s)	#Finalized (%)	Avg (s)	#Finalized (%)	Avg (s)
TPC-H	500	422 (84.4)	0.72	56 (11.2)	1.26	22 (4.4)	16.98
IMDB	500	492 (98.4)	0.15	5 (1.0)	0.73	3 (0.6)	2.35
DSB	500	398 (79.6)	0.88	64 (12.8)	1.47	38 (7.6)	9.47

Table 7: Stage-wise equivalence verification results of E³-Rewrite across datasets.

structurally complex workloads such as DSB demand larger models to fully realize the benefits of reward-aligned optimization and plan-sensitive rewriting.

B.4 Dynamic Demonstration Pool Update Analysis

Setting	Equiv. Ratio	Avg. Latency (s)	Improved Queries
w/o Demos	96.4%	35.39	177
Static Demos	97.2%	33.48	186
Dynamic Update	99.6%	29.67	210

Table 11: Effect of dynamic demonstration pool update on TPC-H.

To isolate the contribution of the dynamic demonstration pool update, we compare three variants on the TPC-H benchmark: (A) removing demonstrations entirely, (B) using a static demonstration pool without updates, and (C) our full dynamic pool update strategy. As shown in Table 11, static demonstrations already improve over the no-retrieval setting, increasing the equivalence ratio from 96.4% to 97.2%, reducing average latency from 35.39s to 33.48s, and increasing the number of improved queries from 177 to 186. More importantly, enabling dynamic pool updates further improves all metrics, achieving 99.6% equivalence, 29.67s average latency, and 210 improved queries. These results indicate that the dynamic update mechanism enriches the demonstration pool with high-quality executable and equivalent rewrites, thereby providing more effective guidance for subsequent queries.

B.5 Equivalence Verification Efficiency and Stage Analysis

The equivalence verification process of E³-Rewrite follows a three-stage pipeline combining symbolic reasoning, semantic evaluation, and execution-based validation. As shown in Table 7, SQLSolver

and QED resolve 84.4%, 98.4%, and 79.6% of queries on TPC-H, IMDB, and DSB, respectively, with sub-second average latency. Queries not finalized by the symbolic checker are passed to the LLM-based semantic checker, which resolves an additional 11.2%, 1.0%, and 12.8% of cases. Only 4.4%, 0.6%, and 7.6% of queries require execution-level comparison.

Execution-based verification is more expensive, with average latencies of 16.98s, 2.35s, and 9.47s on TPC-H, IMDB, and DSB, respectively. Overall, the hierarchical design allows most queries to be resolved by the lower-cost symbolic and semantic stages, while reserving execution comparison for a small fraction of difficult cases.

B.6 Cost-Latency Consistency Analysis

Our RL objective uses optimizer-estimated cost as a lightweight training signal, while our primary evaluation metric is measured runtime latency. To quantify their consistency, for each rewrite pair (q, q') we compute the *relative* improvements:

$$\Delta C = \frac{\text{cost}(q) - \text{cost}(q')}{\text{cost}(q)}, \quad (6)$$

$$\Delta T = \frac{T(q) - T(q')}{T(q)}, \quad (7)$$

where $\text{cost}(\cdot)$ is obtained from EXPLAIN and $T(\cdot)$ is the end-to-end execution time under the same environment. We then measure the absolute gap $e = |\Delta C - \Delta T|$ and report its distribution over rewrite pairs.

Dataset	#Pairs	$e < 0.05$	$[0.05, 0.10)$	≥ 0.10
TPC-H	500	492 (98.4%)	8 (1.6%)	0 (0%)
IMDB	500	488 (97.6%)	6 (1.2%)	6 (1.2%)
DSB	500	472 (94.4%)	26 (5.2%)	2 (0.4%)

Table 12: Distribution of the gap between relative cost and latency improvements.

As shown in Table 12, the vast majority of

rewrite pairs exhibit a very small gap between estimated cost reduction and measured latency improvement. Across all benchmarks, over 94% of rewrites fall within $e < 0.05$, indicating that optimizer-estimated cost closely tracks runtime behavior in practice. The few cases with larger gaps are primarily observed on more complex workloads, which we attribute to optimizer estimation noise and runtime system variability. Overall, this analysis supports the use of cost as a lightweight and reasonably aligned proxy during training, while final evaluation is conducted using measured latency.

C Case Study

C.1 TPC-H Correlated Subquery

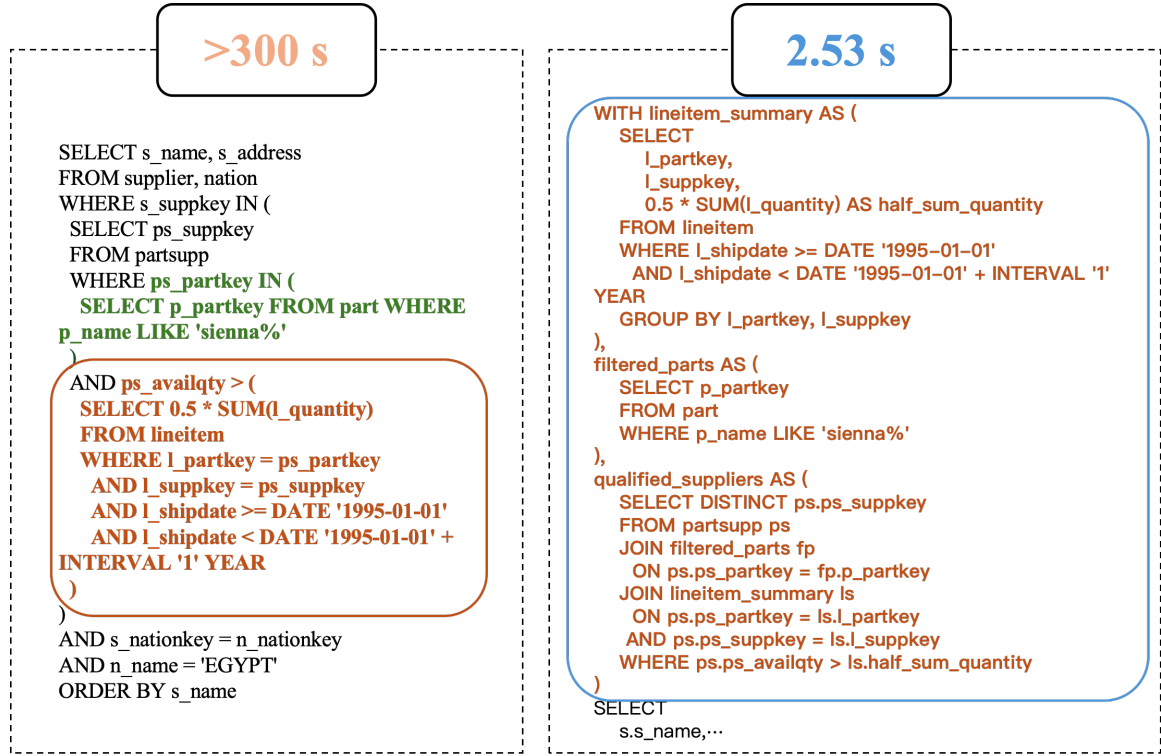
Figure 6 shows a TPC-H query that retrieves suppliers from Egypt who provide parts with names starting with `sienna%`, where the available quantity exceeds half of the shipped quantity for the same part-supplier pair in 1995. The original query contains nested correlated subqueries over `part`, `partsupp`, `lineitem`, and `supplier`, causing repeated aggregation and redundant scans.

E³-Rewrite converts the correlated subqueries into CTEs that pre-filter relevant parts and pre-aggregate shipment quantities by part and supplier. This transformation replaces repeated subquery evaluation with explicit joins, allowing the optimizer to reuse intermediate results and generate a more efficient execution plan. As a result, the execution time is reduced from over 300 seconds to 2.53 seconds.

C.2 DSB Scalar Subquery

Figure 7 shows a DSB query that finds Illinois customers whose 1998 catalog-return amount exceeds 120% of the state-level average. The original query relies on a nested scalar subquery over `customer_total_return`, resulting in repeated scans of identical aggregated results.

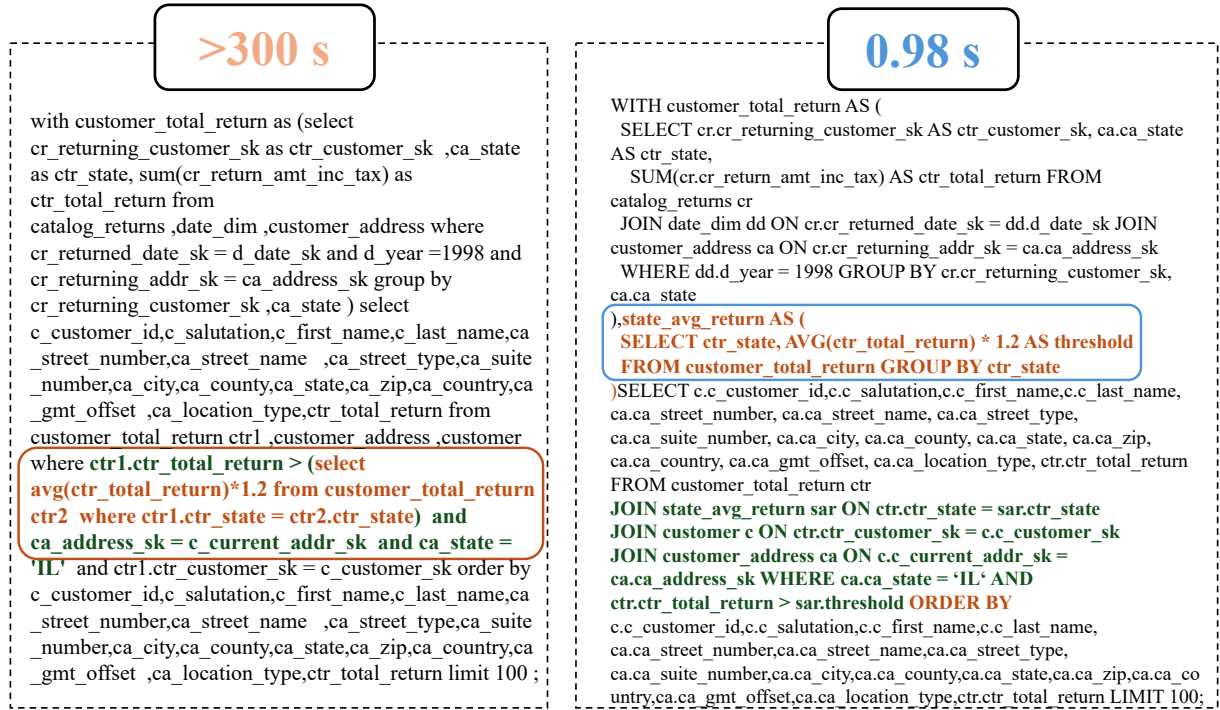
E³-Rewrite restructures the query using two CTEs: `customer_total_return`, which aggregates return amounts by customer and state, and `state_avg_return`, which computes the state-level threshold. By materializing these intermediate aggregations and joining them explicitly, the rewritten query avoids repeated scalar-subquery evaluation and exposes a clearer join structure to the optimizer. This reduces the runtime from over 300 seconds to 0.98 seconds.



(a) Original SQL

(b) Our Rewrite

Figure 6: Example of correlated subquery rewriting on the TPC-H benchmark. E³-Rewrite transforms the nested subqueries into pre-aggregated CTEs with predicate pushdown, reducing execution time from over 300 s to 2.53 s.



(a) Original SQL

(b) Our Rewrite

Figure 7: Example of scalar subquery rewriting on the DSB benchmark. E³-Rewrite replaces the nested comparison with two CTEs for per-customer and per-state aggregation, reducing execution time from over 300 s to 0.98 s.