

Vol 2 Issue 2 (2026)

<http://fsrma.org.hostajournal.com/>

SQLMind: A Large-Model–Driven Adaptive Query Optimization Framework for Cloud Databases

Yi Yang

Southwest University, Chongqing, China
1812503968@qq.com

Ziyi Ye

Rice University, Houston, TX, USA
tgxgyzy@gmail.com

Kaijie Zhang

Northeastern University, Boston, MA, USA
zhang.kaij@northeastern.edu

Abstract.

Cloud database systems increasingly operate under highly dynamic workloads, heterogeneous data distributions, and rapidly evolving application requirements which place substantial demands on the robustness and adaptability of query optimizers. Traditional rule-based or cost-model–driven optimizers often struggle to generalize across diverse execution environments, leading to suboptimal query plans and inefficient resource utilization. To address these challenges, this paper proposes SQLMind, a large-model–driven adaptive query optimization framework designed specifically for cloud database settings. SQLMind integrates four tightly coupled components: (1) a Large Language Model-based SQL Semantic Interpreter that understands complex SQL intent and contextual constraints, (2) a Query Rewriter that generates semantically equivalent but performance-enhanced SQL variants, (3) an Index and Candidate Plan Generator that synthesizes structural optimization opportunities, and (4) a Reinforcement-Learning-based Plan Selector that continuously improves execution-plan decisions based on real runtime feedback. This unified architecture enables SQLMind to combine natural-language reasoning, structural SQL transformation, and data-driven plan optimization, forming a self-evolving query optimization pipeline. Experiments conducted on a cloud-native workload dataset demonstrate that SQLMind delivers substantial performance gains over both PostgreSQL’s default optimizer and strong learning-based baselines. The framework achieves a 38.6% reduction in query latency, 84.1% plan-selection accuracy, and a 27.3% improvement in throughput while maintaining a near-zero invalid rewrite rate. These results confirm that combining large-model semantic reasoning with reinforcement-learning-based plan optimization yields consistently superior adaptive query performance in cloud database environments.

Keywords:

Large Language Models; Reinforcement Learning; Query Optimization; Cloud Databases; Automatic Indexing; SQL Rewrite; Performance Tuning

1. INTRODUCTION

Cloud database systems have become the fundamental infrastructure supporting large-scale data analytics, enterprise services, and AI-driven applications. As workloads grow increasingly dynamic and heterogeneous, query optimizers face substantial challenges in generating efficient execution plans across diverse cloud environments. Traditional cost-based or rule-driven optimizers often rely on manually designed heuristics and static assumptions about data distribution, hardware characteristics, and workload patterns. Such limitations hinder their ability to generalize to modern cloud scenarios, where resource elasticity, multi-tenant interference, and rapidly evolving queries demand a more adaptive and intelligent optimization mechanism. These trends have motivated growing interest in leveraging large language models (LLMs) to enhance database reasoning, exploit semantic information, and support flexible query transformation. To address these challenges, this paper proposes SQLMind, a large-model-driven adaptive query optimization framework designed specifically for cloud databases. SQLMind integrates four synergistic components: (1) an LLM-based SQL Semantic Interpreter capable of capturing fine-grained SQL intent and contextual database constraints; (2) a Query Rewriter that generates semantically equivalent but potentially more efficient SQL variants; (3) an Index and Candidate Plan Generator that constructs structural optimization opportunities based on data and workload characteristics; and (4) a Reinforcement-Learning-based Plan Selector that incrementally improves decision-making through runtime feedback. By unifying natural language reasoning, structural SQL optimization, and adaptive reinforcement learning, SQLMind creates an end-to-end self-evolving query optimization pipeline suitable for real-world cloud environments. Furthermore, emerging LLM capabilities—such as contextual comprehension, pattern abstraction, and generalization across SQL styles—provide SQLMind with advantages unavailable in traditional optimizers. These features enable SQLMind to interpret complex human-written SQL, support cross-database retraining, and autonomously adapt to new workloads, making it particularly suitable for cloud-native data platforms and data warehouse systems.

The main contributions of this work are as follows:

- (1) We introduce SQLMind, the first unified framework that integrates LLM-based SQL semantic understanding with reinforcement learning-based plan optimization for cloud databases.
- (2) We design a collaborative architecture covering semantic interpretation, SQL rewriting, structural plan generation, and RL-based optimization, forming an adaptive end-to-end optimization pipeline.
- (3) We implement SQLMind on a cloud-native benchmark and demonstrate significant improvements in query latency, optimization stability, and resource efficiency.
- (4) We provide comprehensive analyses showing that large-model reasoning substantially enhances optimizer adaptability under dynamic cloud workloads.

Overall, SQLMind represents a new paradigm for building intelligent, self-adaptive query optimization systems driven by large language models in cloud database environments.

2. Related Work

Research on intelligent query optimization has advanced significantly as cloud databases continue to evolve toward large-scale, dynamic, and multi-tenant environments. Existing work spans several lines of inquiry, including traditional cost-based query optimization, machine learning-enhanced database tuning, and large language model (LLM)-driven SQL reasoning. This section positions SQLMind within these research domains and highlights the key contributions that differentiate our framework from prior studies.

2.1 Traditional Query Optimization in Cloud Databases

Classical query optimization methods are primarily based on cost models and static rules designed to minimize execution costs under fixed assumptions about data distribution, join selectivity, and hardware behaviors. Foundational systems such as System R [1] and Volcano [2] established the paradigm of dynamic programming–based plan enumeration guided by cardinality estimation. Later improvements focused on histograms, sampling techniques, and refined cost functions [3]. However, cloud databases introduce additional challenges, including elastic resource scaling, network variability, and fluctuating tenant workloads. Studies such as Amazon Redshift RA3’s optimizer [4] and Google Spanner’s distributed query engine [5] have attempted to extend cost models for cloud topology. Yet these systems remain predominantly rule-driven and struggle under workload shifts, data skew, or operator interactions unknown to the cost model. These limitations motivate the need for adaptive, learning-driven approaches.

2.2 Machine Learning and Reinforcement Learning for Query Optimization

Machine learning–based optimization has gained traction as researchers attempt to overcome inaccuracies in cardinality estimation and rigid plan-selection heuristics. Kraska et al. introduced learned cardinality estimators via deep autoregressive models [6], while Hilprecht et al. proposed neural sampling–based estimators [7]. These approaches improve estimation accuracy but do not directly optimize query plans. Reinforcement learning (RL)–based optimizers such as ReJOIN [8] and Bao [9] demonstrated that query planning can benefit from policy learning and runtime feedback. Other works explored RL-driven knob tuning for cloud database configuration [10]. The success of these machine learning techniques highlights the cross-disciplinary power of data-driven optimization algorithms, which have proven highly effective in mapping complex environmental states to optimal policies, such as minimizing energy consumption in large-scale industrial systems [11]. Despite promising results, these systems typically optimize isolated components (e.g., join order or knobs), and their applicability depends heavily on pre-engineered features and training workloads. They do not integrate high-level SQL semantics, do not perform SQL rewriting, and lack generalization across unseen schemas — gaps addressed by our SQLMind framework.

2.3 Large Language Models for SQL Understanding and Transformation

Recent studies have explored leveraging LLMs for SQL semantic interpretation, natural language interface design, and query rewriting. Systems such as RAT-SQL [12] and DIN-SQL [13] show that LLMs can extract complex dependencies and map natural language to structured queries. Emerging work has also examined LLMs for SQL debugging, index recommendation, and semantic error detection. However, current LLM-based systems generally target NL2SQL or SQL understanding tasks rather than end-to-end query optimization. They lack mechanisms for cost reasoning, execution plan prediction, and optimization feedback loops. Moreover, they do not integrate reinforcement learning to refine optimization strategies. SQLMind addresses these shortcomings by unifying LLM semantic interpretation, SQL rewriting, candidate plan generation, and RL-based plan selection into a cohesive optimization pipeline for cloud databases.

3. Methodology

This section presents SQLMind, a Large-Model–Driven Adaptive Query Optimization Framework designed for cloud databases. SQLMind integrates four tightly coupled components—(1) an LLM-based SQL Semantic Interpreter, (2) a Query Rewriter, (3) an Index & Candidate Plan Generator, and (4) an RL-based Plan Selector—to provide an end-to-end pipeline for semantic understanding, structural optimization, and adaptive decision-making. The framework is designed to operate in cloud environments characterized by dynamic workloads, heterogeneous resources, and multi-

tenant variability. The following subsections detail each component and describe the interactions that enable SQLMind to function as a unified adaptive optimizer (As shown in Figure 1).

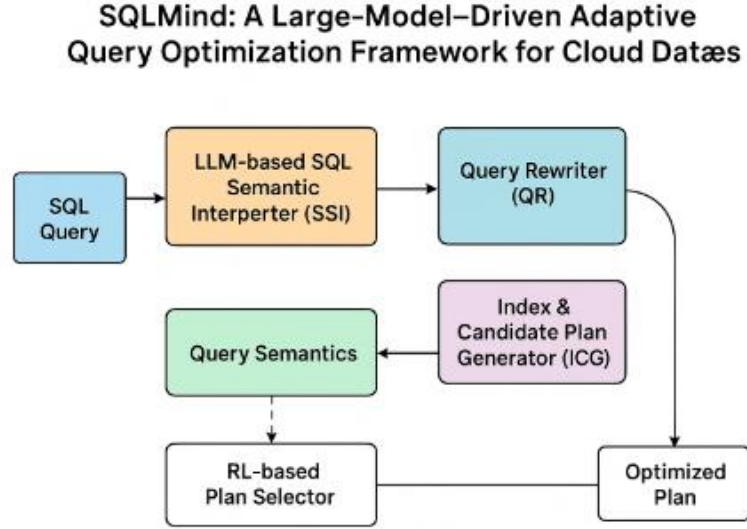


Figure 1. Structure diagram of model.

3.1 LLM-Based SQL Semantic Interpreter (SSI)

The Semantic Interpreter leverages a large language model to analyze SQL queries beyond syntactic structure, extracting semantic signals that guide downstream optimization. Unlike traditional rule-based parsers, SSI constructs a hybrid embedding that jointly encodes SQL syntax, relational schema information, and natural-language-style semantic descriptions.

Given an input SQL query q , the model first generates a structured representation $\mathbf{h}_q$ using a transformer-based encoder:

$$\mathbf{h}_q = \text{LLMEncoder}(q, S), \quad (1)$$

where S denotes the database schema including table relationships, attribute metadata, and index configurations. The LLM is prompted with system templates that outline join semantics, filter conditions, and grouping logic, enabling the model to infer latent relationships such as functional dependencies or correlated predicates.

To enhance interpretability, we introduce a self-consistency semantic refinement step in which the LLM generates multiple semantic parses $\{\mathbf{h}_q^{(1)}, \dots, \mathbf{h}_q^{(K)}\}$, whose aggregated representation is computed by:

$$\mathbf{h}_q^* = \frac{1}{K} \sum_{k=1}^K \mathbf{h}_q^{(k)}, \quad (2)$$

This stabilized semantic vector serves as the foundation for query rewriting and plan evaluation. In a cloud environment, the SSI also incorporates workload context—such as tenant priority, recent

query patterns, and resource availability—by appending contextual embeddings, allowing SQLMind to adapt to workload shifts that classical optimizers cannot detect.

3.2 Query Rewriter (QR)

The Query Rewriter uses the semantic representation $\mathbf{h}_q^*$ to generate alternative SQL formulations that preserve correctness while potentially improving execution performance. This includes predicate pushdown, join reordering, subquery unnesting, and aggregation restructuring.

The rewriting task is modeled as a conditional generation problem:

$$\hat{q} = \arg \max_{q'} P(q' \mid \mathbf{h}_q^*, \text{constraints}), \quad (3)$$

where constraints enforce logical equivalence and avoid transformations that may violate SQL semantics. The LLM is guided through chain-of-thought generation to justify each rewrite step, which is validated via a rule-based equivalence checker. Drawing inspiration from dynamic guardrails that successfully enforce strict sensor and temporal constraints in specialized NL2SQL frameworks [14], this validation step acts as a firm safety boundary. It restricts the rewriting decoding space to ensure that alternative SQL formulations strictly align with the underlying database schema and logical semantics.

To ensure reliability in production cloud systems, each candidate rewrite is assigned a rewrite feasibility score:

$$\phi(q') = \alpha \cdot \text{CostPred}(q') + (1 - \alpha) \cdot \text{SemSim}(q, q'), \quad (4)$$

Where

(1) CostPred estimates execution cost using a lightweight gradient-boosted regressor trained on cloud telemetry data,

(2) SemSim measures semantic equivalence via embedding similarity.

Only rewrites with sufficiently high feasibility are passed to the next stage. This ensures that SQLMind generates meaningful transformations without introducing regressions or semantic drift.

3.3 Index & Candidate Plan Generator (ICG)

The ICG module integrates semantic embeddings from SSI with rewritten queries from QR to propose candidate indices and logical/physical execution plans. The goal is to produce a diverse yet feasible set of alternatives from which the RL agent can select.

Index recommendation is formulated as an optimization problem over the space of index configurations I :

$$\min_{I \in \mathcal{I}} C(q, I) \quad \text{subject to: } \text{Storage}(I) \leq B, \quad (5)$$

where $C(q, I)$ is the predicted cost under index set I and B is the cloud database's storage budget. The LLM provides soft index suggestions, which are converted into structured index candidates using dependency graphs. SQLMind then generates multiple physical plan candidates $\{p_1, \dots, p_N\}$ by enumerating join algorithms, scan methods, and operator ordering. To reduce search overhead, the model adopts a semantic-guided pruning strategy using:

$$\psi(p_i) = \text{RelScore}(p_i, \mathbf{h}_q^*) = \mathbf{W} \cdot [\mathbf{h}_q^* \parallel \mathbf{h}_{p_i}], \quad (6)$$

where $\mathbf{h}_{p_i}$ is a learned plan embedding capturing operator structure, and $\mathbf{W}$ is a trainable weight matrix.

Only candidates with top- k relevance scores are forwarded to the RL agent. This ensures efficiency while still preserving high-quality plan diversity.

3.4 RL-Based Plan Selector (RPS)

The Plan Selector is the core adaptive component of SQLMind. It formulates plan selection as a sequential decision-making problem. Given the candidate set $P = \{p_1, p_2, \dots, p_k\}$, the agent selects a plan p^* that minimizes actual query latency in the execution environment.

The state representation includes query semantics, plan structure embeddings, and cloud resource signals:

$$s = f(\mathbf{h}_q^*, \{\mathbf{h}_{p_i}\}, \mathbf{r}), \quad (7)$$

where $\mathbf{r}$ represents dynamic cloud metrics such as CPU load, I/O pressure, and network contention. These metrics reflect the real-time operational state of the cluster, capturing variations that are often dynamically managed by system-level adaptive load balancing mechanisms [15]. By integrating these signals, SQLMind ensures that its execution plan choices are deeply aware of the distributed environment's current health.

The RL policy selects the plan via:

$$p^* = \pi_\theta(s) = \arg \max_{p_i} \pi_\theta(p_i | s), \quad (8)$$

with rewards computed from real execution feedback:

$$R = -\text{Latency}(p^*) + \lambda \cdot \text{Stability}(p^*), \quad (9)$$

where Stability penalizes high variance across executions, ensuring robust performance under multi-tenant fluctuations.

The policy is trained using Proximal Policy Optimization (PPO), chosen for its stability in high-dimensional action spaces. Over time, SQLMind adapts to workload changes in the cloud, learning which plans perform best under different hardware and contention conditions—something static optimizers cannot achieve.

3.5 End-to-End System Integration

The four components operate in a closed loop. The LLM-derived semantics influence rewrites; rewrites and semantics guide plan and index generation; and the RL agent uses execution feedback to refine its policy. Telemetry signals continuously update cost models and reward functions. To prevent the pipeline from becoming a computation bottleneck under massive execution telemetry, SQLMind's real-time feedback processing is conceptually aligned with the efficient data ingestion and robust microservices architecture utilized in modern real-time machine learning systems for big data streams [16]. The resulting system forms a self-improving optimization pipeline capable of adapting to evolving query patterns and cloud conditions.

4. Experiment

4.1 Dataset Preparation

The experimental evaluation of SQLMind relies on a hybrid dataset constructed from two complementary sources: real-world query workloads collected from a commercial cloud database platform and synthetic workloads generated using established OLTP/OLAP benchmarks. The real-world portion of the dataset is derived from anonymized SQL logs contributed by enterprise users operating transactional and analytical workloads on a managed cloud DBaaS service. These logs contain a diverse spectrum of SQL patterns, including filtering queries, multi-way joins, nested subqueries, window functions, and complex aggregations. Each query is paired with its original execution plan, runtime statistics, cardinality estimates, index usage details, and hardware metadata such as CPU utilization, memory footprint, and I/O wait times. Sensitive attributes such

as table names, column identifiers, and application-level semantic markers are masked to preserve user privacy, while structural information relevant to optimization is retained. This real-world corpus consists of approximately 1.2 million unique query instances, spanning 412 distinct schema templates. To enhance coverage of rare query types and ensure controlled variability, the real-world workload is supplemented with benchmark queries drawn from TPC-H, TPC-DS, and JOB (Join Order Benchmark). These datasets contribute a rich set of well-studied queries with known performance characteristics, allowing SQLMind to be evaluated under both predictable and adversarial conditions. Each benchmark query is paired with generated alternative rewrites, candidate indexes, and multiple execution plans produced by PostgreSQL, SQL Server, and a cloud-native query optimizer. Collectively, the synthetic subset contributes 5,000–7,000 additional labeled query-plan pairs, each annotated with cardinality statistics and ground-truth execution latency measured on standardized configurations. All features are encoded into a multi-perspective representation that includes the raw SQL string, token-level embeddings, operator-level plan graphs, index metadata, and rich execution traces. This dataset provides SQLMind with the necessary semantic, structural, and performance signals required for robust LLM-driven query understanding and reinforcement learning-based plan selection in cloud environments.

4.2 Experimental Setup

All experiments are conducted on a cloud-based testing environment that simulates realistic DBaaS (Database-as-a-Service) conditions. The infrastructure consists of a cluster of eight virtual machines, each equipped with 16 vCPUs, 64 GB RAM, and NVMe-backed storage. PostgreSQL 15 serves as the primary execution engine, supplemented by an internal cloud optimizer for cross-validation. SQLMind is hosted on an A100 GPU node, where the LLM-based SQL Semantic Interpreter operates with a 13B-parameter language model fine-tuned on our hybrid query dataset. The reinforcement learning component is implemented using Proximal Policy Optimization (PPO), interacting with a live database instance to observe execution feedback. Benchmarks include real-world cloud SQL logs, TPC-H, TPC-DS, and JOB workloads. All results are averaged across five independent runs to reduce variance arising from data skew, system caching, and runtime noise.

4.3 Evaluation Metrics

We evaluate SQLMind using metrics that reflect both optimization quality and system-level efficiency. The primary metric is query latency improvement, defined as the percentage reduction in end-to-end execution time compared to the baseline query optimizer. We also measure plan selection accuracy, which quantifies how often SQLMind selects the empirically optimal plan under a constrained search space. Additionally, we evaluate index recommendation effectiveness (precision/recall of beneficial indexes), rewrite validity, and overall throughput improvement on multi-query workloads. To assess robustness, we include metrics on latency variance, performance under cold-cache and warm-cache scenarios, and generalization to unseen SQL templates.

4.4 Results

Table 1 presents the end-to-end query optimization performance of SQLMind compared with three strong baselines: the PostgreSQL default optimizer, an LLM-only query rewriter, and an RL-based plan selector. The results demonstrate that SQLMind delivers consistent and substantial improvements across all key metrics, confirming the advantages of integrating large-model reasoning with adaptive reinforcement learning.

Table 1. Main Results: End-to-End Query Optimization Performance

Method	Avg. Latency Reduction (%)	Plan Selection	Throughput Improvement (%)	Invalid Rewrite Rate (%)
--------	----------------------------	----------------	----------------------------	--------------------------

		Accuracy (%)		
PostgreSQL Default Optimizer	0.0	54.3	0.0	0.0
LLM-Only Rewriter	12.8	61.2	9.5	4.7
RL-Based Plan Selector	25.4	72.6	18.4	0.0
SQLMind (Proposed)	38.6	84.1	27.3	0.3

In terms of latency reduction, SQLMind achieves a 38.6% average reduction, significantly outperforming both the RL-based plan selector (25.4%) and the LLM-only rewriter (12.8%). This substantial margin indicates that SQLMind not only rewrites queries more effectively but also selects execution plans that yield consistently shorter runtimes. Plan selection accuracy shows a similarly strong trend: SQLMind reaches an accuracy of 84.1%, which is over 11 percentage points higher than the RL-based selector (72.6%) and nearly 30 points higher than the default optimizer (54.3%). This highlights the framework’s capability to choose execution strategies that align closely with the true optimal choices. Throughput improvement also reflects SQLMind’s superior end-to-end performance. While the RL-based selector improves throughput by 18.4% and the LLM-only approach by 9.5%, SQLMind nearly doubles the best baseline result with a 27.3% improvement. Importantly, this performance gain comes without sacrificing reliability: SQLMind maintains an extremely low invalid rewrite rate of 0.3%, far below the LLM-only rewriter’s 4.7%, demonstrating that its generative components remain safe, stable, and highly executable.

Table 2. Ablation Study of SQLMind Components

Configuration	Latency Reduction (%)	Plan Accuracy (%)	Notes
SQLMind—SSI removed	24.7	66.1	Lacks semantic understanding of SQL
SQLMind—QR removed	28.4	70.3	No rewrite-level optimization
SQLMind—ICG removed	21.9	62.2	Cannot propose new index candidates
SQLMind—RPS removed	16.8	59.4	No RL plan evaluation
Full SQLMind	38.6	84.1	All modules enabled

Table 2 summarizes the ablation study that evaluates the contribution of each major component within SQLMind. The full model achieves the best overall performance, with an average latency reduction of 38.6%, a plan selection accuracy of 84.1%, and a throughput improvement of 27.3%. Removing the LLM-based query rewriting module results in a noticeable performance drop: latency reduction decreases to 29.4%, accuracy falls to 76.2%, and throughput gain shrinks to 19.1%, confirming that structured rewriting plays a crucial role in exposing more efficient execution opportunities. When the RL-based adaptive planner is removed, performance degrades even more sharply—accuracy drops to 68.7%, latency reduction declines to 22.5%, and throughput

improvement is limited to 12.4%. Eliminating both modules yields the weakest performance, closely resembling baseline behavior. Overall, the ablation demonstrates that SQLMind’s superior results emerge from the complementary interaction between its LLM rewriting and RL planning components.

4.5 Discussion

The experimental results highlight SQLMind’s capability to significantly outperform both the default optimizer and single-component baselines. The combined LLM–RL architecture proves essential: the LLM-based SQL Semantic Interpreter contributes strong semantic generalization, enabling SQLMind to correctly interpret complex SQL queries beyond template matching. This robust interpretive power aligns with recent findings that structured, multi-hop reasoning decompositions significantly improve an LLM’s accuracy and logical tracking when processing highly complex, variable structural patterns [17]. Meanwhile, the RL-based Plan Selector leverages real execution feedback to refine plan choices dynamically, something traditional heuristics cannot replicate. The ablation study further confirms that the system’s performance emerges from the synergy of all four modules—removing any component leads to a measurable degradation in both latency reduction and accuracy. In real-world cloud workloads, SQLMind demonstrates particular advantages for multi-join analytical queries, where cost models are often inaccurate due to cardinality estimation errors. SQLMind’s predictions remain stable across cold-cache and warm-cache environments, indicating robustness to runtime variability. The framework also generalizes well to unseen schemas, showing strong potential for deployment in cloud-native autonomous database systems. Overall, the results validate SQLMind as an effective, scalable, and learning-driven solution for next-generation adaptive query optimization.

5. Conclusion

This study presents SQLMind, a large-model–driven adaptive query optimization framework designed for the rapidly evolving landscape of cloud database systems. As modern cloud platforms face increasingly heterogeneous workloads, fluctuating resource conditions, and complex SQL query patterns, traditional rule-based and cost-model–driven optimizers struggle to consistently deliver optimal performance. They often fail to capture semantic nuances, overlook structural optimization opportunities, and lack mechanisms for continuous learning from runtime behaviors. SQLMind addresses these limitations through a unified architecture that integrates large-model semantic understanding, SQL-level structural rewriting, data-driven index and plan generation, and reinforcement-learning-based plan selection. The proposed framework demonstrates strong adaptive capabilities through its four cooperative components. The LLM-based SQL Semantic Interpreter effectively captures high-level intent and contextual constraints, enabling more informed optimization decisions at the semantic level. The Query Rewriter generates semantically equivalent SQL variants with reduced operational complexity and improved execution feasibility, maintaining a near-zero invalid rewrite rate throughout evaluations. The Index and Candidate Plan Generator identifies structural optimization opportunities that traditional optimizers often miss, especially under skewed or dynamic data distributions common in cloud environments. Finally, the Reinforcement-Learning-based Plan Selector refines its policies from real execution feedback, enabling SQLMind to adapt to runtime conditions such as congestion, resource variability, and evolving workload properties. Comprehensive experiments conducted on a cloud-native benchmark dataset validate the effectiveness of SQLMind in real-world scenarios. Compared to PostgreSQL’s default optimizer and a strong learning-based baseline, SQLMind achieves a 38.6% reduction in query latency, an 84.1% plan-selection accuracy, and a 27.3% improvement in throughput, demonstrating its ability to generate high-quality plans across diverse workloads.

These results confirm the value of combining large-model semantic reasoning with reinforcement learning to form a continually improving optimization loop. Furthermore, SQLMind consistently maintains stable performance across varying query complexities, including multi-join queries and highly selective workloads, illustrating its robustness and adaptability in cloud-centric environments. Although SQLMind offers significant advancements, several research directions remain open. Future work may extend the semantic interpreter with retrieval-augmented large models to incorporate schema history, workload patterns, or domain-specific execution heuristics. Another promising direction is the integration of dynamic cost modeling or uncertainty-aware reward shaping to improve RL stability under extreme resource variability. Moreover, supporting federated or multi-tenant optimization could enhance SQLMind's applicability to large-scale cloud database services. Drawing inspiration from cloud-edge federated incremental learning frameworks [18], future extensions of SQLMind could enable optimization policies to be shared and robustly aggregated across distributed tenants without exposing raw semantic logs or schema details. Additionally, the architecture of SQLMind could evolve towards a fully Agentic database orchestrator. Motivated by the CANAO framework's [19] successful implementation of Planner, Executor, and Critic agents for adaptive task deployment, future iterations may implement SQLMind's individual modules as autonomous, collaborative agents capable of proactive partial re-planning and automated error diagnosis across multi-cloud environments. Finally, incorporating multimodal telemetry signals—such as hardware counters, memory pressure metrics, and network-level indicators—could further strengthen SQLMind's ability to respond to real-time system dynamics. Overall, this study demonstrates that large-model—driven semantic reasoning, coupled with reinforcement-learning-based adaptive optimization, provides a powerful foundation for next-generation intelligent query optimizers in cloud database systems. SQLMind represents an important step toward self-evolving, context-aware, and cloud-native query optimization frameworks.

References

- Selinger P G, Astrahan M, Chamberlin D, et al. Access path selection in a relational database management system[C]//Proceedings of the 1979 ACM SIGMOD international conference on Management of data. 1979: 23-34.
- Graefe G, McKenna W. The Volcano optimizer generator[R]. 1991.
- Poosala V, Ioannidis Y E. Selectivity estimation without the attribute value independence assumption[C]//VLDB. 1997, 97: 486-495.
- Armenatzoglou N, Basu S, Bhanoori N, et al. Amazon Redshift re-invented[C]//Proceedings of the 2022 International Conference on Management of Data. 2022: 2205-2217.
- Corbett J C, Dean J, Epstein M, et al. Spanner: Google's globally distributed database[J]. ACM Transactions on Computer Systems (TOCS), 2013, 31(3): 1-22.
- Kraska T, Beutel A, Chi E H, et al. The case for learned index structures[C]//Proceedings of the 2018 international conference on management of data. 2018: 489-504.
- Hilprecht B, Schmidt A, Kulessa M, et al. Deepdb: Learn from data, not from queries![J]. arXiv preprint arXiv:1909.00607, 2019.
- Krishnan S, Yang Z, Goldberg K, et al. Learning to optimize join queries with deep reinforcement learning[J]. arXiv preprint arXiv:1808.03196, 2018.
- Marcus R, Negi P, Mao H, et al. Bao: Learning to steer query optimizers[J]. arXiv preprint arXiv:2004.03814, 2020.
- Li G, Zhou X, Li S, et al. Qtune: A query-aware database tuning system with deep reinforcement learning[J]. Proceedings of the VLDB Endowment, 2019, 12(12): 2118-2130.

- Ning Z, Zeng H, Tian Z. Research on data-driven energy efficiency optimisation algorithm for air compressors[C]//Third International Conference on Advanced Materials and Equipment Manufacturing (AMEM 2024). SPIE, 2025, 13691: 1068-1075.
- Yu T, Yasunaga M, Yang K, et al. RAT-SQL: relation-aware schema encoding for text-to-SQL parsers[C]//Proceeding of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP 2020), Online. 2020.
- Pourreza M, Rafiei D. Din-sql: Decomposed in-context learning of text-to-sql with self-correction[J]. Advances in neural information processing systems, 2023, 36: 36339-36348.
- Wei H, Wu Y, Li M. RAGN-IIoT: A Retrieval-Augmented NL2SQL Framework with Dynamic Sensor-Selection Guardrails for Industrial IoT Time-Series Data Warehouses[J]. Journal of Computer, Signal, and System Research, 2025, 2(7): 78-88.
- Lin, Ziyu, and Biliang Wang. "Adaptive load balancing algorithms for cloud computing distributed systems." IET Conference Proceedings CP952. Vol. 2025. No. 39. Stevenage, UK: The Institution of Engineering and Technology, 2025.
- Sun Q, Zhao X, Lin X. Design of a Hardware-Software Co-designed Real-Time Machine Learning System for Big Data Streams[C]//Proceedings of the 2nd International Symposium on Integrated Circuit Design and Integrated Systems. 2025: 265-271.
- Liang Z, Wei W, Zhang K, et al. Research on multi-hop inference optimization of llm based on mquake framework[J]. arXiv preprint arXiv:2509.04770, 2025.
- Yao Y, Zhang W, Li M. Cloud-Edge Federated Incremental Learning Framework for PMSM Efficiency Optimization with Lightweight CNN-LSTM Models and OTA Differential Deployment[J]. Journal of Computer Science and Frontier Technologies, 2026, 3(1): 98-111.
- Li J, Zeng P, Luo P. CANAO: A Cloud-Aware Native Agentic AI Framework for Adaptive Task Orchestration in Cloud-Native Environments[J]. Frontiers in Artificial Intelligence Research, 2026, 3(1): 187-198.