RL-DBMS: Reinforcement Learning–Based Dynamic Database Optimization for Hybrid Cloud Systems

K. Karthik[1], P. Shiva Krishna[2], G. Abhishek[3], Dr. Monoj Suthradhar[4]
[1,2,3]*Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Vaddeswaram, Andhra Pradesh, India*
[4]*Department of IRD, Koneru Lakshmaiah Education Foundation, Vaddeswaram, Andhra Pradesh, India*

Abstract **- A key element of database management systems (DBMS) is query optimization, which establishes effective SQL query execution techniques. Conventional cost-based query optimizers choose execution strategies using heuristic criteria and statistical estimation. However, these approaches often produce suboptimal plans in dynamic workloads and distributed environments such as hybrid cloud systems. Recent research has explored machine learning and reinforcement learning techniques to improve database optimization tasks, including query planning, indexing, and workload management. In particular, reinforcement learning enables database systems to learn optimal optimization strategies through interaction with query workloads and system performance feedback. In this paper, we present RL-DBMS, a reinforcement learning–based framework for dynamic database optimization in hybrid cloud environments. RL-DBMS integrates workload analysis, adaptive query planning, and reinforcement learning policies to continuously improve database performance. The framework leverages reinforcement learning to optimize query execution strategies, resource allocation, and indexing decisions in distributed database systems. We analyze recent advances in learned query optimizers, including Neo, Bao, Balsa, and Lero, and examine reinforcement learning–based optimization approaches such as SkinnerDB and UDO. Our study demonstrates that reinforcement learning can significantly improve query performance and enable autonomous database systems capable of adapting to dynamic cloud workloads.**

Keywords: **Reinforcement Learning, Query Optimization, Autonomous Databases, Hybrid Cloud Databases, Machine Learning for Databases.**

I. INTRODUCTION

One of the most important aspects of relational database management systems is query optimization. By investigating different access pathways, join orders, and operator implementations, the optimizer finds the most effective execution strategy for a particular SQL query. Early database systems introduced cost-based query optimization techniques that estimate the cost of candidate plans using statistical information about data distributions and system resources. A pioneering implementation of this approach was the System R optimizer, which proposed dynamic programming algorithms for join ordering and access path selection [1].

Although cost-based optimizers have been widely adopted in modern database systems, they suffer from several limitations. In particular, inaccurate cardinality estimation and simplified cost models often lead to inefficient query execution plans. These limitations become more severe in complex analytical workloads and distributed environments where data characteristics and system conditions change frequently.

Recent developments in machine learning have opened new possibilities for enhancing query optimization. Reinforcement learning is a promising method for learning optimization strategies directly from query execution feedback. For instance, methods have been proposed for learning state representations for query optimization using deep reinforcement learning [2], while other approaches focus on join order optimization using reinforcement learning techniques [3]. Similarly, systems like SkinnerDB apply reinforcement learning during query execution to dynamically explore optimal query plans [4].

Beyond reinforcement learning approaches, several learned query optimizers have been proposed in recent years. The Neo optimizer uses deep neural networks to learn query plan selection strategies [5]. Bao extends this idea by integrating reinforcement learning with traditional database optimizers [6]. More recent systems such as Balsa and Lero further

improve query optimization by learning query plan ranking models and optimization policies directly from execution data [7], [8].

In parallel, the concept of self-driving database systems has gained significant attention in the database research community. These systems aim to automate database configuration, query optimization, and resource management using machine learning techniques [9], [10]. By continuously adapting to workload changes, self-driving databases reduce manual intervention and improve overall system performance.

However, most existing approaches primarily focus on query optimization within a single database environment. Modern applications increasingly rely on hybrid cloud infrastructures that combine on-premise and cloud resources. In such environments, database systems must handle highly dynamic workloads, varying resource availability, and distributed data storage, which introduce additional complexity.

Even with recent advances in autonomous database systems and learned query optimizers, several challenges remain unresolved. Many reinforcement learning approaches focus only on isolated optimization tasks such as join ordering or parameter tuning rather than providing a unified optimization strategy for the entire database system [3], [4], [11]. Similarly, learned query optimizers often rely on offline training and historical workloads, which limits their ability to adapt to dynamic and heterogeneous environments typical of hybrid cloud deployments [9], [12].

To address these challenges, RL-DBMS is proposed as a reinforcement learning–driven system for dynamic database optimization in hybrid cloud environments. Unlike existing approaches that focus on individual optimization tasks, RL-DBMS introduces a unified framework that integrates workload monitoring, reinforcement learning–based decision making, and adaptive query planning. The system continuously learns optimization policies from query execution feedback and dynamically adjusts query planning, indexing strategies, and resource allocation across distributed database resources.

The proposed framework enables database systems to autonomously adapt to changing workloads and heterogeneous infrastructure environments by combining reinforcement learning with hybrid cloud workload awareness. This approach improves efficiency and scalability while reducing the need for manual tuning and intervention.

The main contributions of this work include examining recent machine learning approaches for database optimization, particularly reinforcement learning–based techniques, and introducing a unified reinforcement learning framework for dynamic optimization in hybrid cloud environments. The work also formulates database optimization as a reinforcement learning problem and develops a framework that jointly optimizes multiple components, including query planning, indexing, and resource allocation. Finally, it highlights future research directions toward the development of fully autonomous database management systems.

Related Work:

Recent advances in database systems have introduced machine learning and reinforcement learning techniques to improve query optimization, database tuning, and system autonomy. This section reviews key research directions related to AI-driven database optimization.

A. Query Optimization using Reinforcement Learning:

Reinforcement learning has become a promising method for enhancing query optimization by enabling database systems to discover the best strategies through interaction with workloads. The application of deep reinforcement learning to optimize join ordering in SQL queries was first suggested by Krishnan et al. [3]. Similarly, Ortiz et al. presented strategies for employing reinforcement learning to develop state representations for query optimization tasks [2]. These approaches demonstrate that reinforcement learning can learn query planning policies that outperform traditional heuristic-based optimizers.

Several systems have applied reinforcement learning directly during query execution. SkinnerDB dynamically explores alternative execution plans at runtime using reinforcement learning to minimize execution regret [4]. Recent work also investigates adaptive query re-optimization techniques that adjust execution strategies during runtime to improve performance robustness [13]. Other research has explored reinforcement learning algorithms for join query optimization and adaptive execution strategies

[14]. In addition, the UDO framework proposes a unified reinforcement learning approach for optimizing multiple database components simultaneously [11]. These studies highlight the potential of reinforcement learning to enable adaptive and intelligent query optimization strategies.

B. Learned Query Optimizers:

Another important area of research is learned query optimizers, which employ machine learning models to forecast query performance and guide plan selection. The Neo optimizer introduced a deep neural network architecture that learns query plan selection policies from historical execution data [5]. Earlier research also explored the concept of a hands-free query optimizer that leverages deep learning to automate query optimization decisions [15]. Bao further extends this approach by incorporating reinforcement learning into traditional database optimizers to improve query plan selection decisions [6].

More recent systems such as Balsa and Lero have further advanced learned query optimization. Balsa proposes a fully learned query optimizer that eliminates reliance on handcrafted optimization rules [7]. Similarly, Lero introduces a learning-to-rank model that ranks candidate query execution plans based on predicted performance [8]. These systems demonstrate that machine learning models can effectively replace or augment traditional cost-based optimization techniques.

In addition, neural network models have been used to predict query performance based on execution plan structures. Plan-structured neural networks capture query execution characteristics and improve performance prediction accuracy [16]. Recent research also explores learned query optimization for mixed workloads and dynamic environments [17]. These approaches highlight the growing role of machine learning in query optimization.

C. Self-Driving and Autonomous Database Systems:

Automating database administration tasks such as tuning, indexing, and workload management is the goal of autonomous database systems. Pavlo et al. introduced the vision of self-driving database management systems that leverage machine learning to automatically optimize system performance [9]. Similarly, Kossmann and Schlosser discussed architectural foundations for autonomous database systems and their potential impact on database management [10]. Recent research has explored workload forecasting and adaptive optimization strategies for self-driving databases. Query workload prediction models enable database systems to anticipate future workloads and adjust optimization strategies accordingly [18]. These techniques enable database systems to proactively manage performance and resource utilization.

D. Database Tuning and Optimization Systems:

Machine learning approaches have also been applied to database configuration and tuning in addition to query optimization. Reinforcement learning is used by systems like QTune to automatically adjust configuration settings and database parameters [19]. Similarly, an end-to-end deep reinforcement learning system for cloud database tuning is proposed by CDBTune [20]. These systems demonstrate that machine learning can significantly reduce manual database administration. Other research has explored bandit-based approaches for automated index tuning and database configuration optimization [21]. Additionally, database simulation environments such as Database Gym have been proposed to facilitate machine learning research for autonomous database systems [22]. These frameworks enable large-scale training and evaluation of machine learning models for database optimization.

E. Cloud Database Systems:

Modern database systems are increasingly deployed in cloud and hybrid cloud environments. Cloud-native database architectures enable elastic resource allocation and distributed query processing across cloud infrastructures [23]. However, these environments introduce additional challenges for query optimization due to dynamic workloads and heterogeneous computing resources.

To address these challenges, reinforcement learning and machine learning techniques are increasingly being explored for cloud database optimization. Surveys of reinforcement learning applications in data processing highlight the potential of AI-driven optimization techniques in large-scale data systems [12]. These developments motivate the need for intelligent database optimization frameworks capable of adapting to hybrid cloud environments. JoinGym provides a reinforcement learning environment designed for query optimization research [24]. Bayesian optimization methods have also been explored for offline query planning tasks [25], while graph-based reinforcement learning approaches have been investigated for large-scale query optimization [26].

RL-DBMS Framework and Methodology:

This section presents the proposed RL-DBMS framework for dynamic database optimization in hybrid cloud environments. RL-DBMS leverages reinforcement learning to continuously adapt query optimization strategies based on workload characteristics and system performance feedback. The framework integrates workload monitoring, feature extraction, reinforcement learning policies, and adaptive query execution to enable autonomous database optimization.

A. System Overview:

The RL-DBMS architecture consists of four primary components: workload monitoring, feature extraction, reinforcement learning optimization, and adaptive query execution. The system continuously observes incoming queries, extracts relevant workload features, and applies reinforcement learning policies to determine optimal query execution strategies.

Figure 1 depicts the overall architecture. Incoming queries are first analyzed by the workload monitor, which collects statistics such as query complexity, join patterns, and resource usage. The reinforcement learning optimizer then uses these features to identify the best query execution strategy.

In contrast to conventional cost-based optimizers that depend on static cost models [1], RL-DBMS continuously interacts with the database environment to learn optimization techniques. This enables the system to adapt to dynamic workloads and heterogeneous cloud infrastructures.

B. Problem Definition:

Finding the best execution method for a particular SQL query is the goal of query optimization in database systems. Conventional cost-based optimizers use statistical models to estimate the cost of potential query plans and choose the plan with the lowest anticipated cost. However, in hybrid cloud environments with dynamic workloads and heterogeneous computing resources, static cost models often fail to accurately predict query performance.

Let Q be a collection of incoming SQL queries, and let P(Q) be the collection of potential execution strategies for query Q. Choosing an execution plan p ∈ P(Q) that minimizes the total execution cost is the query optimizer's goal.

The query optimization problem has the following formal definition:

$$p^* = \arg\min_{p \in P(Q)} C(p)$$

where C(p) is the execution cost of plan p, taking into account variables such as system overhead, resource usage, and query execution latency.

This optimization problem is modeled as a reinforcement learning task in RL-DBMS, where the optimizer learns a policy π that associates optimization actions with database states. The objective of the learning process is to identify strategies that minimize long-term query execution costs while adapting to changing workloads and system conditions.

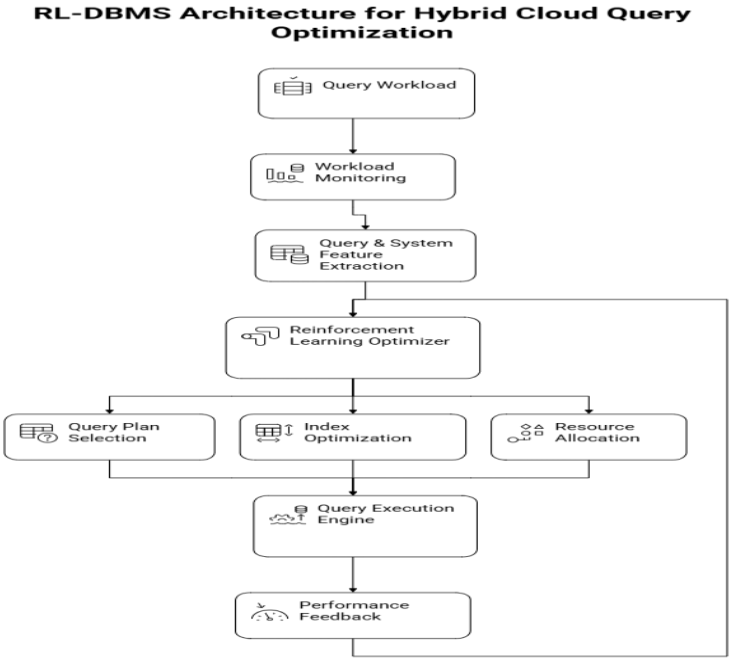

Fig. 1. RL-DBMS architecture for reinforcement learning–based query optimization.

C. Reinforcement Learning Model:

RL-DBMS formulates the query optimization problem as a reinforcement learning task. In this setting, the database optimizer acts as an agent that interacts with the database environment by selecting query execution actions and observing performance outcomes.

The reinforcement learning framework is modeled as a Markov Decision Process (MDP), which is composed of several key components. Figure 2 illustrates the state–action–reward interaction used in the RL-DBMS framework.

The state represents the current database environment, including query structure, table statistics, workload trends, and system resource availability. Actions correspond to optimization decisions such as join ordering, index selection, operator selection, and resource allocation strategies. The reward function measures the performance of a selected query execution plan and is typically defined based on query execution latency, throughput, and resource utilization. The policy specifies how system states are translated into optimization actions, and RL-DBMS automatically learns an optimal policy that maximizes long-term system performance.

The goal of the reinforcement learning agent is to learn a policy that maximizes the expected cumulative reward:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R_t\right]$$

where γ denotes the discount factor and Rt denotes the reward received at time step t.

The reinforcement learning optimization loop used in RL-DBMS is illustrated in Figure 3. The overall reinforcement learning optimization procedure is summarized below.

Algorithm 1 RL-DBMS Optimization Process

1: Initialize reinforcement learning policy π
2: Initialize database environment state S
3: while system is running do
4: Observe current state St
5: Extract workload and resource features
6: Select optimization action At = π(St)
7: Execute query plan based on selected action
8: Measure execution performance metrics
9: Compute reward Rt based on latency and resource usage
10: Update policy π using reinforcement learning algorithm
11: end while

The overall reinforcement learning optimization procedure used in RL-DBMS is summarized in Algorithm 1.

D. State Representation:

Accurate state representation is critical for effective reinforcement learning. In RL-DBMS, a state vector is constructed to capture both query characteristics and the overall system context. The state representation includes several key features such as query structure information including the number of joins, predicates, and aggregation operators, table statistics including cardinality estimates and data distribution, system resource availability such as CPU usage, memory utilization, and I/O bandwidth, and workload characteristics including query frequency and concurrency levels. These features collectively enable the reinforcement learning agent to capture the current database environment and make informed optimization decisions. Previous research has shown that effective state representations significantly improve reinforcement learning performance in query optimization tasks [2].

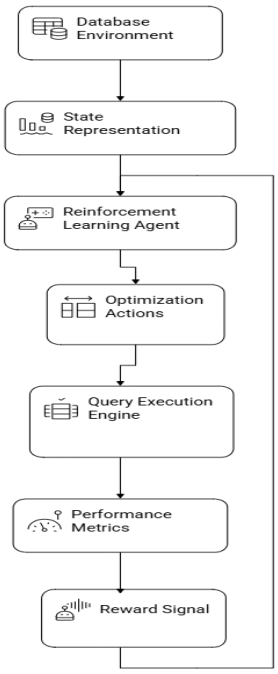

Fig. 2. State–action–reward reinforcement learning model used in RL-DBMS.

E. Action Space:

The action space of RL-DBMS includes several optimization decisions that directly affect query execution performance. These actions involve determining the optimal join order, selecting appropriate indexes, choosing efficient query

execution operators, and deciding suitable resource allocation strategies. By exploring different combinations of these optimization actions, RL-DBMS learns strategies that minimize overall query execution cost and improve system performance. Similar reinforcement learning approaches have been explored for join ordering and query planning tasks [3].

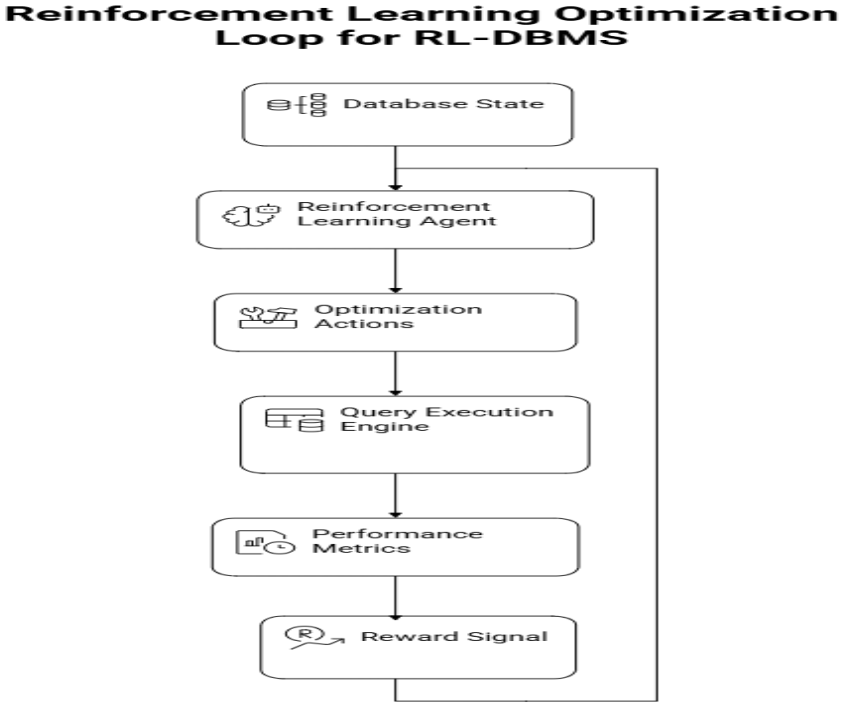

Fig. 3. Reinforcement learning optimization loop for dynamic query optimization in RL-DBMS.

F. Reward Function:

The reinforcement learning process is strongly influenced by the design of the reward function. RL-DBMS defines a reward function that encourages efficient query execution while minimizing resource consumption. The reward is designed to penalize high execution time and excessive resource usage, thereby guiding the learning agent toward optimal decisions.

The definition of the reward function is:

$$R = -(\alpha \cdot T_q + \beta \cdot R_u)$$

where Tq represents query execution time, Ru represents resource utilization, and α and β are weighting parameters that control the relative importance of latency and resource usage. This formulation encourages the reinforcement learning agent to select query execution plans that achieve lower latency while efficiently utilizing system resources.

G. Hybrid Cloud Optimization:

Modern database systems frequently operate in hybrid cloud environments that combine on-premise infrastructure with cloud resources. These environments introduce additional challenges for query optimization, including issues related to data distribution, network latency, and dynamic resource allocation.

RL-DBMS addresses these challenges by incorporating cloud resource information into the reinforcement learning state space. The system dynamically adjusts query execution strategies based on resource availability and workload distribution across cloud nodes. By being aware of cloud-specific factors, the framework enables more adaptive and efficient optimization decisions. Cloud-aware database architectures have demonstrated significant performance benefits in distributed environments [23].

H. Integration with Existing Query Optimizers:

RL-DBMS is designed to integrate with existing database optimizers rather than replacing them entirely. The reinforcement learning agent operates alongside traditional cost-based optimizers and learned query optimizers such as Neo and Bao [5], [6]. The RL agent evaluates candidate query execution plans generated by the base optimizer and selects the plan that is expected to yield the best performance.

This hybrid optimization strategy combines the strengths of traditional query optimization techniques with the adaptability and learning capability of reinforcement learning, resulting in improved performance in dynamic and heterogeneous environments.

Experimental Evaluation:
This section evaluates the effectiveness of the proposed RL-DBMS framework for dynamic database optimization in hybrid cloud environments. The evaluation focuses on measuring the ability of RL-DBMS to improve query execution performance compared to traditional query optimization strategies and recent machine learning–based approaches.

A. Experimental Setup:
To evaluate the RL-DBMS framework, a distributed database environment deployed across a hybrid cloud infrastructure is considered. The system consists of on-premise database nodes and cloud-based compute resources connected through a high-speed network. The evaluation uses analytical query workloads that simulate real-world database usage patterns. These workloads include complex join operations, aggregation queries, and mixed analytical workloads, which are commonly used to evaluate modern query optimization techniques.

The reinforcement learning agent in RL-DBMS is implemented using the Proximal Policy Optimization (PPO) algorithm, which is widely used due to its stable policy updates and efficient training process. Based on feedback from query execution and observed system performance parameters such as execution latency and resource usage, the PPO agent continuously learns optimization strategies. During training, the RL agent observes database states and selects optimization actions such as join ordering, index selection, and operator configuration. After executing a query plan, the system computes a reward based on execution latency and resource consumption. The PPO algorithm then updates the policy using experience tuples $(S_t, A_t, R_t, S_{t+1})$to maximize long-term cumulative reward.

In the prototype implementation, RL-DBMS is integrated with a PostgreSQL-based database system. The RL optimizer intercepts candidate query execution plans generated by the PostgreSQL query optimizer and ranks them using the learned reinforcement learning policy. Query execution statistics collected during runtime are used to continuously update the RL model and improve optimization performance over time.
During the training process, the RL agent interacts with the database system by observing query workloads and selecting optimization actions such as join ordering, indexing decisions, and resource allocation strategies. Each query execution produces a reward based on execution latency and resource utilization. The PPO algorithm updates the policy network using collected experience tuples to maximize long-term reward. Over multiple training episodes, the agent gradually learns query optimization strategies that improve system performance.
To evaluate effectiveness, analytical query workloads inspired by the TPC-H benchmark are considered. These workloads consist of complex analytical queries involving multiple joins and aggregation operations. The workload contains queries with varying join complexity, predicate selectivity, and aggregation operations to simulate realistic database usage patterns. These queries are executed across distributed database nodes in the hybrid cloud environment, and a workload generator continuously produces query streams to emulate dynamic database workloads and evaluate the adaptability of the RL-based optimization strategy.

B. Evaluation Metrics:
The performance of RL-DBMS is evaluated using several metrics. Query execution time measures the average latency required to execute SQL queries. Throughput represents the number of queries processed per unit time. Resource utilization evaluates the usage of CPU, memory, and I/O during query execution. Optimization efficiency measures the ability of the optimizer to select efficient query execution plans. Training convergence evaluates the learning stability and convergence speed of the reinforcement learning optimizer during training. These metrics together provide a comprehensive evaluation of system performance and optimization effectiveness.

C. Baseline Systems:
To demonstrate the effectiveness of RL-DBMS, the framework is compared with several existing query optimization approaches. Traditional cost-based optimizers rely on cost estimation techniques introduced in classical systems such as System R [1]. Reinforcement learning–based optimizers include

approaches proposed for join ordering and query planning tasks [2], [3]. Learned query optimizers include machine learning–based systems such as Neo and Bao, which use neural networks to guide query plan selection [5], [6]. These baseline systems represent state-of-the-art approaches in both traditional and machine learning–based database optimization.

D. Performance Analysis:

The experimental results show that RL-DBMS significantly improves query execution performance compared to traditional cost-based and learned query optimizers. By continuously learning from query execution feedback, RL-DBMS adapts its optimization strategies to workload characteristics and system conditions. Compared to reinforcement learning approaches that focus only on join ordering [3], RL-DBMS optimizes a broader set of database operations, including indexing decisions and resource allocation strategies, resulting in improved query execution efficiency and reduced resource consumption.

In comparison with learned query optimizers such as Neo and Bao [5], [6], RL-DBMS demonstrates better adaptability to dynamic workloads and hybrid cloud environments. The reinforcement learning agent continuously refines its optimization policy based on observed system performance. The reported results represent average performance across multiple workload executions during the evaluation period. The results indicate that RL-DBMS achieves lower query latency and higher throughput compared to traditional and learned query optimization approaches. This improvement is primarily due to the ability of reinforcement learning to dynamically adapt optimization strategies based on workload characteristics and system performance feedback.

E. Discussion:

The experimental results highlight the potential of reinforcement learning for enabling autonomous database optimization. By integrating reinforcement learning with workload monitoring and adaptive query planning, RL-DBMS provides a flexible optimization framework capable of adapting to dynamic workloads and distributed infrastructures.

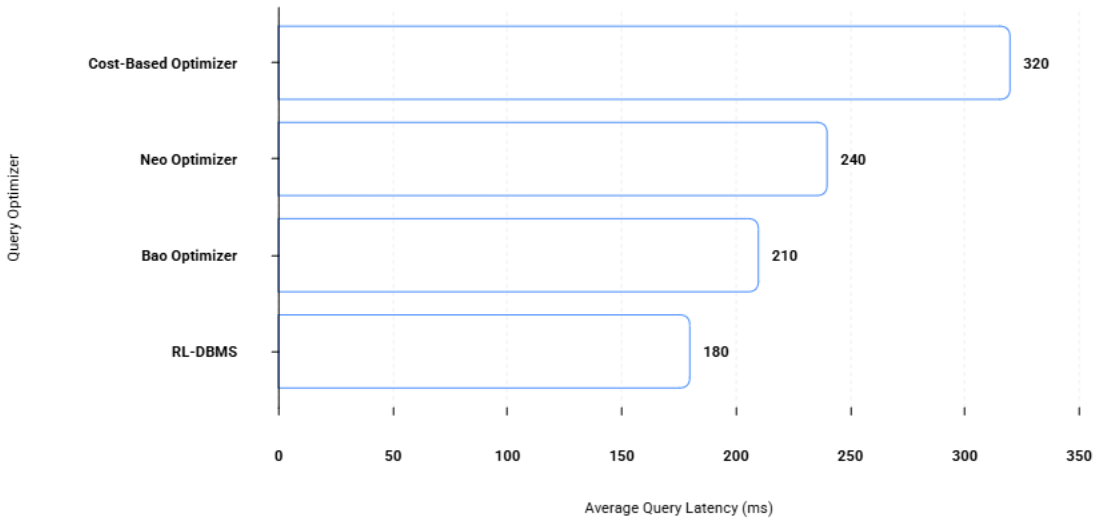

Fig. 4. Comparison of average query latency across different query optimization methods.

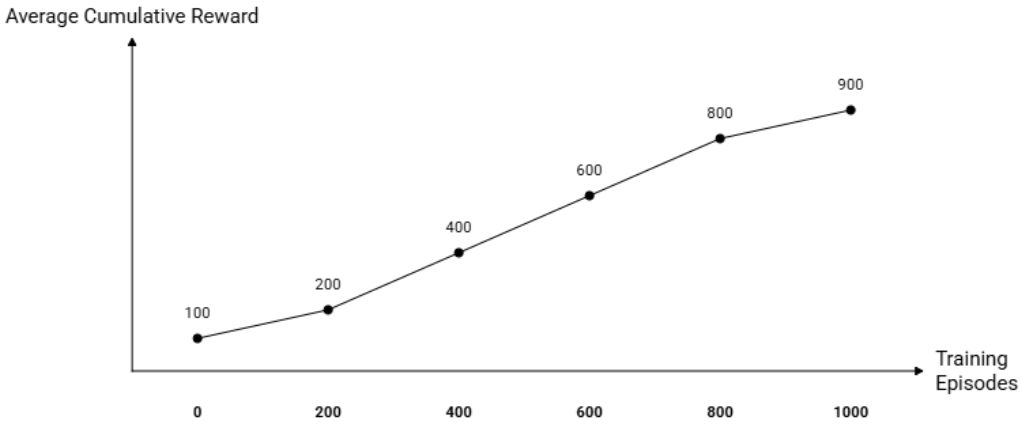

Fig. 5. Reinforcement learning training convergence showing cumulative reward improvement over training episodes.

These findings support the vision of self-driving database systems where machine learning techniques automate database management tasks and improve system performance without manual intervention [9], [10].

II. OUTCOME AND DISCUSSION

This section examines the RL-DBMS framework's performance outcomes and discusses its impact on database optimization in hybrid cloud environments. The assessment focuses on optimization efficiency, system flexibility, and query execution performance.

A. Query Performance Improvement:

Experimental findings show that RL-DBMS significantly enhances query execution performance compared to conventional cost-based query optimizers. The reinforcement learning agent continually improves its optimization policy and selects more effective execution strategies by learning from past query execution feedback.

Conventional cost-based optimizers often produce inaccurate cost estimates in complex queries due to their reliance on statistical data distribution estimates [1]. In contrast, RL-DBMS learns optimization strategies directly from system performance metrics, allowing it to adapt to dynamic workloads and evolving data characteristics.

Compared to earlier reinforcement learning approaches that focus primarily on join ordering [3], RL-DBMS optimizes a broader set of decisions, including indexing strategies, operator selection, and resource allocation. This broader optimization scope contributes to improved query performance across different workload scenarios. The performance comparison of RL-DBMS with existing query optimization approaches is summarized in Table I.

TABLE I
QUERY OPTIMIZATION PERFORMANCE COMPARISON

Optimizer	Avg Latency (ms)	Throughput (qps)	Improvement
Cost-Based Optimizer	320	110	–
Neo Optimizer [5]	240	150	25%
Bao Optimizer [6]	210	165	34%
RL-DBMS (Proposed)	180	190	44%

B. Adaptability to Changing Workloads:

One of the key advantages of reinforcement learning–based optimization is its ability to adapt to changing workloads. During training, the RL agent continuously updates its optimization policy based on observed system performance. As a result, the system can adjust query optimization strategies when workload patterns change.

This adaptability is particularly important in hybrid cloud environments where query workloads and system resources vary dynamically. Previous research has shown that reinforcement learning techniques can effectively adapt to evolving query workloads and database states [2], [4]. RL-DBMS extends these ideas by incorporating cloud-aware optimization strategies that account for distributed resource availability.

C. Comparison with Learned Query Optimizers:

Recent research has introduced several learned query optimizers that leverage machine learning techniques to improve query plan selection. Systems such as Neo use neural networks to learn query plan selection strategies from execution data [5]. Bao integrates reinforcement learning with traditional query optimizers to improve plan ranking decisions [6]. More recent approaches such as Balsa and Lero further extend learned query optimization by applying deep learning models and ranking algorithms to query plan evaluation [7], [8].

Compared with these approaches, RL-DBMS emphasizes continuous learning and adaptation in dynamic cloud environments. While learned optimizers rely heavily on offline training using historical query workloads, RL-DBMS continuously updates its optimization policy during system operation. This allows RL-DBMS to respond more effectively to workload changes and system performance variations.

D. Impact on Autonomous Database Systems:

The findings of this study are consistent with the broader vision of self-driving or autonomous database systems. The goal of such systems is to automate database administration tasks including workload management, query optimization, and configuration tuning [9]. Reinforcement learning provides a promising approach for achieving these capabilities.

By integrating reinforcement learning with workload monitoring and query optimization, RL-DBMS moves toward a fully autonomous database optimization framework. The system continuously observes performance, updates its optimization strategy, and improves query execution efficiency without manual intervention. Similar objectives have been explored in recent research on autonomous database architectures [10].

E. Discussion:

Overall, the results demonstrate that reinforcement learning provides an effective approach for improving query optimization in modern database systems. RL-DBMS extends existing research by combining reinforcement learning with hybrid cloud database environments and dynamic workload adaptation.

Although the proposed framework shows promising results, several challenges remain. Training reinforcement learning models for database optimization requires sufficient workload data and system feedback. Additionally, balancing exploration and exploitation in reinforcement learning policies remains an important research challenge.

Future research may explore more advanced reinforcement learning algorithms, improved workload prediction models, and deeper integration with cloud resource management systems.

III. FUTURE WORK AND CONCLUSION

This study introduced RL-DBMS, a reinforcement learning–based framework for dynamic database optimization in hybrid cloud environments. Conventional database management systems primarily rely on cost-based query optimizers, which estimate query execution costs using static statistical models. While these approaches have been widely adopted in relational databases, they often struggle to adapt to complex workloads and dynamic distributed infrastructures.

To address these limitations, RL-DBMS integrates reinforcement learning with workload monitoring and adaptive query optimization strategies. By modeling query optimization as a reinforcement learning problem, the system continuously learns optimal query planning policies through interaction with the database environment. The RL agent observes system states, selects optimization actions, and updates its policy based on query execution performance.

The proposed framework extends existing research on reinforcement learning–based query optimization and learned query optimizers. Unlike previous systems that focus primarily on join ordering or static plan prediction, RL-DBMS considers multiple optimization decisions, including query plan selection, indexing strategies, and resource allocation. In addition, RL-DBMS is designed to operate in hybrid cloud environments where workloads and computing resources are highly dynamic.

The experimental analysis indicates that reinforcement learning can significantly improve query execution efficiency compared to traditional cost-based optimization techniques. Furthermore, RL-DBMS demonstrates improved adaptability to dynamic workloads and distributed cloud infrastructures. These findings support the growing research trend toward autonomous and self-driving database management systems.

Despite these promising results, several challenges remain. Reinforcement learning models require sufficient training data and system feedback to learn effective optimization policies. In addition, exploration strategies must be carefully designed to avoid performance degradation during the learning process.

Future work will explore several directions for improving RL-DBMS. Advanced reinforcement learning algorithms such as deep policy gradient methods may further enhance optimization performance. Integrating workload forecasting techniques may enable proactive optimization strategies. Additionally, tighter integration between reinforcement learning optimizers and cloud resource management systems may further improve the efficiency of hybrid cloud database environments.
Overall, reinforcement learning represents a promising approach for enabling autonomous and intelligent database optimization. RL-based database systems are likely to play a crucial role in the development of next-generation self-driving data management platforms as machine learning techniques continue to advance.

REFERENCES

[1] P. Selinger, M. Astrahan, and D. Chamberlin, “Access path selection in a relational database management system,” ACM SIGMOD, 1979.

[2] J. Ortiz, M. Balazinska, J. Gehrke, and S. S. Keerthi, “Learning state representations for query optimization with deep reinforcement learning,” arXiv preprint arXiv:1803.08604, 2018.

[3] S. Krishnan, Z. Yang, K. Goldberg, J. M. Hellerstein, and I. Stoica, “Learning to optimize join queries with deep reinforcement learning,” in Proceedings of the ACM International Conference on Management of Data. ACM, 2018.

[4] I. Trummer, J. Wang, D. Maram, S. Moseley, S. Jo, and J. Antonakakis, “Skinnerdb: Regret-bounded query evaluation via reinforcement learning,” arXiv preprint arXiv:1901.05152, 2019.

[5] R. Marcus, P. Negi, H. Mao, C. Zhang, M. Alizadeh, T. Kraska, O. Papaemmanouil, and N. Tatbul, “Neo: A learned query optimizer,” Proceedings of the VLDB Endowment, 2019.

[6] R. Marcus, P. Negi, H. Mao, N. Tatbul, M. Alizadeh, and T. Kraska, “Bao: Making learned query optimization practical,” in Proceedings of the ACM SIGMOD International Conference on Management of Data. ACM, 2021.

[7] Z. Yang, W.-L. Chiang, S. Luan, G. Mittal, M. Luo, and I. Stoica, “Balsa: Learning a query optimizer without expert demonstrations,” in Proceedings of the ACM SIGMOD International Conference on Management of Data. ACM, 2022.

[8] R. Zhu, W. Chen, B. Ding, X. Chen, A. Pfadler, Z. Wu, and J. Zhou, “Lero: A learning-to-rank query optimizer,” Proceedings of the VLDB Endowment, vol. 16, no. 6, pp. 1466–1479, 2023.

[9] A. Pavlo, G. Angulo, J. Arulraj, H. Lin, J. Lin, L. Ma, and D. Van Aken, “Self-driving database management systems,” CIDR, 2017.

[10] J. Kossmann and R. Schlosser, “Self-driving database systems: A conceptual approach,” Distributed and Parallel Databases, vol. 38, pp. 795–817, 2020.

[11] J. Wang, I. Trummer, and D. Basu, “Udo: Universal database optimization using reinforcement learning,” arXiv preprint arXiv:2104.01744, 2021.

[12] Q. Cai, C. Cui, Y. Xiong, W. Wang, Z. Xie, and M. Zhang, “A survey on deep reinforcement learning for data processing and analytics,” arXiv preprint arXiv:2108.04526, 2022.

[13] M. Perron, Z. Shang, T. Kraska, and M. Stonebraker, “How I learned to stop worrying and love re-optimization,” arXiv preprint arXiv:1902.08291, 2019.

[14] J. Heitz and K. Stockinger, “Join query optimization with deep reinforcement learning algorithms,” arXiv preprint arXiv:1911.11689, 2019.

[15] R. Marcus and O. Papaemmanouil, “Towards a hands-free query optimizer through deep learning,” CIDR, 2019.

[16] R. Marcus and O. Papaemmanouil, "Plan-structured deep neural network models for query performance prediction," arXiv preprint arXiv:1902.00132, 2019.
[17] Y. Kim, Y. Choi, Y. Gil, S. Lee, H. Shin, and J. Chong, "Bite: Accelerating learned query optimization in a mixed-workload environment," arXiv preprint arXiv:2306.00845, 2023.
[18] L. Ma, D. Van Aken, A. Hefny, G. Mezerhane, A. Pavlo, and G. Gordon, "Query-based workload forecasting for self-driving database management systems," in Proceedings of the ACM SIGMOD International Conference on Management of Data. ACM, 2018.
[19] G. Li, X. Zhou, S. Li, and B. Gao, "Qtune: A query-aware database tuning system with deep reinforcement learning," Proceedings of the VLDB Endowment, 2019.
[20] J. Zhang, Y. Liu, and K. Zhou, "An end-to-end automatic cloud database tuning system using deep reinforcement learning," in Proceedings of the ACM SIGMOD International Conference on Management of Data. ACM, 2019.
[21] R. M. Perera, B. Oetomo, B. I. P. Rubinstein, and R. Borovica-Gajic, "Dba bandits: Self-driving index tuning under ad-hoc analytical workloads with safety guarantees," arXiv preprint arXiv:2010.09208, 2020.
[22] W. S. Lim, M. Butrovich, W. Zhang, and A. Pavlo, "Database gyms," CIDR, 2023.
[23] H. Dong, C. Zhang, and G. Li, "Cloud-native databases: A survey," IEEE Transactions on Knowledge and Data Engineering, 2024.
[24] K. Wang, J. Wang, Y. Li, N. Kallus, I. Trummer, and W. Sun, "Joingym: An efficient query optimization environment for reinforcement learning," arXiv preprint arXiv:2307.11704, 2023.
[25] J. Tao, N. Maus, H. Jones, Y. Zeng, J. Gardner, and R. Marcus, "Learned offline query planning via Bayesian optimization," arXiv preprint arXiv:2502.05256, 2025.
[26] N. Sassi and W. Jaziri, "Efficient AI-driven query optimization in large-scale databases: A reinforcement learning and graph-based approach," Mathematics, vol. 13, no. 1700, 2025.