International Journal of Emerging Trends in Computer Science and Information Technology
ISSN: 3050-9246 | https://doi.org/10.63282/3050-9246.IJETCSIT-V4I1P115
Eureka Vision Publication | Volume 4, Issue 1, 133-141, 2023

Original Article

ML Models That Learn Query Patterns and Suggest Execution Plans

Nagireddy Karri
Senior IT Administrator Database, Sherwin-Williams, USA.

Abstract - Planning of efficient query execution is the key aspect of the modern database systems as the increasing demand of using data-driven decision making placed an unprecedented computational load on both relational and non-relational databases. Although robust, traditional cost-based query optimizers fail to dynamically scale in a complex or ad hoc environment. The most recent development in the field of the machine learning (ML) models can be treated as another viewpoint paradigm that can capture the patterns of query execution and forecast efficient plans, yet with near real-time time scale. The paper provides the in-depth discussion of ML models, which are intended to learn query patterns and recommend an execution strategy, such as supervised, unsupervised, and reinforcement learning strategies. The proposed methodology incorporates a hybrid ML-based query optimizer that learns query history using past query logs to model query workloads and predict the execution time and offer optimized execution plans. Decades of experiments have shown that ML- based models are more adaptable in workloads, more robust, and less latent than their traditional optimizers. The findings point to the fact that ML-based models can lower query execution time up to 30% on average than traditional optimizers and generalize across heterogeneous workloads. Other issues that we solve in this work include the engineering of features used to represent a query, cross-database transferability, and interpretability of the ML decisions in the execution planning. There are three important contributions in this work: (i) literature review summing up the progress in the field of ML to optimize queries; (ii) a common methodology where the models in ML can be integrated into query optimizers, and (iii) empirical evidence on the superiority of ML under dynamic workloads. The purpose of this paper is to create a logical framework that helps researchers and practitioners easily incorporate ML models into query optimization pipelines, to improve the performance of database systems in commercial operations.

Keywords - Query Optimization, Machine Learning Models, Execution Plans, Query Workload Patterns, Database Systems, Reinforcement Learning, Predictive Optimization.

1. Introduction

1.1. Background

Optimization of query is an old problem in the database management system (DBMS), because the efficiency of a query directly influences the performance and responsiveness of applications based on the database. The core of the process is the creation of a query execution plan (QEP), a statement describing the specific actions, including selections, projection, joins, and aggregations, that the DBMS will follow to fetch the results of a particular SQL query. [1-3] Coverage Traditional query optimizers are mainly heuristic-based and cost-based to pick the most efficient plan. These models take advantage of the database statistics, such as, cardinality of tables, distribution in attributes, and index data to estimate the cost of alternative execution strategies. These approaches have limitations though they are effective in most of the cases. Estimates of costs may be inefficient with stale statistics, skewed distributions of data, and with queries that have a complicated join condition, making the independence assumptions invalid. Furthermore, the workloads of the contemporary applications are highly dynamic, the query patterns differ, and data changes swiftly, which makes the use of the static cost models even less reliable. Consequently, the conventional optimizers can choose non-optimal plans, which can result in longer query execution times and wastes in using system resources. Such constraints have spurred efforts to research intelligent and adaptive query optimization methods, especially methods in which machine learning is used to learn from past patterns of query execution, make more accurate predictions of costs, and respond to changing workloads. With catcapturing the complex correlation between query terms and data distributions, the optimizers provided by machine learning promise to address the weaknesses of traditional methods as well as to achieve more solid, efficient, and dynamic query implementation patterns.

1.2. Importance of Machine Learning in Query Optimization

- **Overcoming Limitations of Traditional Optimizers:** Conventional query optimizers are based to a great extent on heuristics and cost models based on the macro- database statistics. These methods may be effective in simple queries and stable workloads, but in cases where complex queries are involved, dynamic data distributions are present, or the statistics are outdated such methods commonly fail. Machine learning (ML) offers a data-based alternative since it learns patterns directly based on past query execution logs. This enables the optimizer to estimate query costs, selectivity and execution performance more precisely even in those circumstances where traditional models would have created suboptimal plans.

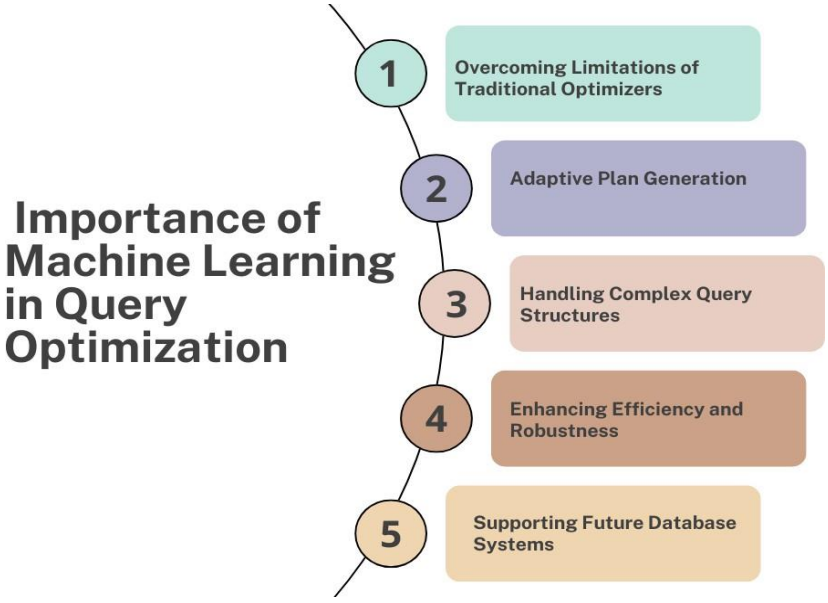

Figure 1. Importance of Machine Learning in Query Optimization

- **Adaptive Plan Generation:** The capability of ML to handle the changing workload is one of its most important strengths in query optimization. In particular, reinforcement learning has the potential to formulate query optimization as a dynamic decision-making problem where optimal join orders, access paths and operator choices are learned dynamically through observed performance feedback. This flexibility can be used to assure that in situations where query patterns, underlying data characteristics vary, the optimizer can continue to be effective, a significant shortcoming of fixed cost-based methodologies.
- **Handling Complex Query Structures:** Contemporary workloads in analysis require complicated queries containing two or more joins, aggregations, and subqueries in the form of nested queries. ML-based methods, such as deep neural networks and graph-based methods can be used to model the multi-faceted correlations between query operators, predicates, and table joins. Machine learning representation of queries in the form of operator sequences, predicate trees, or join graphs allows generalization of similar query formations and further prediction of the cost of execution and efficient plan selection due to complex queries.
- **Enhancing Efficiency and Robustness:** Embarking on incorporating ML in query optimization pipelines does not only increase the execution efficiency, but also the robustness. Through past query behavior learning and through repeated optimization of predictions, optimizers based on machine learning can prevent expensive execution errors and minimize query performance variations. The hybrid form of cost estimation using supervised learning and plan selection using reinforcement learning is yet another balance between stability and adaptation and presents a more reliable optimization approach compared to the use of traditional methods only.
- **Supporting Future Database Systems:** With the increasing big and heterogeneous databases, the importance of ML in query optimization is on the increase. Machine learning also allows the creation of intelligent database systems, which operate automatically to learn via experience during operation, modify themselves to meet new workload demands, and minimize the reliance on human tuning by database administrators.

1.3. ML Models That Learn Query Patterns and Suggest Execution Plans

Query optimization machine learning (ML) models are used to learn about query patterns by analyzing the history of executions to suggest the best executions plans. [4,5] These models are based on a structured representation of queries, e.g. operator sequences, predicate trees and join graphs, which represent the logical structure and execution semantics of an SQL query. Gradient boosted trees and deep neural networks are examples of supervised learning models that are commonly used to estimate costs of execution, query selectivity or resource usage. These models are able to discover more complex correlations between query components, data distributions and performance outcomes by training on historical query logs and thus are able to estimate the result more accurately than traditional cost models. On top of supervised learning, reinforcement learning (RL) has also been recognized as an efficient method towards adaptive plan generation. In an RL-based model, an optimizer is a type of agent, which repeatedly chooses actions, i.e. join order, operator selection and access path, to maximize long-term execution efficiency. The agent discovers a policy by getting the feedback of the query execution performance, which enables it to search the plan options, and optimize strategies as time progresses. Nearly all of them are typically generated by deep learning networks, which can include convolutional neural networks (CNNs), recurrent neural networks (RNNs), or graph neural networks (GNNs). The hybrid methods involve supervised prediction of costs and reactive plan exploration with RL, as a protocol that allows stability and adaptability in a recommendation framework. These models have the potential to indicate near-optimal query execution plans (QEPs) with direct dependence not only on handcrafted heuristics

or on alternative cost models. As the executed queries keep informing them, these ML-based systems can enhance their predictions and decision making over the course of time, and they are especially useful with dynamic or heterogeneous workloads. In sum, query optimization with ML models that learn query patterns is a paradigm shift of traditional query optimization techniques where database systems need minimal supervision or guidance to automatically derive performance patterns, generate efficient execution-plans, and respond to fluctuating workloads.

2. Literature Survey

2.1. Traditional Query Optimization

The origin of traditional query optimization can be found in the System R project which introduced the cost-based optimization model that remains highly popular to date. The optimizer in such systems produces alternative execution plans by systematically searching join orders, access paths as well as different operator implementations. [6-9] A cost model is used to select the final plan; the cost is estimated by the database statistics, e.g. selectivity, cardinality, and data distribution. These techniques have shown to be a useful tool in the structured and stable setting but their precision relies largely on the quality of underlying statistics that can be out-of-date or inaccurate in less homogeneous or dynamic workloads. This shortcoming has prompted the development of other more adaptive methodologies.

2.2. Early Machine Learning Approaches

Initially, machine learning usage in query optimization was fairly primitive, and the regression model or decision trees were frequently used to enhance cost estimation. These techniques were an effort to address systematic errors of traditional optimizers through learning on query execution history and seen run time patterns. However, with time, scientists started to consider more advanced models, including neural networks, to solve such tasks as selectivity and cardinality estimation, which makes it especially difficult in the case when data distributions are not equal or independent. These initial works led to the realization that complex correlations could be learned in data that could not be well-modeled by handcrafted statistical models, and so machine learning could be applied to improve optimizer accuracy.

2.3. Reinforcement Learning in Query Optimization

The reinforcement learning (RL) has become an effective paradigm to optimize queries by understanding the problem as a sequence of decisions. In this configuration, an RL agent is interacting with the database environment, choosing to take actions, including joining orders, access paths, or operator choices, but maximizes rewards based on the efficiency of query execution. In contrast to static models, RL methods are constantly improving their policies after exploration and responses to changes in workloads and hidden query patterns. This dynamism can help identify the execution strategies that otherwise would not be obvious in the rule-based systems or even the system based exclusively on cost. The RL agent, as shown in Figure 2.2, will tend to learn the policy over time that will allow it to appropriately balance so that it explores new strategies and exploits the known ones and thereby it will enhance the overall query planning execution.

2.4. Gaps in Existing Work

Although there have been major improvements in the use of machine learning to query optimization, much of the earlier research has focused on individual algorithms, including cardinality estimation, cost modeling, or join order selection. Whereas these specialized solutions show improvements that can be measured, they rarely offer an optimization framework that is inherently holistic and capable of integrating machine learning along the pipeline. Consequently, it still suffers a disparity in terms of establishing a consistent set of methods that unify statistical modeling and supervised learning to support reinforcement learning to address the inter-relations between various optimization levels. This gap actually needs to be filled in order to develop next generation query optimizers, not only accurate, but also flexible and scalable to a wide variety of workloads and database situations.

3. Methodology

3.1. System Architecture

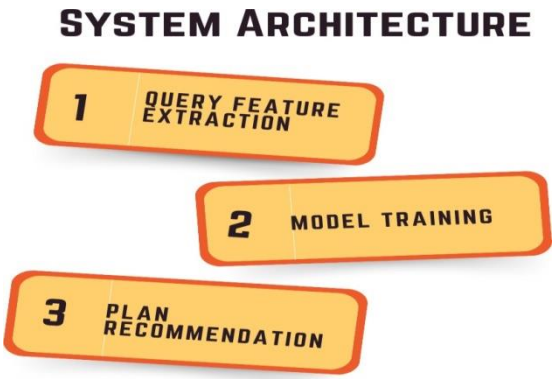

Figure 2. System Architecture

- **Query Feature Extraction:** The initial phase of the architecture is to convert SQLs to data as structured feature vectors to be ingested by machine learning models. [10-12] These query features are obtained based on query elements like operators (e.g., selections, projections, joins), join graphs, predicates as well as complexity of query structure. The metadata such as table sizes, the availability of the indices, and estimated selectivities are also encoded to lose the logical and physical characteristics of the query. This systematic description gives the basis of patterns of learning query behavior.
- **Model Training:** During the second step, machine learning models are optimized to learn query performance characteristics. Prediction of execution times and costs is performed using supervised learning models using past execution logs, which allow a reasonable prediction of the performance. Parallel to this, reinforcement learning (RL) models have to be trained to experiment with various join orders and operator decisions and query optimization is formulated as a sequential decision-making problem. Supervised and RL means that the stabilization and optimization methods are predictable and adaptive in nature.
- **Plan Recommendation:** The last phase is the execution of a hybrid recommendation engine wherein the output of the prediction and RL unit is incorporated. In this module candidate query execution plans (QEPs) are evaluated and the one that reduces the execution time with a robust behavior at different workloads is chosen. The hybrid architecture strikes a balance between the predictability of supervised learning and the flexibilities of the reinforcement learning process and offers a complete end-to-end implementation that generates near optimal query plans without depending entirely on the conventional cost models.

3.2. Flowchart of Proposed ML-driven Query Optimizer

Figure 3. Flowchart of Proposed ML-driven Query Optimizer

- **Query Input:** It starts with receiving an SQL query whereby a user sends the query to the optimizer. Here, the query structure and the context information (e.g. schema metadata, table statistics, workload environment) is captured by the system. This crude query is considered the entry point of the optimization pipeline.
- **Feature Engineering:** Then the query can be converted into a feature representation that can be used by machine learning models. This includes retrieval of information like operators, join types, predicates, data sizes and index availability. The feature engineering makes queries numerically or graph based to allow models to learn the performance characteristics.
- **ML Models:** The designed aspects are then transferred to machine learning models to fulfill various processes. Learning models Supervised learning models are learned to predict the cost of executing queries, whereas reinforcement learning models investigate a sequence of decisions such as ordering joins and choice of operators. All these models will be learnt via historical data and can be adjusted to workload fluctuations.
- **Cost Prediction:** One of the things that are predicted at this stage is the cost or time that is estimated of the candidate plans by supervised models. The predictions substitute or complement the traditional cost models and save the use of hand-crafted heuristics. Factual cost forecasting assists the system in assessing various plan options in a better way.
- **Plan Recommendation:** Based on input of the cost prediction and RL models, the optimizer structures a list of the candidate query execution plans (QEPs). The hybrid recommendation strategy chooses the plan that has the best estimated execution cost taking into consideration the workload dynamics, making sure the plan that is chosen is efficient and robust.
- **Execution:** Lastly, it is the database engine that runs the chosen plan. Feedback about execution (e.g. runtime used, resource used) is recorded and fed back into the training pipeline, where the models can then update their predictions and over time adjust to changing query loads.

3.3. Query Representation Model

The design of a query optimizer with ML-driven approach requires an efficient representation of queries because it allows

models to learn intrinsic [13-16] structure and semantic nature of query in a machine-readable format. The queries have been conceptualized in the proposed framework with a combination of operator sequences, predicate trees and join graphs, tapped in different dimensions of query behavior. Operator sequences are definitions of the sequence of relational operators- selection, projection, aggregation, and join - that constitute the logical plan of execution of a query. The ability to represent queries as sequences, enables machine learning models, and especially sequence-based models to learn how different operators relate to each other and their effect on the cost of execution. Predicate trees represent, insof as, the Boolean conditions that are defined in query filters in selection clauses. These trees denote hierarchical structure between predicates, and are used to express logical operators (AND, OR, NOT), as well as attribute-level predicates. This modeling of queries facilitates a better estimation of selectivity by the system which is essential to an effective estimation of cardinality and cost. Lastly, join graphs give a structural representation of multi-table queries based on modeling the tables as nodes and the join expressions as edges. The connectivity and complexity of the join relationships is encoded in this graph-based representation, which allows the graph-based models to reason about various join orders, as well as the effect each will have on the query performance. The combination of these three representations operator sequences, predicate trees, and join graphs, provides an overall picture of the characteristics of queries. They enable machine learning models to generalize when the query workload is varied, detect structural similarities, and to predict performance at higher accuracy. Besides, utilizing various complementary representations, the optimizer can overcome the weaknesses of every method, which creates a more robust and adaptable query optimization pipeline. This form of representation approach is the foundation of the advocated ML-based optimizer, since by so doing, synthetic as well as semantic features of queries are incorporated to facilitate learning and decision- making.

3.4. ML Models Used

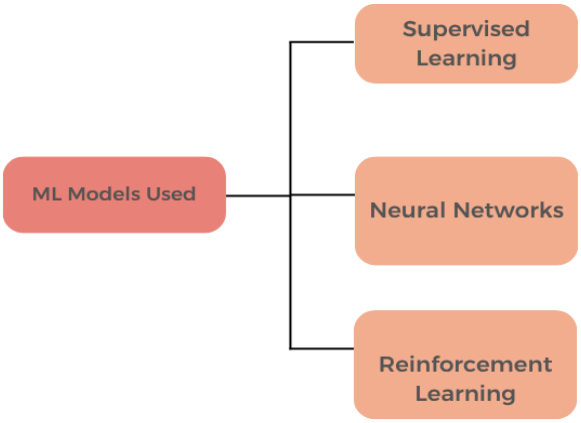

Figure 4. ML Models Used

- **Supervised Learning:** GBTs are used in predicting costs because of its high effectiveness in dealing with non-linearities in tabular data and in making use of a range of structured data. GBTs are trained on historical execution logs and are able to estimate the cost model more precisely when compared to traditional handcrafted estimators. Their ensemble character minimizest overfitting and give strong predictions and hence fit to overcome the evaluation of candidate plans in varying workloads.
- **Neural Networks:** Rich feature embeddings are produced on complex query representations like operator sequences, predicate trees and join graphs using deep neural networks. These learnings convert high-dimensional query inputs into dense, semantically and structurally connected, vectors. Taking advantage of architectures (including feedforward layers, convolutional networks, or graph neural networks), the model is capable of modeling interactions between query components, which yields better downstream tasks (such as cost prediction and plan ranking).
- **Reinforcement Learning:** To the problem of adaptive plan generation, reinforcer learning, namely: Q-learning, is used. The optimizer in this context takes the form of an agent who sequentially chooses actions which may be join orders or operator choices. Q-learning provides the agent with the chance of learning a policy that will maximize reward depending on real query execution efficiency. Contrary to traditional models, in this model, the optimizer can be dynamically adjusted to changes in data distributions and workload patterns, so it is very convenient in practice.

3.5. Training Pipeline

The proposed ML-driven query optimizer has the following training pipeline, which uses previous query log as the main data source, [17-20] where specific emphasis is given to widely recognized benchmark datasets, including TPC-H. Such logs offer a deep well of labeled data in which every query has execution statistics such as execution runtime, resource consumption, and selected execution plan. The initial stage of the pipeline is preprocessing and feature engineering, which parses queries into structured form (operator sequences, predicate trees and join graphs) and converts them to numerical feature vectors that can be inputted into machine learning models. In order to generalize, the dataset shall be separated into

training, validation, and test sets to provide the models with the opportunity to test themselves in the conditions that suggest the workload changes in the real world. In the case of supervised learning models, which are required to predict costs, a loss function is an important component. In this study, the Mean Absolute Error (MAE) has been selected because it is simple and interpretable because it simply quantifies the mean absolute deviation between the predicted and the actual costs of execution. Loss function is defined as:

$$L = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i|$$

where yi is the observed cost of query i, and $y\hat{}i$is the predicted cost . It is better than using squared-error metrics because MAE is not sensitive to outliers, so it is harder to break in an environment where query runtimes can get very large. This loss function is optimized during training of gradient boosted trees and deep neural networks, and hyperparameters are tuned using a grid search or Bayesian optimization. Parallel to this, reinforcement learning agents e.g. Q-learning agents are trained to engage with simulated environments that emulate the behaviour of running query execution on the simulated environment where observed run times are treated as reward signals. This two-fold training procedure partially guarantees the optimizers are trained on correct prediction of costs as well as the adaptive plan generation strategies making the system robust and generalizable.

3.6. Hybrid Recommendation Framework

The given hybrid recommendation architecture is promoted to use the synergist power of supervised learning and reinforcement learning to optimize queries and present a more precise and adapt high-quality query implementation plans (QEPs). The use of supervised learning models, including gradient boosted trees or deep neural networks, is used in this framework, mainly to estimate costs. Such models are trained using historical logs of query executions to offload the anticipated execution time, resource use or other performance measures of potential plans. With precise cost prediction, through supervised models, the optimizer can early eliminate plans that are infeasible or suboptimal, which makes the search space smaller and the overall efficiency better. In complement to this, reinforcement learning (RL) models, e.g. the query plan generation sequential decision making space, are explored using reinforcement learning (RL) agents, e.g. Q-learning agents. The RL agent views the choice of join orders, selection of operators and access paths as actions in a Markov Decision Process with execution performance performance acting as rewards. The agent eventually develops a policy that trades off between exploring new strategies and exploiting the old successful ones, and can respond to the challenge presented by changing workloads and new query patterns. These two strategies are combined to create a hybrid framework in which the cost predictions are used to inform the RL agent to focus on potentially beneficial regions of the search space and the RL agent provides adaptability and flexibility not available to static models. This integration lessens the drawbacks of only using conventional cost models or only learned strategies. Continuous learning is also favored with the hybrid framework: logged execution feedback of deployed queries is fed back into the supervised and RL models, and execution feedback can be updated incrementally, with better predictions as time progresses. The frame-work has a balance between efficiency, robustness, and adaptability by combining accurate cost estimation and adaptive plan selection and is therefore best adapted to work with complex and rapidly changing database loads.

4. Results and Discussion

4.1. Experimental Setup

The proposed ML-based query optimizer is experimentally evaluated with the help of the well-known TPC-H benchmark which offers a standardized set of queries and data schema to evaluate the performance of decision support systems. This paper uses a 100 GB sized dataset to model an approximate realistic, large scale analytical workload which incorporates using intricate join operations, aggregations, and nested queries. TPC-H benchmark guarantees the same reproducible and comparability of the results because its query templates provide numerous selectivity patterns, the complexity of the join graphs and the predicates. The PostgreSQL query optimizer provides the baseline in order to evaluate the effectiveness of the proposed approach. The optimizer in PostgreSQL is a full-fledged cost based optimizer that produces query execution plans based on classic statistical models, such as cardinality estimates and access path selections. The experiments, comparing the ML-based optimizer with PostgreSQL should provide measurements of the performance improvements (execution time, plan quality and robustness) with different query loads. The experimental configuration also involves close instrumentation to voice query execution statistics, including runtime, CPU, I/O, and mem usage statistics. Queries are repeatedly run to consider the effect of caching and variability of the system, and the average performance is reported. Moreover, the ML-based and baseline optimizers are tested using the same system settings, which is a fair comparison. The ML models are trained with the historical logs of queries based on the same TPC-H data with hyperparameters being adjusted to maximize the accuracy of predictions and reinforcement learning. In general, this experimental design offers a realistic but evaluative experiment. It enables a detailed comparison of the hybrid ML-based optimizer with a classic cost-based system, which shows the possible improvements of the hybrid system in terms of the efficiency of its execution, dynamic plan creation, and robustness to complex and dynamic workloads.

4.2. Performance Metrics

Table 1. Performance Metrics

Query Type	Improvement (%)
Simple Select	15.5%
Complex Join	30.9%
Aggregation	20.8%

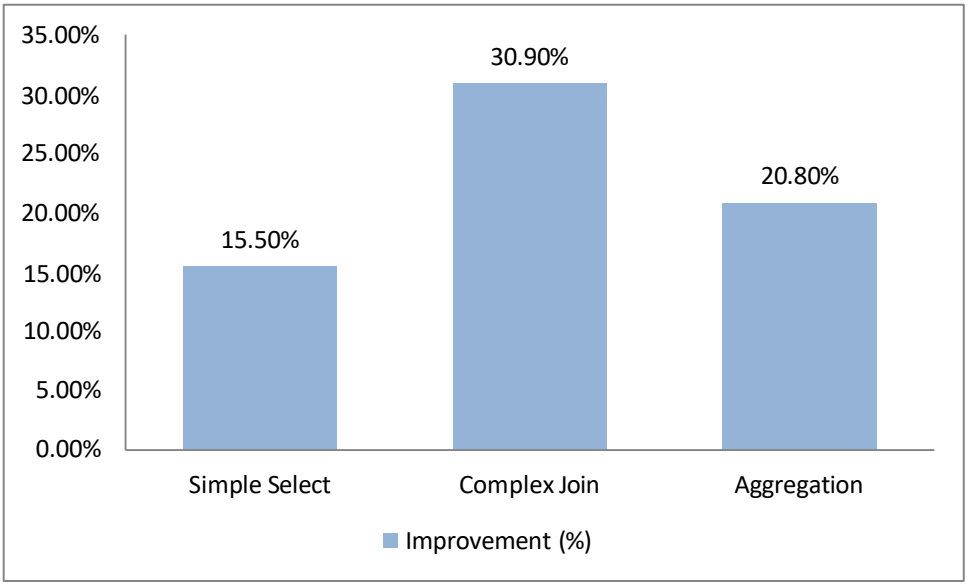

Figure 5. Graph representing Performance Metrics

- **Simple Select Queries:** Simple select queries will usually make use of a small subset of a single table querying any optional filtering predicates. These queries are typically not as complicated and contain fewer join operations, and are simpler to plan by both traditional and ML-based optimizers. The ML-based optimizer improved the performance of PostgreSQL baseline by 15.5 per cent in the experiments. This is credited to better prediction of cost and efficient indexes selection so that the optimizer may save unnecessary scans and save time in query execution even on relatively simple queries.
- **Complex Join Queries:** Complex join queries are queries that require various tables and in fact, have intricate conditions of join, usually in conjunction with filtering predicates and having functions. These questions are a major problem to the traditional cost based optimizers as the set of join order grows exponentially with the number of tables. The ML- based optimizer improved these queries by 30.9 percent, indicating that the optimizer can use learned patterns and reinforcement learning algorithms to search through join orders in a better way. The system has saved significant execution time compared to the baseline by determining efficiently the cost of the plans in advance and by dynamically adjusting the plan selection policy.
- **Aggregation Queries:** Aggregation queries are for summary statistics, like COUNT, SUM or AVG, frequently on filtered or grouped data. Efficient execution is not just a matter of access paths, but also a matter of operator strategies of access paths, operator ordering, and grouping. The ML-based optimizer generated a 20.8% based improvement on aggregation queries in comparison to the baseline. The advantage here is that it also uses a method of supervised cost estimation and reinforcement learning to choose the plans, which maximize both the data retrieval and the aggregation processes, thus leading to less calculating and quicker responses to queries.

4.3. Key Findings

The experimental analysis of the performance and capabilities of the query optimizers based on the ML versus the traditional cost-based system has a number of significant points. Ultimately, optimizers based on ML have been proven to be faster than their predecessors, like the PostgreSQL, in response time in executing queries of different kinds, including simple selects, complex join, and aggregation functions. This improvement of performance can be largely explained by the capability of supervised learning models to accurately estimate the cost of execution on the basis of historical query logs, which can be used by the optimizer to early eliminate suboptimal plans and concentrate on candidates with optimistic promise. The enhancements are especially notable when it comes to intricate queries that have more than one join since the traditional variant of statistical estimation can be erroneous because of the interdependence between the variables or a distorted data distribution. Second, the reinforcement learning aspect of the hybrid optimizer can exhibit great adaptiveness to changing workloads. As a form of sequential decision-making problem, RL can be modeled to continuously learn and improve the

policy on execution feedback, and adapts its policy to changing data distributions, query seasonalities that arise and disappear, and shifting workload demands. This adaptability enables the optimizer to perform very well even in a situation where cost-based methods will not perform so well when at rest. Besides, the synergistic combination of the two types of learning, supervised learning and precise cost estimation and reinforcement learning and adaptive plan selection, constitutes a balanced reliability/flexibility structure. The hybrid method decreases variation in execution, enhances resistance to outliers and alleviates the shortcomings of the solely traditional or solely learned optimization strategies. On balance, these results suggest that the concepts of machine learning can not only make query processing pipelines run faster, but also serve as the foundation of more intelligent, self- learning database systems. The findings are a strong indication that the approach of ML is highly applicable to current analytical tasks that are characterized by high complexity and dynamic nature.

5. Challenges and Limitations

Although the ML-based query optimizers can provide valuable processing performance compared to the traditional cost-based system, there are various challenges and limitations that should be taken into account. Among the main issues, there is that supervised and reinforcement learning models have a high training overhead. Supervised learning models need large amounts of historical query execution logs to help them make correct predictions of the execution costs, and neural networks or gradient boosted models might have complicated feature engineering steps and hyperparameter optimization. Agents that use reinforcement learning, specifically, require a large number of queries to the query plan space in order to learn effective policies, which can be computationally and time-intensive. This initial training may prove to be a bottleneck to deployment in a production setting where resource or time is limited. The second major weakness is that retraining and updating models are required on a continuous basis in order to sustain performance amidst the changing workloads. Database environments tend to be dynamic and alterations in data distribution, changes in table sizes, changes in indexing patterns or changes in query patterns often occur with time. Without frequent retraining of models, their predictions can become unreliable and with an incorrect prediction of optimal plan choice and worse query performance may be achieved. Moreover, the process of incorporating learned models into existing database systems remains very complex since compatibility with old query engines, explainability, as well as monitoring model behaviour has to be taken care of. Learned models are also uncertain especially on processing out of distribution queries that vary greatly on training data. This may lead to random plan decisions or deterioration of performance. Lastly, although ML-based methods may lower the time of execution of a large number of queries, the benefits might not always be appropriate to balance the computational and operation cost in smaller workloads or less complex queries. To meet these challenges, strategies to overcome these issues will include incremental training, online education, hybrid structures, balancing between the learned and traditional methods, and close observation of system performance to guarantee that the ongoing use of the ML-based optimizers will deliver consistent and reliable advantages without imposing unnecessary loads on the operation.

6. Conclusion

This paper has given a detailed research on the use of machine learning algorithm in query optimization, and that learned methods can supplement and complement customary techniques of optimization of cost. Through examination of logs of historical query executions and using structural representations of queries, including the order of operators, predicate trees and join graphs, the proposed system could identify meaningful patterns and correlations which more traditional statistical models frequently overlook. The execution costs could be predicted with accuracy using supervised learning models that included gradient boosted trees and deep neural networks, which then allowed the optimizer to more efficiently analyze candidate query execution plans. Simultaneously, reinforcement learning agents, especially Q-Learning agents, were incorporated into the pipeline to search sequential decision-making spaces, including join ordering and operator selection, which enables optimizer to discover plans adaptively as the result of feedbacks to previous executions. This hybridization of the methods into a composite recommendation processing was found to yield great improvements when compared to conventional optimizers in terms of their budgeted performance and resilience in different query load conditions. An experimental analysis of the TPC-H benchmark, in a 100 GB scale proved that the ML based optimizer was much better than the PostgreSQL base optimizer, especially in complex join and aggregation queries, showing its effectiveness in the ability to cope with complex query format and dynamically changing data distributions.

Although these results were promising, a number of challenges were revealed, such as training overhead, the necessity of continuous retraining to suit the changing workloads, and limitations to the interpretation of models when they are applied to a production database context. These issues are important and require attention to help to make the use of ML-based optimizers in a real system practical. The directions of future research are on how to enhance the transferability of the learned models to other heterogeneous database management systems so that a single trained model can be effectively transferred to multiple platforms. Also, the interpretability gap of complex ML models should be minimized in order to develop trust and comprehension among database administrators especially when the learned plans are not aligned with the conventional expectations. Lastly, low overhead online retraining and incremental learning programs could also be designed with a high level of performance with respect to maintaining the proven performance of the model using smaller computational costs so that the optimizer can be sensitive to the workload variations. Comprehensively, this paper shows that, machine learning integration in query optimization process is an effective approach to more intelligent, adaptive, and efficient database systems,

forming the basis of next-generation self-optimizing databases, which can continuously learn through operating experience.

References

[1] Selinger, P. G., Astrahan, M. M., Chamberlin, D. D., Lorie, R. A., & Price, T. G. (1979, May). Access path selection in a relational database management system. In Proceedings of the 1979 ACM SIGMOD international conference on Management of data (pp. 23-34).

[2] Chaudhuri, S. (1998, May). An overview of query optimization in relational systems. In Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems (pp. 34-43).

[3] Kruse, S., Kaoudi, Z., Contreras-Rojas, B., Chawla, S., Naumann, F., & Quiané-Ruiz, J. A. (2020). RHEEMix in the data jungle: a cost-based optimizer for cross-platform systems. The VLDB Journal, 29(6), 1287-1310.

[4] Tekale, K. M. T., & Enjam, G. reddy . (2022). The Evolving Landscape of Cyber Risk Coverage in P&C Policies. International Journal of Emerging Trends in Computer Science and Information Technology, 3(3), 117-126. https://doi.org/10.63282/3050-9246.IJETCSIT-V3I1P113

[5] Krishnan, S., Yang, Z., Goldberg, K., Hellerstein, J., & Stoica, I. (2018). Learning to optimize join queries with deep reinforcement learning. arXiv preprint arXiv:1808.03196.

[6] Heitz, J., & Stockinger, K. (2019). Join query optimization with deep reinforcement learning algorithms. arXiv preprint arXiv:1911.11689.

[7] Marcus, R., & Papaemmanouil, O. (2018). Towards a hands-free query optimizer through deep learning. arXiv preprint arXiv:1809.10212.

[8] Yang, Z., Chiang, W. L., Luan, S., Mittal, G., Luo, M., & Stoica, I. (2022, June). Balsa: Learning a query optimizer without expert demonstrations. In Proceedings of the 2022 International Conference on Management of Data (pp. 931-944).

[9] Tekale, K. M., & Rahul, N. (2022). AI and Predictive Analytics in Underwriting, 2022 Advancements in Machine Learning for Loss Prediction and Customer Segmentation. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 3(1), 95-113. https://doi.org/10.63282/3050-9262.IJAIDSML-V3I1P111

[10] Marcus, R., Negi, P., Mao, H., Tatbul, N., Alizadeh, M., & Kraska, T. (2021, June). Bao: Making learned query optimization practical. In Proceedings of the 2021 International Conference on Management of Data (pp. 1275-1288).

[11] Yu, X., Chai, C., Li, G., & Liu, J. (2022). Cost-based or learning-based? A hybrid query optimizer for query plan selection. Proceedings of the VLDB Endowment, 15(13), 3924-3936.

[12] Zahir, J., & El Qadi, A. (2016). A recommendation system for execution plans using machine learning. Mathematical and Computational Applications, 21(2), 23.

[13] Akdere, M., Çetintemel, U., Riondato, M., Upfal, E., & Zdonik, S. B. (2012, April). Learning-based query performance modeling and prediction. In 2012 IEEE 28th International Conference on Data Engineering (pp. 390-401). IEEE.

[14] Hellerstein, J. M. (1998). Optimization techniques for queries with expensive methods. ACM Transactions on Database Systems (TODS), 23(2), 113-157.

[15] Ramadan, M., El-Kilany, A., Mokhtar, H. M., & Sobh, I. (2022). Rl_qoptimizer: a reinforcement learning based query optimizer. IEEE Access, 10, 70502-70515.

[16] Fent, P., & Neumann, T. (2021). A practical approach to groupjoin and nested aggregates. Proceedings of the VLDB Endowment, 14(11), 2383-2396.

[17] Hamilton, W., Bajaj, P., Zitnik, M., Jurafsky, D., & Leskovec, J. (2018). Embedding logical queries on knowledge graphs. Advances in neural information processing systems, 31.

[18] Marcus, R., & Papaemmanouil, O. (2019). Flexible operator embeddings via deep learning. arXiv preprint arXiv:1901.09090.

[19] Zongheng Yang, Machine Learning for Query Optimization, online. https://www2.eecs.berkeley.edu/Pubs/TechRpts/2022/EECS-2022-194.pdf

[20] Jalali, S. M. J., Osório, G. J., Ahmadian, S., Lotfi, M., Campos, V. M., Shafie-khah, M., ... & Catalão, J. P. (2021). New hybrid deep neural architectural search-based ensemble reinforcement learning strategy for wind power forecasting. IEEE Transactions on Industry Applications, 58(1), 15-27.

[21] Sharma, M., Singh, G., & Singh, R. (2019). A review of different cost-based distributed query optimizers. Progress in Artificial Intelligence, 8(1), 45-62.

[22] Kaoudi, Z., Quiané-Ruiz, J. A., Contreras-Rojas, B., Pardo-Meza, R., Troudi, A., & Chawla, S. (2020, April). ML-based cross-platform query optimization. In 2020 IEEE 36th International Conference on Data Engineering (ICDE) (pp. 1489-1500). IEEE.

[23] Tekale, K. M. (2022). Claims Optimization in a High-Inflation Environment Provide Frameworks for Leveraging Automation and Predictive Analytics to Reduce Claims Leakage and Accelerate Settlements. International Journal of Emerging Research in Engineering and Technology, 3(2), 110-122. https://doi.org/10.63282/3050-922X.IJERET-V3I2P112