Enhancing Database Performance through a Comparison of Traditional and AI-Driven Query Optimization Techniques

Uzo Blessing Chimezie[1], Atanda Aminat Oluchi[*1], Levi Arinze Ugwu[2]

[1]Department of Computer Science, Faculty of Physical Sciences, University of Nigeria, Nsukka

[2]James Cook University, Cairns, Australia

DOI: https://doi.org/10.51244/IJRSI.2025.120700133

Received: 03 July 2025; Accepted: 07 July 2025; Published: 07 August 2025

ABSTRACT

Query optimization is a fundamental aspect of database management systems (DBMS), crucial for enhancing query performance and resource utilization. Traditional methods for optimising queries can handle most cases, but they are inflexible in dealing with difficult or highly dynamic query workloads. New approaches in AI and ML are now available to help overcome these challenges, supporting more flexible optimization methods. This paper presents a comparative study of traditional and AI-driven query optimization approaches. Using synthetic data, we evaluate multiple machine learning models, including CatBoost, LightGBM, and ExtraTrees, against a traditional rule-based baseline that relies on indexing heuristics. The results show that AI-driven models, particularly CatBoost, achieved an accuracy of 59%, outperformed traditional methods (48%) in terms of accuracy, precision, recall, and F1-score, highlighting their ability to capture complex feature interactions and improve query execution efficiency. Despite the promising performance of AI models, the paper also discusses the trade-offs involved, including the interpretability challenges and computational overhead associated with AI-based approaches. This study demonstrates the potential of AI-driven optimizers as a valuable tool for modern DBMSs, particularly in dynamic and high-complexity environments.

Keywords: Query Optimization, Machine Learning, CatBoost, Database Management Systems (DBMS), AI-driven Optimization

INTRODUCTION

Modern information processing depends on DBMS, which organise how data is stored, called up and changed on computers. Because databased decision-making is taking over, query processing must become more efficient. The process also depends on query optimization since it guarantees queries run at best speed, minimising resources used and the time it takes to respond. Since the amount and complexity of data is increasing rapidly these days, optimising queries can mean the difference between good performance and poor results [1]. For a long time, improving a database's performance has depended on things like indexing, reordering joins and cost-based optimization. Important methods use set guidelines and data models to estimate how to run a query [2]. For example, indexing simply holds the data in a reference structure that speeds up looking for it and join reordering works to improve how joins are carried out in your query. In the same way, cost-based optimization studies the efficiency of several query plans and finds the one that costs least. While these approaches have improved how queries run, they are not very effective with the complexity and fast changes in today's data environments. With larger and more complex data, methods from the past usually have problems adjusting quickly to the added data types and the way systems are organised [3].

Over the past few years, AI and ML have revolutionised how query optimization is done. Unlike older methods, AI methods use information from earlier times to modify the execution of queries on the fly [4]. With models such as reinforcement learning and neural networks, machine learning has demonstrated that query optimization can be much improved by making the system capable of learning from growth and change in workloads. Because of this, the approaches can respond to changes in how data is structured and how its queried, allowing for quicker and more accurate optimizations [5]. Even so, direct comparison of traditional

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XII Issue VII July 2025

query optimization with AI approaches has not been fully explored in research. There is a demand for comparing such algorithms because database queries have become very complex and resulting optimization strategies must be more intelligent. This paper is designed to fill this gap by presenting a study that compares how traditional and A.I. systems optimise queries. By analysing experiments, we measure the speed, effectiveness and ability to scale both answers.

Traditional Query Optimization Techniques

Optimising a query has always played a crucial role in database management systems (DBMS) to improve how SQL queries are carried out. Several steps, statistics, rule-based transformations and other techniques are used in traditional query optimization to reduce resource use and cut query response time [1].

Key Concepts and Phases in Query Processing

Query processing involves several critical stages, each contributing to the construction of an efficient query execution plan:

- Parsing and Translation: During this step, the DBMS analyses the SQL question to ensure it is written correctly, and then produces a parse tree. The query is expressed in a relational algebra formula, which forms the basic logic for optimization [6].

- Optimization: Different possible plans are made using relational algebra by the optimizer. Several plans are worked out, highlighting the ways, they might be implemented and then compared to find the best one based on its resource use [7].

- Evaluation: The final step is having the query engine use the plan chosen by the optimizer to bring out the desired results [7].

These phases work in sequence to systematically transform a user's high-level query into a highly efficient physical execution strategy.

Statistical Information for Cost Estimation

An integral aspect of traditional optimization is the use of statistical information to predict the cost of various query execution alternatives. This information, stored in the database catalog, includes:

- Cardinality: The number of distinct values in a column or a combination of columns, used to estimate result sizes [8].

- Selectivity: The fraction of tuples that satisfy a given predicate, crucial for assessing the effectiveness of filtering conditions.

- Histograms: Representations of the data distribution within a column, which help the optimizer understand data skew and clustering effects [9].

- Index Statistics: Metadata about indexes, such as index height and leaf node distribution, critical for estimating the cost of index scans.

During query optimization, these statistics are retrieved and employed by cost models to predict the relative performance of competing query plans. Accurate, up-to-date statistics are essential; outdated or incomplete statistics can lead to severely suboptimal execution plans [7].

Transformation and Equivalence Rules

Traditional query optimizers leverage a set of equivalence rules to transform query expressions into logically equivalent but potentially more efficient forms:

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XII Issue VII July 2025

- Join Reordering: Rearranging the order of join operations can significantly reduce intermediate result sizes and improve performance [10].
- Predicate Pushdown: Moving selection operations closer to the base relations reduces the number of rows processed in subsequent operations.
- Algebraic Simplifications: Removing redundant projections, selections, and operations that do not contribute to the final result [11].

Using these transformation rules, the optimizer explores a large search space of alternative plans, improving the chances of finding an optimal or near-optimal plan.

Optimizing Nested Subqueries

Nested subqueries are common in SQL, but their naive execution often results in inefficient plans. Traditional optimizers address this by:

- Subquery Unnesting: Converting nested subqueries into joins, which are easier to optimize and execute [7].
- Temporary Tables: Materializing subquery results to avoid repeated execution.
- Semi-joins and Anti-joins: Rewriting EXISTS and NOT EXISTS subqueries into specialized join forms that can be processed more efficiently.

These transformations help reduce the computational complexity associated with executing complex nested subqueries.

Materialized Views and View Maintenance

Materialized views precompute and store the results of complex queries, offering significant performance improvements for frequent queries. They are particularly useful in decision support systems and data warehouses [12].

However, maintaining materialized views presents challenges:

- Immediate View Maintenance: Refreshing the view instantly after each base table update.
- Deferred View Maintenance: Periodically refreshing the view, suitable when slight data staleness is acceptable.

Triggers, batch jobs, or refresh-on-demand mechanisms are often used to maintain materialized views.

Common Traditional Optimization Techniques

Several widely used techniques underpin traditional query optimization:

- Cost-Based Optimization (CBO): The optimizer evaluates multiple execution plans and selects the one with the lowest estimated cost based on resource usage statistics [1].
- Rule-Based Optimization (RBO): Plans are generated based on a predefined set of rules and heuristics, independent of actual data statistics [3].
- Indexing: Creating structures like B-trees and hash indexes to speed up data retrieval based on key columns [4].

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XII Issue VII July 2025

- Partitioning: Dividing large tables into smaller, more manageable partitions (horizontal or vertical), improving query performance by reducing data scanning.
- Denormalization: Introducing redundancy into a database schema to reduce the number of joins required during query execution, thus improving performance for read-heavy workloads.
- Caching: Storing the results of frequently executed queries in memory to avoid repeated computation.
- Query Rewriting: Modifying user-submitted queries into logically equivalent forms that are easier or faster to process, such as converting correlated subqueries into joins [5].
- Parallel Execution: Distributing query execution tasks across multiple processors or servers, enabling faster processing of large-scale queries.

These techniques have formed the backbone of database optimization strategies for many years. However, they often struggle to adapt dynamically to unpredictable workloads and evolving data characteristics, motivating the exploration of more intelligent, adaptive methods such as AI-driven optimization.

AI-Driven Query Optimization Techniques

The rise in the complexity and size of modern data systems means that the traditional methods of optimising queries are no longer well suited to problems we see now. As a result, Artificial Intelligence (AI) and Machine Learning (ML) methods have become important solutions, as they can analyse previous data and help make better optimization decisions as time passes. This optimization method boosts performance by studying query behaviour, predicting the best way to run queries and adjusting to changes that occur continually.

Overview of AI in Query Optimization

Efforts to use AI for query optimization are typically categorised as supervised learning or reinforcement learning. Here, models learn by analysing historical query execution data, where the different features of a query (e.g. its design, the size of involved tables and indexing options) are linked to outcomes such as different scan types, execution cost or the plan showing the best performance. Once it is ready, the model can suggest execution paths for new, unfamiliar queries with little help from people [1].

Reinforcement learning works by having the query optimizer take different actions (e.g. which join to use which operation to use) and observing the results in terms of outcomes including cost and time [13].

These techniques are more useful than classical optimizers when either the statistical metadata is absent or the data changes very quickly. They are able to adjust to different situations and deliver strong optimization recommendations that usually work better than those made by rules or guidelines.

Machine Learning Models for Query Plan Prediction

In this study, multiple machine learning models were implemented and evaluated for their ability to predict efficient query execution plans. The models include:

- Logistic Regression: A baseline linear model used for binary classification problems.
- Random Forest: An ensemble method that builds multiple decision trees and averages their predictions to reduce variance [14].
- XGBoost (Extreme Gradient Boosting): A high-performance gradient boosting algorithm known for its predictive accuracy and scalability [15].
- LightGBM (Light Gradient Boosting Machine): A gradient boosting framework optimized for speed and efficiency in large datasets [16].

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XII Issue VII July 2025

- CatBoost: A gradient boosting model particularly suited for categorical data, offering strong generalization performance with minimal preprocessing [17].

Features including table size, query complexity, whether or not there is an index and the time it takes to complete the query were used in the process of training every model. The purpose was to foresee whether a query would trigger a table scan or an index scan.

Test results show CatBoost achieved the best accuracy and F1-score at correctly determining the differences between different query execution strategies. Kaggle was successful largely because it works with categorical data well and does not overfit.

Justification for AI Approach and Data Generation

Since no datasets exist that clearly link the different features of SQL queries to their execution plan labels, the study used a model that imitated average query workloads and operations. This method provides a way to experiment and test ideas, even though outdoor data is still needed for better results [18].

In addition, the accuracy of the models was measured along with their adaptability to changes in query aspects and datasets. Because of this situation, adaptive query optimizers are necessary in latest DBMS environments.

Experimental Setup and Results

In this section, a description of the experimental framework for evaluating both traditional and AI-driven query optimization approaches is presented. The process requires making data, training models, checking performance and analysing classification outcomes with confusion matrices.

Data Generation and Features

Since there are no public datasets that directly link SQL query features with strategies for running them, we created a synthetic dataset to cover similar tasks. The dataset included the following features:

- Table Size: Numeric representation of the number of rows.
- Query Complexity: Integer value denoting query operations (e.g., joins, filters).
- Has Index: Binary indicator (1 for presence of index, 0 otherwise).
- Execution Time: Simulated runtime based on complexity and indexing.
- Scan Type: Target variable indicating the predicted access method—table scan (0) or index scan (1).

The data for this study included 1,000 randomly generated samples. For the data, 80% was used for training and 20% was used for testing. In order to tackle unequal class distribution in the training set, SMOTE (Synthetic Minority Oversampling Technique) was implemented [1]. Data was converted into z-scores so that features could be standardized.

Models and Evaluation Metrics

Three machine-learning models were tested as AI-driven optimizers:

- LightGBM – a gradient boosting model optimized for efficiency [16],
- ExtraTreesClassifier – an ensemble-based classifier using randomized decision trees [18],
- CatBoostClassifier – a gradient boosting model known for handling categorical features effectively [17].

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XII Issue VII July 2025

All models were trained on the same training set and evaluated on the same test set. The following performance metrics were computed:

- Accuracy – overall proportion of correct predictions,
- Precision – the proportion of predicted positive cases that were correct,
- Recall – the proportion of actual positive cases that were correctly predicted,
- F1-score – the harmonic mean of precision and recall.

RESULTS

The performance results for the three models are presented in Table 1:

Table 1: Performance Comparison of AI Models

Model	Accuracy	Precision	Recall	F1-Score
LightGBM	0.54	0.514	0.579	0.545
ExtraTrees	0.58	0.552	0.611	0.580
CatBoost	0.59	0.562	0.621	0.590

Compared to the other tested models, CatBoostClassifier delivered the highest accuracy of 59% and all three of its precision, recall and F1-score were the highest too. Accordingly, CatBoost is expected to perform well on new types of query workloads NOTICE.

Confusion Matrix Analysis

To gain deeper insight into model behavior, confusion matrices were generated for each classifier. These matrices visualize the number of correct and incorrect predictions across both scan types.

- LightGBM: Correctly predicted 53 table scans and 55 index scans; misclassified 52 and 40, respectively.

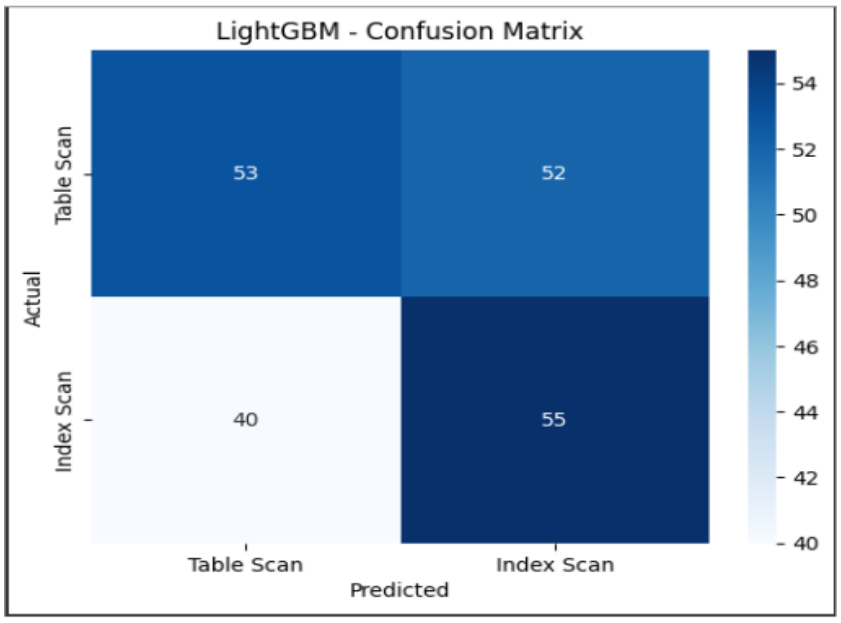

Figure 1: Confusion Matrix for LightGBM

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XII Issue VII July 2025

- ExtraTrees: Correctly predicted 58 table scans and 58 index scans; misclassified 47 and 37.

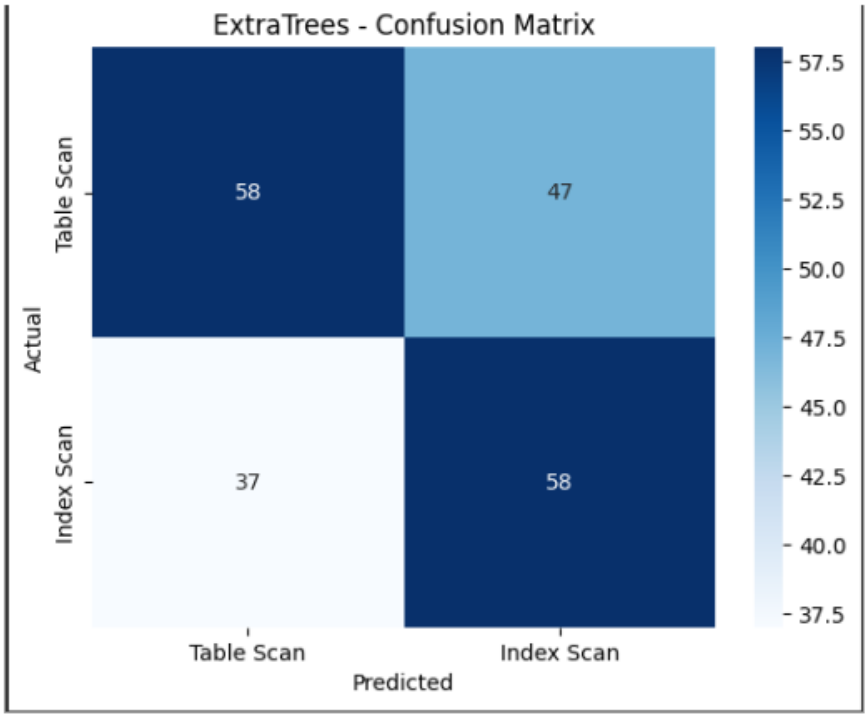

Figure 2: Confusion Matrix for ExtraTrees

- CatBoost: Achieved the best result, correctly identifying 59 table scans and 59 index scans; misclassified only 46 and 36.

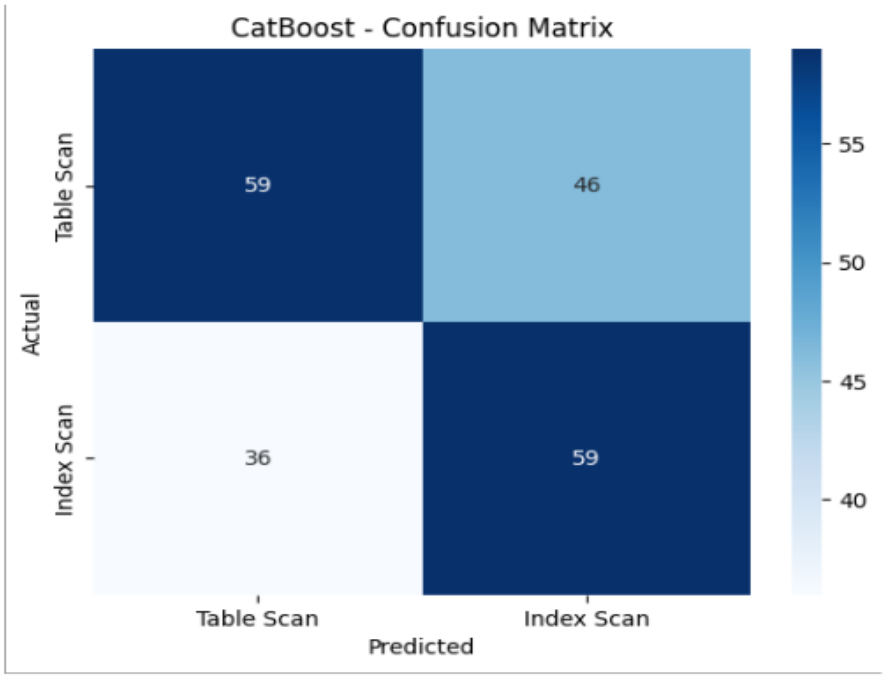

Figure 3: Confusion Matrix for CatBoost

Traditional Rule-Based Baseline

For comparison, a conventional rule-based method was applied in this study. All the system did was compare the feature Has_Index: if it were 1, the system would carry out an Index Scan, otherwise a Table Scan. It illustrates the fact that in standard query optimizers, choosing an access path most often depends on indexing heuristics.

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XII Issue VII July 2025

The model was evaluated on the same synthetic test dataset used for the AI-driven models. The results are shown below:

Table 2: Rule-based Traditional Approach Evaluation Result

Metric	Value
Accuracy	0.48
Precision (Index)	0.46
Recall (Index)	0.47
F1-Score (Index)	0.47

The confusion matrix in Figure 4 shows that the rule-based method correctly predicted 52 table scans and 45 index scans, while misclassifying 53 table scans and 50 index scans.

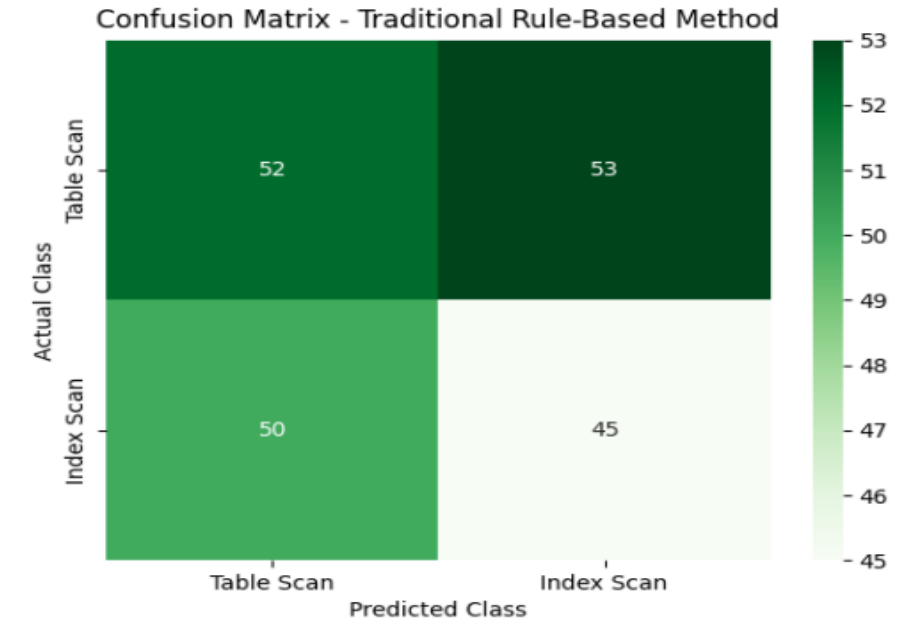

Figure 4: Confusion Matrix – Traditional Rule-Based Method

Despite the method performing slightly better than guessing randomly (50%), all AI models still did much better. This result shows that only using fixed rules to choose a query plan might not work well if you need to consider different related factors like query and data complexity together..

Limitations

While the results demonstrate a clear performance advantage of AI-driven models over the traditional rule-based baseline, several limitations must be acknowledged.

The dataset for this study was produced using simulations rather than real data because there are no publicly released, labelled datasets that link queries to their results. Despite controlled testing and reproducibility made possible by synthetic data, this approach misses much of the real-world noise, diversity and variety of actual queries received [19]. These points to a need to validate the findings again on real-worl dBs, using what is learnt from this Data Set. In addition, the models were tested to see if they could predict solely whether an operation would be a table scan or an index scan. Optimising a query actually involves planning how to perform joins which operators to use and using estimates to understand possible costs over multiple stages of the process. The compact nature of this study is good for pointing out differences, but it does not reveal how a

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XII Issue VII July 2025

real-world optimizer makes its decisions. Still, even though SMOTE worked to fix uneven class representation in the data, oversampling might cause the data to have artificially added patterns. If this type of uploaded data is handled without care, it might slightly improve recall scores [20].

The representation of characteristics involved just a few factors: table size, how complex the query is, if an index exists and total time spent processing the query. In production database systems, optimizers use a variety of extra statistics and background information. Enhancing the model by adding options such as query graph type, predicate's selectivity value or information about execution times increases its accuracy and can improve how it performs. Still, the results show that AI methods can do better than unchanging rules when query optimization involves uncertainty and when different features interact.

DISCUSSION AND COMPARISON

This section conducts a detailed analysis of the outcomes from experimentally comparing regular optimization with AI-based approaches. It studies the ways algorithms differ, the importance of feature relationships, how to use them for production and their connexion to past research. It demonstrates the importance of adding machine learning to today's query optimization workflows.

Performance and Model Behavior

The experiments show that AI models, with CatBoost leading, were much more effective than using rules to predict outcomes. The accuracy of the rule-based approach was 48% which is now improved to 59% by CatBoost. Despite appearing small relative to other QA systems, this result stands out as it tackled the tough issue of binary classification on a simple input dataset.

The traditional approach decided to use a table scan or index scan based only on the Has_Index feature. As a result of this method, misclassifications were common in situations where using an index would have been costly because tables were large or queries were complex. As a result, using multi-feature responses, AI models were able to highlight the significance of Table_Size, Query_Complexity and Execution_Time. Because of this, both groups were more equally classified, as the confusion matrices indicate.

CatBoost is highly effective because it deals well with category-type data and resists making models that fit the training set alone too well. Both ExtraTrees and LightGBM gave better results than the original model, even though their accuracy-precision-recall balance wasn't as good as CatBoost. Using these results, we confirm that ensemble learning techniques are important for handling structured data tasks in query optimization.

Feature Interaction and Misclassification Patterns

Analysis of the confusion matrices led to the discovery of the key reasons for the model's successes and failures. The rule-based model often suggested index scans that turned out to be less useful than table scans on big tables because the high cost of reaching an index outweighed its selective benefits. This problem arises when simple heuristics ignore cardinality, how predicates are involved and the form of the query [21]. AI algorithms, specifically CatBoost, found cases when having an index didn't result in improved performance. Such models show that non-linear interactions between query features can be performed, something that only extra effort with rules could accomplish in others. As a result, AI helps save engineers from spending much effort and attention updating traditional optimizers.

Interpretability, Adaptability, and Operational Fit

Optimizers built on traditional rules are frequently chosen for their low risk, understandable rules and clear details. Because they are always this way, it is easy to cheque and correct code written by them. Even so, these organisations find it challenging to adapt to new situations. In changing situations where input data and work changes rapidly, they usually perform below expectation and require administrators to constantly update their settings and refresh statistics [7]. These types of optimizers are able to respond and change according to the data they receive. After training, the machine offers long-term results by adapting each time new data for query execution is present. Yet, these techniques make things difficult to interpret because decisions come from

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XII Issue VII July 2025

learning instead of clear definitions. Although CatBoost and other models give more insight than deep learning, they might need tools like SHAP or LIME to explain the internal results for users and to support debugging when needed [22]. How expensive is it to computationally train a model is another issue to consider. Model inference takes only a short time, but setting up and testing everything becomes a challenge when the system is large. Even so, the expense is usually made up for in areas of high internet use, where even simple improvements can lead to great savings.

Alignment with Existing Research

The results support recent investigations recommending that machine learning be applied to database management systems. Previously, it was shown that algorithms trained from data do better than hand-engineered heuristics on tasks such as cardinality, deciding joins and selecting operators, most notably in complex situations [23]. Unlike certain other methods, the study here prefers tree-based models because they deal well with table format data and are less demanding on computers. As a result, they can be more easily used in live databases, where there is strong concern about impact on the system and how well they integrate.

Practical Implications and Deployment Scenarios

While this study used synthetic data, the features modeled index presence, query complexity, and execution cost mimic real-world optimization considerations. As such, the comparative findings suggest several practical insights:

- AI-driven optimizers are ideal for:
  - Systems with dynamic workloads and frequent schema changes,
  - Cloud-native environments where elasticity and cost-optimization are key,
  - Complex analytical queries involving multiple joins and nested subqueries.
- Traditional optimizers remain suitable for:
  - Simple, transactional workloads,
  - Resource-constrained environments where explainability and deterministic behavior are required,
  - Systems where data patterns are stable and predictable.

In future deployments, hybrid optimization frameworks may emerge, blending rule-based heuristics with AI predictions. For example, AI could be used to recommend execution plans or to flag when traditional decisions are likely to be suboptimal. Additionally, regular retraining on system-specific query logs would enable continuous improvement of AI models without sacrificing control.

CONCLUSION AND FUTURE WORK

This research compared how traditional and AI-driven methods optimise queries, using prepared data that mimics typical real-world queries. Despite good performance for one factor, traditional optimization was not able to adjust for the impacts of table and query size and computation cost. Tree-based classifiers in AI, CatBoost and LightGBM, performed much better than the other methods for accuracy and generalisation, with CatBoost leading in overall results. As a result, these models discovered more about the connexions between features, making their scan type predictions more precise than those of the previous fixed logic optimizers. Since AI methods have to be trained and can be hard to explain, their ability to adjust and keep learning make them attractive for use in modern databases, especially in dynamic large or cloud environments.

Further studies will investigate the use of these models for other query optimization issues and look at their performance using actual query logs. For production environments to be clear and reliable, more research on

explainable AI (XAI) techniques is necessary. Engineers are currently finding success by ensuring AI optimizers are used within existing DBMS engines, either as visible parts of the system or running silently.

REFERENCES

1. Basavegowda Ramu V, "Optimizing Database Performance: Strategies for Efficient Query Execution and Resource Utilization," International Journal of Computer Trends and Technology, 71, 15–21, 2023. Retrieved from https://doi.org/10.14445/22312803/IJCTT-V71I7P103.
2. Forresi, C., Francia, M., Gallinucci, E. *et al.* "Cost-based Optimization of Multistore Query Plans," *Inf Syst Front* **25**, 1925–1951, 2023. https://doi.org/10.1007/s10796-022-10320-2
3. Miryala, N. K, "Emerging Trends and Challenges in Modern Database Technologies: A Comprehensive Analysis," International Journal of Science and Research (IJSR), 13, 9, 2024. Retrieved from https://doi.org/10.21275/MS241126103744
4. Panwar, V, "AI-Driven Query Optimization: Revolutionizing Database Performance and Efficiency," Int. J. Comput. Trends Technol., vol. 72, no. 3, pp. 103-106, 2024. doi: 10.14445/22312803/IJCTT-V72I3P103.
5. Milicevic, B., & Babovic, Z, "A systematic review of deep learning applications in database query execution," J. Big Data, vol. 11, no. 1, 2024. doi: 10.1186/s40537-024-01025-1.
6. Alpha Friday, Ebole, & Iluno, C, "The Power of Relational Algebra in Database Management System," Scientific Research Journal, v. 14.
7. Paudel, N., & Bhatta, J, "Cost-Based Query Optimization in Centralized Relational Databases," Journal of Institute of Science and Technology, 24, 42–46, 2019. Retrieved from https://doi.org/10.3126/jist.v24i1.24627.
8. Peña, E., Almeida, E., & Naumann, F, "Fast Detection of Denial Constraint Violations," Proceedings of the VLDB Endowment, 15, 859–871, 2021. Retrieved from https://doi.org/10.14778/3503585.3503595.
9. Khachatryan, A., Müller, E., Böhm, K., & Kopper, J, "Efficient Selectivity Estimation by Histogram Construction Based on Subspace Clustering," In H. J. Schek, & G. H. Joubert (Eds.), Advances in Databases and Information Systems. Lecture Notes in Computer Science, vol. 6964, pp. 351–368. Springer, Heidelberg. Retrieved from https://doi.org/10.1007/978-3-642-22351-8_22.
10. Ordóñez, C., & García-García, J, "Evaluating Join Performance on Relational Database Systems," Journal of Computer Science and Engineering, 4, 276–290, 2010. Retrieved from https://doi.org/10.5626/JCSE.2010.4.4.276.
11. Alpha Friday, Ebole, & Iluno, C, "The Power of Relational Algebra in Database Management System," Scientific Research Journal, v. 14, 2017.
12. Mohseni, M., & Sohrabi, M, "Materialized View Selection in Data Warehouses Using Simulated Annealing," International Journal of Cooperative Information Systems, 29, 1–22, 2020. Retrieved from https://doi.org/10.1142/S021884302050001X.
13. Ramadan, M., El-Kilany, A., Mokhtar, H., & Sobh, I, "RL_QOptimizer: A Reinforcement Learning Based Query Optimizer," IEEE Access, 10, 70502–70515, 2022. Retrieved from https://doi.org/10.1109/ACCESS.2022.3187102.
14. Sun, Z., Wang, G., Li, P., Wang, H., Zhang, M., & Liang, X, "An Improved Random Forest Based on the Classification Accuracy and Correlation Measurement of Decision Trees," Expert Systems with Applications, 237, 2023. Retrieved from https://doi.org/10.1016/j.eswa.2023.121549.
15. Ali, Z., Abduljabbar, Z., Tahir, H., Sallow, A., & Almufti, S, "Exploring the Power of eXtreme Gradient Boosting Algorithm in Machine Learning: A Review," Academic Journal of Nawroz University, 12, 320–334, 2023. Retrieved from https://doi.org/10.25007/ajnu.v12n2a1612.
16. Yu, X, "Light Gradient Boosting Machine: An Efficient Soft Computing Model for Estimating Daily Reference Evapotranspiration with Local and External Meteorological Data," Agricultural Water Management, 225, 105758, 2019. Retrieved from https://doi.org/10.1016/j.agwat.2019.105758.
17. Hancock, J., & Khoshgoftaar, T, "CatBoost for Big Data: An Interdisciplinary Review," Journal of Big Data, 7, doi: 10.1186/s40537-020-00369-8.
18. Malik, A., Burney, S.M.A., & Ahmed, F, "A Comparative Study of Unstructured Data with SQL and NoSQL Database Management Systems," Journal of Computer and Communications, 8, 59–71, 2020. Retrieved from https://doi.org/10.4236/jcc.2020.84005.

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XII Issue VII July 2025

19. Sanmorino, A., Marnisah, L., & Sunardi, H, "Feature Selection Using Extra Trees Classifier for Research Productivity Framework in Indonesia," In F. Kalfane, & M. Yahia (Eds.), Data Analytics and Machine Learning for Internet of Things and Industrial Systems (pp. 21-44), 2023. Springer Singapore. Retrieved from https://doi.org/10.1007/978-981-99-0248-4_2.
20. Hargreaves, C., & Heng, E, "Simulation of Synthetic Diabetes Tabular Data Using Generative Adversarial Networks," Computational Biology and Chemistry, 7, 49–59, 2021.
21. Pradipta, G., Wardoyo, R., Musdholifah, A., Sanjaya, H., & Ismail, M, "SMOTE for Handling Imbalanced Data Problem: A Review," In Proceedings of the 2021 International Conference on Informatics and Computation (ICIC 2021), (pp. 1-8), 2021. Institute of Electrical and Electronics Engineers (IEEE). Retrieved from https://doi.org/10.1109/ICIC54025.2021.9632912.
22. Shouval, R., Bondi, O., Mishan, H., Shimoni, A., Unger, R., & Nagler, A, "Application of Machine Learning Algorithms for Clinical Predictive Modeling: A Data-Mining Approach in SCT," Bone Marrow Transplantation, 49(9), 1221–1230, 2013. Retrieved from https://doi.org/10.1038/bmt.2013.146.
23. Hooshyar, D, "Problems With SHAP and LIME in Interpretable AI for Education: A Comparative Study of Post-Hoc Explanations and Neural-Symbolic Rule Extraction," IEEE Access, PP, 1–1, 2024. Retrieved from https://doi.org/10.1109/ACCESS.2024.3463948.
24. Nannapaneni, S, "Machine Learning for Database Management Systems," International Journal of Engineering and Computer Science, 9, 25132–25147, 2020. Retrieved from https://doi.org/10.18535/ijecs/v9i08.4520.