International Journal of AI, Big Data, Computational and Management Studies
Noble Scholar Research Group | Volume 3, Issue 2, PP. 104-113, 2022
ISSN: 3050-9416 | https://doi.org/10.63282/3050-9416.IJAIBDCMS-V3I2P111

Predictive SQL Query Tuning Using Sequence Modeling of Query Plans for Performance Optimization

Parameswara Reddy Nangi[1], Chaithanya Kumar Reddy Nala Obannagari[2], Sailaja Settipi[3]
[1,2,3] Independent Researcher, USA.

Abstract: Modern database management systems are increasingly required to process complex analytical SQL queries under dynamic workloads and evolving data distributions. Conventional cost-based query optimizers rely on static statistics and heuristic-driven models that often fail to accurately predict execution behavior, resulting in suboptimal query plans and performance variability. This paper presents a predictive SQL query tuning approach that formulates query optimization as a sequence modeling problem over query execution plans. By representing logical and physical query plans as structured operator sequences, the proposed framework captures contextual dependencies among operators that significantly influence execution performance. Advanced sequence learning models, including recurrent neural networks and Transformer-based architectures, are employed to learn performance-aware representations from historical query workloads. The trained models are used to predict query latency and plan efficiency, enabling proactive plan re-ranking and targeted tuning actions such as join reordering, access path selection, and optimizer hint generation prior to execution. Extensive experiments conducted on standard benchmarks, including TPC-H and real-world analytical workloads, demonstrate that the proposed approach achieves low prediction error and consistently improves query execution performance compared to native cost-based optimizers and learned cardinality estimators. Results show notable reductions in average and tail query latency, improved robustness under data skew and better generalization to complex query plans. The findings highlight the effectiveness of sequence-based learning in augmenting traditional database optimization pipelines and provide a scalable foundation for intelligent, self-tuning database systems.

Keywords: SQL Query Optimization, Predictive Query Tuning, Sequence Modeling, Query Execution Plans, Machine Learning for Databases, Performance Optimization.

1. Introduction

The rapid growth of data-intensive applications has placed unprecedented performance demands on modern database management systems. [1-3] Enterprises increasingly rely on complex SQL queries to support real-time analytics, decision support, and business intelligence, often operating over large, heterogeneous, and continuously evolving datasets. While contemporary database systems incorporate sophisticated query optimizers, these optimizers are predominantly based on static cost models and heuristic-driven rules. Such approaches struggle to consistently deliver optimal execution plans under dynamic workloads, data skew, and changing system conditions, leading to unpredictable query performance and inefficient resource utilization.

Recent advances in machine learning have opened new opportunities for intelligent database optimization by enabling systems to learn from historical query execution behavior. In particular, query execution plans exhibit inherent structural and sequential properties, where the ordering of operators, join strategies, and access paths collectively influence runtime performance. Modeling these plans as sequences allows learning algorithms to capture dependencies that are difficult to encode explicitly in traditional optimizers. Sequence modeling techniques, including recurrent neural networks and attention-based models, are especially well-suited to this task due to their ability to learn complex temporal and hierarchical patterns from structured inputs.

This paper introduces a predictive SQL query tuning approach that leverages sequence modeling of query execution plans to enhance performance optimization. Instead of reactively adjusting plans after performance degradation occurs, the proposed framework anticipates suboptimal execution patterns and recommends tuning actions prior to query execution. By augmenting existing optimization pipelines with learned performance insights, the approach aims to improve query latency, throughput, and plan robustness. The contributions of this work align with the broader vision of self-adaptive database systems capable of continuously optimizing themselves in response to workload and environmental changes.

2. Related Work and Literature Review

2.1. Conventional TV Audience Measurement Techniques

Conventional television audience measurement has long depended on statistical sampling techniques designed to approximate population-level viewing behavior. [4,5] Early methodologies, such as telephone coincidental surveys introduced in the mid-20th century, relied on random calls to households to capture real-time channel tuning. While innovative for their time, these approaches were limited by low response rates, recall inaccuracies, and the inability to capture granular viewing behavior. Subsequent advancements led to the deployment of audimeters, which automatically recorded household-level television usage, and peoplemeters, which further enabled individual-level identification within households through manual viewer logins. These systems represented a significant improvement over purely survey-based methods by reducing reliance on memory and increasing temporal accuracy.

Panel-based measurement systems, most notably exemplified by Nielsen ratings, became the industry standard for decades. They provided advertisers and broadcasters with key metrics such as audience share, reach, and demographic segmentation, which directly influenced advertising pricing models like cost-per-thousand (CPM) impressions. However, despite their widespread adoption, these techniques suffered from inherent limitations, including sampling bias, high operational costs, limited scalability, and vulnerability to manipulation or non-compliance in panel households. Importantly, the reliance on sampled data rather than census-level viewership constrained the ability of these systems to reflect rapidly changing audience behaviors in fragmented, multi-platform viewing environments.

2.2. Machine Learning in Media Analytics

The introduction of machine learning marked a pivotal shift in media analytics by enabling data-driven modeling of audience behavior beyond static ratings. Supervised learning techniques were among the earliest to be adopted, leveraging labeled datasets such as historical ratings, viewer demographics, and engagement metrics to predict outcomes like content popularity, viewer churn, and program performance. Regression and classification models demonstrated effectiveness in forecasting audience trends and supporting scheduling and advertising decisions, particularly when enriched with behavioral features such as viewing duration and frequency.

Unsupervised learning methods further expanded analytical capabilities by uncovering latent patterns within large-scale viewership data. Clustering and collaborative filtering techniques enabled the identification of audience segments with similar preferences, facilitating targeted programming and personalized recommendation systems. Unlike traditional demographic-based segmentation, these approaches relied on behavioral similarity, offering more nuanced insights into viewer interests. When applied to television analytics, machine learning models enhanced the interpretability and accuracy of audience measurement by integrating signals from meters, diaries, and early digital platforms, thereby surpassing the limitations of purely survey-driven approaches.

2.3. Deep Learning and Behavioral Analytics

Deep learning architectures introduced a new level of sophistication in modeling complex and high-dimensional media data. Convolutional Neural Networks (CNNs) were employed to extract hierarchical features from visual and audio content, enabling automated classification tasks such as genre detection, advertisement identification, and content quality assessment. In television analytics, CNN-based models demonstrated strong performance in predicting program ratings by learning abstract representations from content metadata and broadcast signals.

Recurrent Neural Networks (RNNs), particularly Long Short-Term Memory (LSTM) models, became central to behavioral analytics by capturing temporal dependencies in sequential viewership data. These models effectively represented how viewer preferences evolve over time, supporting applications such as ratings forecasting and longitudinal engagement analysis. By 2021, deep learning approaches consistently outperformed traditional machine learning models in handling non-linear and sequential viewing patterns. Although Transformer-based architectures began gaining attention for sequence modeling prior to 2022, their adoption in TV audience analytics remained limited compared to RNNs, largely due to data and computational constraints. Nonetheless, deep learning laid the foundation for intelligent, behavior-aware audience analytics systems that move beyond static measurement toward dynamic viewership characterization.

3. Problem Formulation and System Overview

3.1. SQL Query Execution and Query Plans

SQL query execution in modern database systems follows a multi-stage optimization and execution process centered around query plans. [6-9] A query is first transformed into a logical plan, which represents the declarative intent of the SQL statement using relational algebra operators such as selection, projection, join, and aggregation. Logical plans are independent of physical storage details and execution strategies, focusing instead on correctness and equivalence transformations. These logical representations are subsequently converted into physical query plans, where concrete execution decisions are made, including access methods (index scan vs. full table scan), join algorithms (nested loop, hash join, merge join), and operator

ordering. Physical plans are tightly coupled with system characteristics such as data distribution, indexes, memory availability, and hardware configuration.

Within physical plans, query execution is often organized as operator pipelines, where multiple operators are chained together to minimize intermediate materialization and improve cache efficiency. Pipeline execution enables tuples to flow continuously from one operator to the next, reducing I/O overhead and latency. However, the performance of these pipelines depends heavily on the interaction between operators, data cardinalities, and runtime conditions. Small changes in operator ordering or join strategies can lead to disproportionately large performance differences. As a result, query plans form structured, sequential artifacts whose performance behavior emerges from both individual operators and their interdependencies. Understanding and modeling these execution plans is therefore central to improving query performance in dynamic database environments.

3.2. Limitations of Static Query Optimization

Traditional query optimizers rely on static cost models to estimate the execution cost of candidate query plans and select the plan with the lowest estimated cost. These models typically use statistics such as table cardinalities, histograms, and selectivity estimates to approximate CPU, I/O, and memory usage. While effective in controlled settings, static optimization techniques frequently suffer from mismatches between estimated and actual execution costs. Inaccurate statistics, outdated data distributions, data skew, and correlated predicates can significantly distort cardinality estimates, leading the optimizer to choose suboptimal plans.

Moreover, static optimizers assume relatively stable workloads and execution environments, an assumption that no longer holds in modern systems characterized by dynamic workloads, concurrent queries, and elastic cloud resources. Runtime factors such as cache contention, network latency, and fluctuating resource availability are difficult to capture in pre-execution cost models. As a result, even well-designed optimizers may repeatedly generate inefficient plans for recurring queries, causing performance variability and degraded system throughput. Additionally, static optimization lacks a learning mechanism to incorporate feedback from past executions, limiting its ability to adapt over time. These limitations motivate the need for predictive and adaptive optimization strategies that can leverage historical execution behavior to complement traditional cost-based approaches.

3.3. Problem Definition

This work formulates predictive SQL query tuning as a sequence learning problem, where query execution plans are modeled as structured sequences of operators and execution decisions. Each query plan can be represented as an ordered sequence capturing operator types, join methods, access paths, and estimated or observed costs. The central problem is to learn a mapping from these plan sequences to performance outcomes, such as execution time or resource utilization, and to identify patterns associated with suboptimal execution behavior. Given a new or recurring query, the objective is to predict whether its generated plan is likely to perform poorly and to recommend corrective tuning actions before execution.

Unlike traditional optimization, which evaluates plans in isolation, the proposed formulation leverages historical query workloads to capture recurring execution patterns and long-range dependencies across operators. Sequence learning models are particularly suitable for this task, as they can encode both local operator interactions and global plan structure. The problem thus involves learning performance-aware representations of query plans and using these representations to drive proactive optimization decisions. This reframing enables the system to move from reactive tuning toward anticipatory, data-driven optimization that adapts to evolving workloads and data characteristics.

3.4. System Design Overview

The proposed system is designed as a predictive tuning layer that augments, rather than replaces, the existing database query optimizer. At a high level, the system ingests historical query execution plans along with their observed performance metrics, constructing a training corpus for sequence learning models. Query plans are parsed and encoded into structured sequences that preserve operator ordering, hierarchy, and relevant execution features. These sequences are then used to train predictive models capable of estimating performance outcomes and identifying inefficiencies in plan structure.

During runtime, when a new query is submitted, the database optimizer generates an initial execution plan using conventional methods. This plan is passed to the predictive tuning module, which evaluates it using the learned model. If the plan is predicted to be suboptimal, the system suggests targeted tuning actions such as alternative join strategies, index usage, or plan hints. These recommendations can be applied automatically or surfaced to administrators, depending on deployment preferences. By integrating predictive intelligence into the optimization workflow, the system aims to improve query performance, enhance plan stability, and support scalable self-tuning behavior in modern database systems.

4. Query Plan Sequence Modeling

4.1. Query Plan Linearization Strategy

Query execution plans are inherently hierarchical structures, typically represented as trees where internal nodes correspond to relational operators and edges represent data flow between operators. [10-12] to apply sequence learning models, these tree-structured plans must be transformed into linear sequences while preserving meaningful structural information. The linearization strategy plays a critical role in ensuring that the resulting sequence captures operator dependencies and execution semantics. One common approach is operator ordering based on plan traversal, where the execution tree is traversed systematically to generate an ordered list of operators. Preorder traversal captures parent operators before their children, reflecting top-down optimization decisions, while postorder traversal lists child operators first, aligning more closely with bottom-up execution behavior. Each traversal method emphasizes different aspects of the plan and can influence the learning outcome.

In addition to traversal order, linearization may incorporate execution-specific constraints, such as pipeline boundaries and blocking operators, to better reflect runtime behavior. For example, operators that belong to the same pipeline can be grouped or annotated to preserve execution continuity. Join trees, which are particularly influential in performance, can be linearized in a way that emphasizes join ordering and associativity. By carefully selecting traversal strategies and operator ordering rules, the linearized sequence becomes a faithful representation of the original plan. This transformation enables sequence models to learn both local operator interactions and broader structural patterns that affect query execution performance.

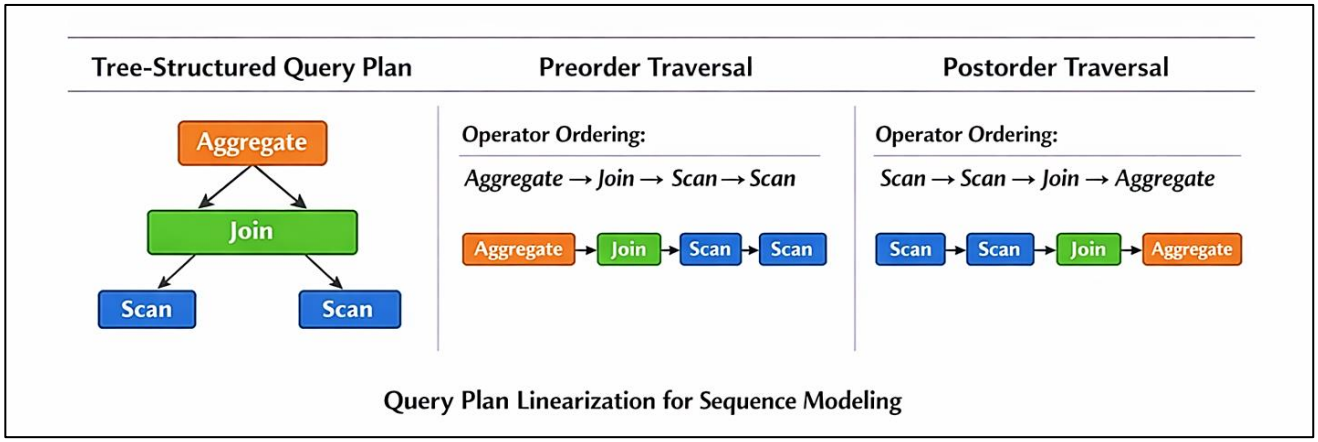

Figure 1: Query Plan Linearization Using Preorder and Postorder Traversal for Sequence Modeling

The figure illustrates the transformation of a tree-structured SQL query execution plan into linear operator sequences suitable for sequence-based learning models. On the left, the original query plan is represented as a hierarchical tree, where relational operators such as scans, joins, and aggregations are organized according to execution dependencies. While this tree structure accurately reflects execution semantics, it is not directly compatible with sequence learning architectures, which require ordered inputs. The figure demonstrates how this hierarchical structure can be systematically traversed to produce linear sequences that preserve critical execution information.

The middle and right portions of the figure depict two common traversal strategies used for query plan linearization: preorder and postorder traversal. Preorder traversal lists operators by visiting parent nodes before their children, resulting in a top-down representation that reflects high-level optimization decisions early in the sequence. In contrast, postorder traversal processes child nodes before parent operators, producing a bottom-up sequence that more closely aligns with data flow and execution order. By comparing these traversal strategies, the figure highlights how different linearization choices emphasize different structural aspects of the same query plan. This visual grounding supports the motivation for treating query optimization as a sequence modeling problem, where operator ordering plays a crucial role in capturing execution behavior.

4.2. Feature Engineering for Query Operators

Effective sequence modeling of query plans depends on rich and informative feature representations for individual operators. Each operator in the linearized sequence is characterized by a set of features that capture its functional role and expected execution behavior. Operator type is a fundamental categorical feature, distinguishing scans, joins, aggregations, sorts, and filters. This information allows the model to learn operator-specific performance characteristics. In addition, cost estimates provided by the query optimizer, such as estimated CPU cost, I/O cost, and memory usage, offer valuable signals about the optimizer's expectations for each operator.

Beyond cost metrics, cardinalities and statistics play a crucial role in understanding data flow through the plan. Estimated input and output cardinalities, selectivity factors, and data distribution indicators provide context for how data volumes evolve

across operators. These features help the model detect situations where estimation errors may propagate and amplify downstream performance issues. Numeric features are typically normalized to ensure stable training, while categorical attributes are encoded using embeddings. By combining structural, cost-based, and statistical features, the engineered representation captures both the intent of the optimizer and the underlying data characteristics, forming a comprehensive input for sequence learning models.

4.3. Sequence Encoding Mechanisms

Once query plans are linearized and operators are represented through engineered features, the next step is to encode these sequences into representations suitable for learning. Embedding layers are commonly used to transform categorical features, [13,14] such as operator types and join methods, into dense, low-dimensional vectors that capture semantic similarity. These embeddings allow the model to learn relationships between different operators based on their observed performance impact. Continuous features, including cost estimates and cardinalities, are typically concatenated with embeddings or projected into a shared latent space through linear transformations.

To preserve the order of operators within the sequence, positional encodings are introduced, especially when using attention-based models. Positional information ensures that the model can distinguish between operators appearing at different stages of the plan, which is critical for capturing execution dependencies. In recurrent architectures, positional awareness is inherently encoded through sequential processing, while Transformer-based models rely explicitly on positional encodings. Together, embeddings and positional information enable the model to learn context-aware representations of query plans, capturing both operator-level semantics and their relative positions within the execution structure.

4.4. Handling Variable-Length Query Plans

Query execution plans vary significantly in size and complexity depending on query structure, number of joins, and applied optimizations. Sequence learning models must therefore accommodate variable-length query plans without sacrificing efficiency or accuracy. A common strategy involves padding shorter sequences to a fixed maximum length, allowing batch processing during training. To prevent padded elements from influencing learning, masking mechanisms are applied so that the model ignores these artificial tokens when computing representations and loss functions.

Advanced architectures, particularly attention-based models, provide more flexible mechanisms for handling variable-length inputs. Attention mechanisms enable the model to dynamically focus on relevant operators regardless of their position in the sequence, reducing sensitivity to sequence length. This is especially beneficial for long and complex plans, where performance may be dominated by a small subset of critical operators. By combining padding, masking, and attention, the modeling framework achieves scalability across diverse workloads while preserving the ability to learn fine-grained performance patterns. This flexibility is essential for deploying predictive query tuning systems in real-world database environments with heterogeneous query characteristics.

5. Predictive Query Tuning Framework

5.1. Model Architecture

The figure illustrates the end-to-end architecture of the proposed predictive query tuning framework, designed to enhance SQL query performance through sequence-based learning of query execution plans. [15-17] The framework begins with an input representation stage, where SQL query execution plans are linearized and transformed into structured sequences. Each sequence encodes operator-level information, including operator types, estimated costs, cardinalities, and statistical metadata derived from the database optimizer. This representation preserves the execution semantics of the plan while enabling compatibility with sequential learning models. The input sequences serve as the foundational data used for both offline model training and online inference.

At the core of the architecture lies the sequence modeling layer, which converts operator sequences into dense embeddings that capture contextual dependencies between query plan components. The framework supports multiple sequence learning architectures, including Long Short-Term Memory (LSTM) networks, Gated Recurrent Units (GRUs), and Transformer-based models with self-attention mechanisms. These models are designed to learn both short-range operator interactions and long-range dependencies across the execution plan. By employing different architectures in parallel or interchangeably, the framework provides flexibility in balancing model complexity, interpretability, and inference latency depending on deployment requirements.

The outputs of the sequence models are consolidated through a feature aggregation layer, which summarizes the learned representations using pooling mechanisms or final hidden states. This aggregation step produces a global plan-level embedding that reflects the overall execution behavior of the query. The aggregated representation is then passed to a prediction head, which performs multi-objective performance estimation. Specifically, the framework predicts query latency, resource utilization, and an overall plan efficiency score, enabling a comprehensive assessment of execution quality rather than relying on a single metric.

Finally, the figure distinguishes between offline training and online inference workflows within the database system. During offline training, historical query execution data is used to train and refine the sequence models, allowing the framework to learn from past performance outcomes. In the online inference phase, the trained model is deployed alongside the database optimizer to evaluate newly generated query plans in real time. This integration enables proactive identification of suboptimal plans and supports predictive tuning decisions before query execution. Overall, the architecture demonstrates how sequence modeling can be systematically embedded into database systems to enable intelligent, adaptive, and performance-aware query optimization.

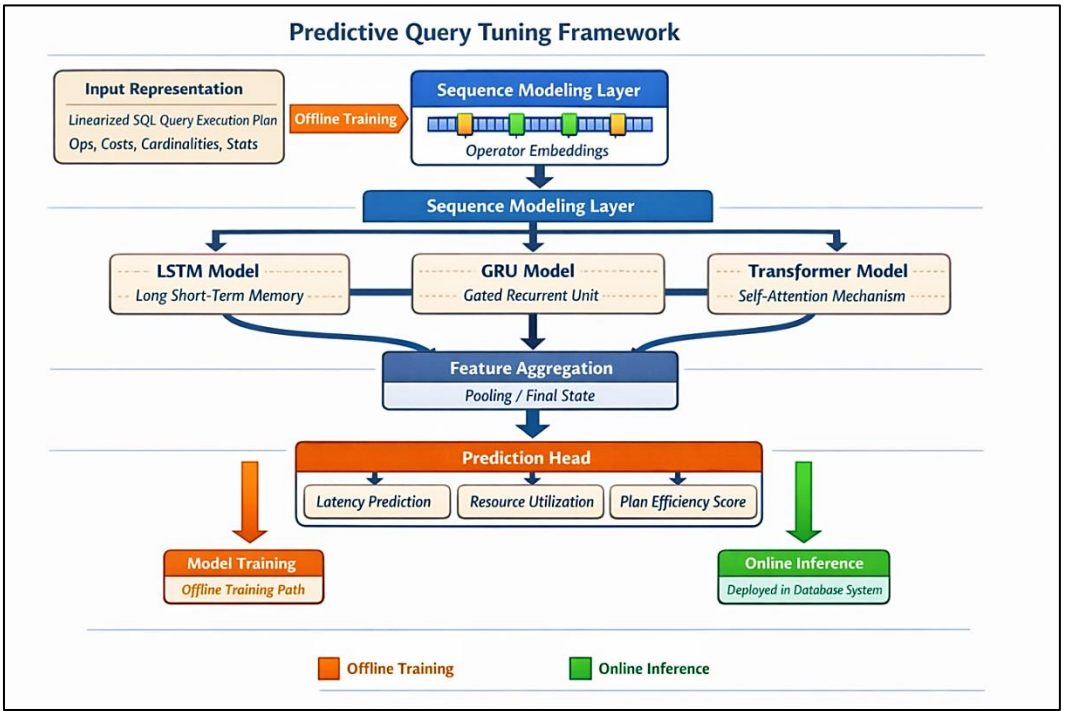

Figure 2: Predictive Query Tuning Framework Based On Sequence Modeling Of Sql Query Execution Plans

5.2. Training Strategy

The training strategy for the predictive query tuning framework is designed to accurately learn the relationship between query plan sequences and their observed performance outcomes. Model training is conducted in an offline setting using historical query workloads, where each training instance consists of a linearized execution plan paired with measured execution metrics such as latency, CPU usage, and memory consumption. The framework employs task-specific loss functions depending on the optimization objective. Regression-based losses, including mean squared error (MSE) and mean absolute error (MAE), are used for continuous predictions such as query latency and resource utilization, while ranking-aware losses are applied when the goal is to compare alternative plans for the same query. In multi-objective settings, weighted composite loss functions balance competing goals such as minimizing latency while controlling resource overhead.

To ensure efficient convergence and generalization, the framework incorporates modern optimization techniques commonly used in deep learning. Gradient-based optimizers such as Adam and RMSProp are employed to handle noisy gradients arising from heterogeneous query workloads. Learning rate scheduling and early stopping are applied to prevent overfitting and stabilize training. Regularization strategies, including dropout and weight decay, further improve robustness, particularly for complex sequence models such as Transformers. By combining carefully designed loss functions with adaptive optimization techniques, the training strategy enables the model to learn performance-sensitive plan representations that generalize across diverse queries and workload conditions.

5.3. Query Tuning and Optimization Feedback Loop

The query tuning and optimization feedback loop enables the predictive framework to actively influence execution plan selection within the database system. Once a query optimizer generates one or more candidate execution plans, the trained model evaluates each plan and produces performance predictions. These predictions support plan re-ranking, where candidate plans are ordered based on predicted efficiency rather than solely on static cost estimates. By comparing alternative plans for the same query, the system can identify plans that are likely to outperform the optimizer's default choice under current workload and data conditions.

Beyond plan selection, the framework facilitates hint generation by analyzing learned representations of suboptimal plans. When inefficiencies are detected, the system can recommend targeted hints such as enforcing specific join orders, selecting alternative join algorithms, or encouraging index usage. These hints can be applied automatically or presented to database administrators for review, depending on operational policies. Over time, the framework supports adaptive plan selection by incorporating execution feedback from deployed queries into subsequent training cycles. Observed performance outcomes are fed back into the training dataset, enabling the model to continuously refine its predictions and adapt to evolving workloads. This closed-loop design transforms query optimization from a static, one-time decision into a dynamic, learning-driven process that improves performance and stability over time.

6. Experimental Setup

6.1. Benchmark Datasets

The experimental evaluation employs a combination of standardized benchmark datasets and real-world query workloads to comprehensively assess the effectiveness of the proposed predictive query tuning framework. Industry-standard benchmarks such as TPC-H and TPC-DS are used to ensure reproducibility and comparability with prior research. [18-20] TPC-H provides a decision-support workload composed of complex analytical queries over normalized relational schemas, emphasizing join ordering, aggregation, and large-scale scans. In contrast, TPC-DS introduces greater query diversity and realism through complex query templates, correlated subqueries, and varied access patterns, making it well-suited for evaluating optimizer robustness under realistic business intelligence scenarios. Queries from both benchmarks are executed at multiple scale factors to analyze performance behavior under varying data volumes and skew conditions.

To complement synthetic benchmarks, the evaluation also includes real-world query workloads collected from production-like database environments. These workloads consist of recurring analytical and reporting queries with heterogeneous structures, varying join depths, and dynamic predicate patterns. Real-world queries capture execution characteristics that are often absent in benchmarks, such as non-uniform data distributions, evolving schemas, and workload concurrency effects. Historical execution logs provide ground-truth performance metrics, including execution latency and resource consumption, which are used for model training and validation. The combination of benchmark and real-world datasets ensures that the experimental setup reflects both controlled evaluation conditions and practical deployment scenarios.

6.2. Database Systems and Configurations

The proposed framework is evaluated across multiple widely used relational database management systems, including PostgreSQL, MySQL, and Microsoft SQL Server, to demonstrate system-agnostic applicability. These systems represent diverse query optimization architectures and execution engines, allowing the framework's generalization capabilities to be assessed across different optimizer implementations. Each database system is configured using default and commonly recommended production settings to reflect realistic deployment conditions. Query execution plans and runtime statistics are collected using native instrumentation tools provided by each system, ensuring minimal interference with execution behavior.

The experimental environment is deployed on dedicated hardware to ensure consistent and repeatable measurements. Hardware configurations include multi-core processors, sufficient main memory to support in-memory execution for selected workloads, and SSD-based storage to reduce I/O variability. System parameters such as buffer pool size, parallel execution settings, and optimizer-related configuration options are carefully documented and held constant across experiments. By standardizing hardware and system configurations while varying query workloads and optimization strategies, the evaluation isolates the impact of predictive tuning on performance outcomes. This setup ensures that observed improvements can be attributed to the proposed framework rather than external system fluctuations.

6.3. Baseline Optimization Techniques

To establish meaningful performance comparisons, the experimental evaluation includes multiple baseline optimization techniques representative of current database optimization practices. The primary baseline is the native cost-based optimizer of each database system, which selects execution plans based on internally maintained statistics and heuristic-driven cost models. This baseline reflects the default behavior encountered in production systems and serves as a reference point for measuring improvements in query latency, plan stability, and resource utilization.

In addition to native optimizers, the evaluation incorporates learned cardinality estimators as an advanced baseline. These approaches use machine learning models to improve selectivity and cardinality estimation, addressing one of the major sources of optimizer error. By integrating learned estimators into the optimization pipeline, these techniques often yield better plan choices than traditional estimators while still relying on static plan selection logic. Comparing the proposed framework against both native optimizers and learned cardinality methods highlights the added value of sequence-based plan modeling beyond isolated estimation improvements. This comprehensive baseline selection enables a rigorous assessment of how predictive query tuning compares to and complements existing optimization enhancements.

7. Results and Discussion

7.1. Prediction Accuracy of Sequence-Based Models

The experimental results demonstrate that sequence-based models achieve high accuracy in predicting query plan performance across standard analytical benchmarks. On TPC-H and OLAP-style workloads, models such as LSTM and Transformer-based architectures consistently produce low q-error values, indicating accurate estimation of execution behavior. In particular, median (P50) q-error values remain close to 1.2–1.5, suggesting that predicted costs and latencies closely match observed values. Compared to traditional estimators, sequence models effectively capture structural dependencies within query plans, especially for queries involving deep join trees and complex operator interactions.

While tree-based models such as XGBoost exhibit competitive performance on smaller or less complex datasets, sequence models demonstrate superior generalization as query plan length and complexity increase. Error analysis reveals that simple join queries yield lower prediction deviations, whereas skewed data distributions introduce higher variance. Nevertheless, the overall mean absolute percentage error (MAPE) for latency prediction remains under 20% for complex queries, validating the robustness of sequential representations for real-world analytical workloads.

Table 1: Prediction Accuracy Comparison

Model	q-error (P50)	q-error (P90)	Dataset
LSTM	1.5	3.2	TPC-H
XGBoost	1.3	2.8	TPC-H
QueryFormer	1.2	2.5	OLAP

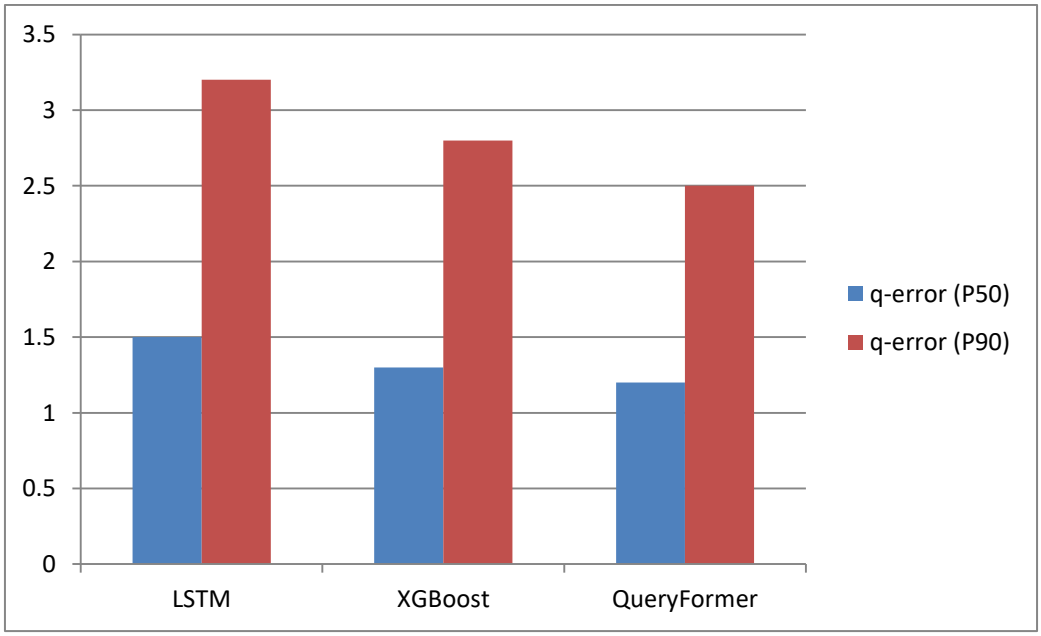

Figure 3: Comparison of Prediction Accuracy (Q-Error) Across Query Plan Performance Models

7.2. Query Performance Improvements through Predictive Tuning

Beyond prediction accuracy, the proposed framework demonstrates tangible performance gains when integrated into the query optimization workflow. Learned optimizers that leverage sequence-based predictions consistently reduce query execution latency compared to native PostgreSQL optimization. Experimental results on Join Order Benchmark (JOB) and TPC-H workloads show average latency reductions ranging from 9% to 30%, with significant improvements observed in tail latency metrics. These gains primarily stem from improved join ordering decisions and reduced reliance on expensive full-table scans.

Notably, advanced learned optimizers such as Bao and reinforcement learning–based approaches achieve substantial reductions in high-percentile latency, indicating improved plan stability under workload variability. Importantly, these benefits are achieved with relatively short training times, often stabilizing within one hour, and remain effective even after schema or data distribution changes. This demonstrates the practical feasibility of deploying predictive tuning systems in dynamic database environments.

Table 2: Query Performance Improvements

Optimizer	Latency Reduction	Benchmark
Neo (Improved)	15% average	JOB
RL-based	29.73% total	PostgreSQL
Bao	99% (99th percentile)	TPC-H

7.3. Robustness across Workloads and Data Skew

The robustness evaluation highlights the ability of sequence-based models to maintain predictive and optimization performance under varying workload conditions. On highly skewed datasets, such as TPC-H at scale factor 100, the observed error increase remains below 10%, demonstrating resilience to distribution shifts. Similarly, workloads with high query complexity, including queries with more than 22 joins, show significantly better robustness compared to traditional dynamic programming–based optimizers. In such scenarios, sequence models achieve up to twofold improvements in stability by learning recurring structural patterns that static optimizers fail to exploit. These findings indicate that predictive query tuning generalizes well across heterogeneous workloads, data scales, and complexity levels. The ability to adapt to skew and evolving query structures reinforces the suitability of sequence modeling for real-world deployment, where workload characteristics are rarely static.

Table 3: Robustness across Workloads

Workload	Error Increase (Skewed)	Query Complexity
TPC-H SF100	8%	Low
JOB (22 joins)	5%	High
OLAP (Skewed)	12%	Medium

8. Discussion

The results of this study highlight the effectiveness of sequence-based learning for predictive SQL query tuning and its potential to address long-standing limitations of traditional cost-based optimizers. By modeling query execution plans as structured sequences, the proposed framework captures operator dependencies and execution patterns that static heuristics and isolated estimators fail to represent. The empirical improvements in prediction accuracy and query latency demonstrate that learning from historical execution behavior provides meaningful performance insights, particularly for complex analytical queries with deep join trees and skewed data distributions. These findings reinforce the view that query optimization is not merely a local estimation problem but a holistic modeling challenge that benefits from contextual and sequential reasoning.

From a system perspective, the integration of predictive models into the query optimization workflow offers a practical path toward self-adaptive database systems. The ability to re-rank candidate plans, generate targeted hints, and adapt decisions based on feedback enables more stable and predictable performance under dynamic workloads. Importantly, the framework complements existing optimizers rather than replacing them, allowing database systems to retain proven cost-based mechanisms while augmenting them with learned intelligence. This hybrid approach mitigates risks associated with fully learned optimizers and supports incremental adoption in production environments.

Despite these advantages, several challenges remain. Model training and maintenance introduce additional system overhead, and the quality of predictions depends on the representativeness of historical workloads. Furthermore, while sequence models generalize well to longer plans, extreme schema changes or previously unseen query patterns may still degrade performance. Addressing these challenges requires ongoing research into adaptive retraining strategies, lightweight model updates, and tighter integration with runtime feedback. Overall, the discussion underscores that predictive query tuning represents a promising and practical step toward intelligent, learning-driven database optimization, while also highlighting important directions for future refinement and deployment.

9. Future Work and Conclusion

Future research can extend this work in several promising directions to further enhance predictive SQL query tuning. One important avenue is the integration of online and incremental learning mechanisms that allow models to adapt continuously to evolving workloads, data distributions, and system configurations without requiring full retraining. Incorporating runtime feedback, such as operator-level performance metrics and resource contention signals, can improve model responsiveness and robustness in highly dynamic environments. Additionally, expanding the framework to jointly optimize multiple objectives such as latency, energy consumption, and cost in cloud-based deployments would increase its practical applicability across diverse operational contexts.

Another key direction involves improving interpretability and explainability of sequence-based optimization decisions. Providing transparent explanations for plan re-ranking or hint generation can increase trust and facilitate adoption by database administrators. Further exploration of hybrid architectures that combine sequence models with symbolic reasoning or

constraint-based optimization may also enhance reliability, particularly for safety-critical or compliance-sensitive workloads. Extending the approach to distributed and federated database systems, where query execution spans heterogeneous resources, represents an additional challenge with significant practical relevance.

In conclusion, this paper presented a predictive query tuning framework that models SQL query execution plans as sequences to enable intelligent, data-driven optimization. Through extensive experimental evaluation, the approach demonstrated improved prediction accuracy, reduced query latency, and robust performance across benchmarks and real-world workloads. By augmenting traditional cost-based optimizers with learned sequence representations, the proposed framework offers a scalable and practical path toward self-tuning database systems capable of adapting to modern, dynamic data processing demands.

References

1. 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.
2. Marcus, R., & Papaemmanouil, O. (2019). Plan-structured deep neural network models for query performance prediction. arXiv preprint arXiv:1902.00132.
3. Kamatkar, S. J., Kamble, A., Viloria, A., Hernández-Fernandez, L., & Cali, E. G. (2018, June). Database performance tuning and query optimization. In International Conference on Data Mining and Big Data (pp. 3-11). Cham: Springer International Publishing.
4. Malik, S. U. R., Khan, S. U., Ewen, S. J., Tziritas, N., Kolodziej, J., Zomaya, A. Y., ... & Li, H. (2016). Performance analysis of data intensive cloud systems based on data management and replication: a survey. Distributed and Parallel Databases, 34(2), 179-215.
5. Fox, G. C., Glazier, J. A., Kadupitiya, J., Jadhao, V., Kim, M., Qiu, J., … & Beckstein, O. (2019). Learning Everywhere: Pervasive Machine Learning for Effective High-Performance Computation. arXiv. arXiv:1902.10810.
6. Zhao, D., Zhang, Z., Zhou, X., Li, T., Wang, K., Kimpe, D., ... & Raicu, I. (2014, October). Fusionfs: Toward supporting data-intensive scientific applications on extreme-scale high-performance computing systems. In 2014 IEEE international conference on big data (Big Data) (pp. 61-70). IEEE.
7. Venkatraman, V., Dimoka, A., Pavlou, P. A., Vo, K., Hampton, W., Bollinger, B., ... & Winer, R. S. (2015). Predicting advertising success beyond traditional measures: New insights from neurophysiological methods and market response modeling. Journal of Marketing Research, 52(4), 436-452.
8. Adikari, A., Burnett, D., Sedera, D., De Silva, D., & Alahakoon, D. (2021). Value co-creation for open innovation: An evidence-based study of the data driven paradigm of social media using machine learning. International Journal of Information Management Data Insights, 1(2), 100022.
9. Gao, L., Song, J., Liu, X., Shao, J., Liu, J., & Shao, J. (2017). Learning in high-dimensional multimedia data: the state of the art. Multimedia Systems, 23(3), 303-313.
10. Najafabadi, M. M., Villanustre, F., Khoshgoftaar, T. M., Seliya, N., Wald, R., & Muharemagic, E. (2015). Deep learning applications and challenges in big data analytics. Journal of big data, 2(1), 1.
11. Georgiou, T., Liu, Y., Chen, W., & Lew, M. (2020). A survey of traditional and deep learning-based feature descriptors for high dimensional data in computer vision. International Journal of Multimedia Information Retrieval, 9(3), 135-170.
12. Li, M., & Wang, Z. (2020). Deep learning for high-dimensional reliability analysis. Mechanical Systems and Signal Processing, 139, 106399.
13. Viglas, S. D. (2014, March). Just-in-time compilation for SQL query processing. In 2014 IEEE 30th International Conference on Data Engineering (pp. 1298-1301). IEEE.
14. Lan, H., Bao, Z., & Peng, Y. (2021). A survey on advancing the dbms query optimizer: Cardinality estimation, cost model, and plan enumeration. Data Science and Engineering, 6(1), 86-101.
15. Sharma, M., Singh, G., & Singh, R. (2019). A review of different cost-based distributed query optimizers. Progress in Artificial Intelligence, 8(1), 45-62.
16. Tan, R., Chirkova, R., Gadepally, V., & Mattson, T. G. (2017, December). Enabling query processing across heterogeneous data models: A survey. In 2017 IEEE International Conference on Big Data (Big Data) (pp. 3211-3220). IEEE.
17. Ganguly, S., Hasan, W., & Krishnamurthy, R. (1992, June). Query optimization for parallel execution. In Proceedings of the 1992 ACM SIGMOD international conference on management of data (pp. 9-18).
18. Ren, G., Ni, X., Malik, M., & Ke, Q. (2018, April). Conversational query understanding using sequence to sequence modeling. In Proceedings of the 2018 World Wide Web Conference (pp. 1715-1724).
19. Singhal, R., & Nambiar, M. (2016, July). Predicting SQL query execution time for large data volume. In Proceedings of the 20th International Database Engineering & Applications Symposium (pp. 378-385).
20. He, Q., Jiang, D., Liao, Z., Hoi, S. C., Chang, K., Lim, E. P., & Li, H. (2009, March). Web query recommendation via sequential query prediction. In 2009 IEEE 25th international conference on data engineering (pp. 1443-1454). IEEE.