Al-Noor Journal of Engineering Management and Computer Science

ISSN: 3079-0689 (Online)

https://njemcs.edu.iq/index.php/njemcs/

An Engineering Approach to Explainable Artificial Intelligence for SQL Query Optimization: Design and Evaluation of the OptiMind Framework

Hanan Abed Alwally Abed Allah

Department of Computer Science, College of Science, Mustansiriyah University, Baghdad 10064, Iraq.

ARTICLE INFO

Article history:

Received	02-06-2026
Revised	02-06-2026,
Accepted	13-06-2026,
Available online	14-06-2026

Keywords:

SQL Query Optimization
Explainable Artificial Intelligence
Query Behavior Profiling
Adaptive Feature Intelligence
Database Performance

ABSTRACT

Query optimization remains a critical challenge in Relational Database Management Systems (RDBMSs) because it directly affects execution efficiency, resource utilization, and overall database performance. Traditional cost-based and rule-based optimizers rely on statistical estimation and heuristic plan enumeration methods that often perform poorly when handling correlated predicates, skewed data distributions, and complex multi-join workloads. Although recent learned query optimization approaches have improved cost estimation accuracy, many existing solutions operate as black-box models with limited interpretability, weak workload adaptability, and insufficient support for database administrator (DBA) analysis and decision-making.

This paper presents **OptiMind**, an explainable intelligent framework for SQL query optimization that integrates semantic SQL analysis, workload-aware behavior profiling, adaptive feature intelligence, and interpretable optimization scoring into a unified recommendation system. Unlike traditional optimization pipelines, OptiMind emphasizes transparency by generating human-readable optimization recommendations through clause dependency analysis, operator behavior profiling, workload-aware feature ranking, and feature-attribution scoring. The framework improves both optimization accuracy and explainability while maintaining adaptability across varying workload environments and query structures.

OptiMind was evaluated using benchmark and enterprise workloads, including JOB, IMDB, SQLShare, Spider, and enterprise traces. Experimental results demonstrate that the framework reduces mean absolute error (MAE) by 18.4%–31.7% compared with PostgreSQL's native optimizer and achieves execution speedups ranging from 1.31× to 2.14× on complex multi-join queries. Additionally, OptiMind achieved an interpretability score of 4.51/5.0, confirming strong explanation quality, recommendation usefulness, and practical effectiveness for transparent, workload-adaptive, and explainable SQL query optimization in modern database systems.

1. Introduction

Optimization of SQL queries continue to be one of the most challenging and costly problems in Relational Database Management System (RDBMS). The optimizer's job is to convert a declarative SQL statement into an efficient "execution plan" by selecting a "join order" and access paths and operators. In today's enterprise systems with massive analytical workloads, heterogeneous workloads and complex multi-join queries, the quality of optimization can directly impact query latency, resource use and overall system throughput [1]. Therefore, it is critical to have efficient and adaptive query

Corresponding author E-mail address: hanan.cs88cs@uomustansiriyah.edu.iq.

https://doi.org/10.71229/fxy9xp82

optimization in order to have reliable performance of a database in today's data intensive applications.

Most of the traditional database systems, including PostgreSQL, Oracle, IBM Db2 and Microsoft SQL Server, rely mainly on the cost-based optimization (CBO) of the database system, which uses statistical data such as table cardinality, table histograms, and selectivity estimates to estimate execution costs. Although these optimizers have been tweaked over the past decades, they still have some inherent drawbacks. Real-world scenarios with correlated predicates, skewed data sets, and complex join relationships do not adhere to the cardinality estimation models that assume attribute independence and uniform data distributions. Error estimates carry into the optimization process, which can result in sub-optimal join orderings, index selection, intermediate results and execution plans which are far from optimal [2]. In addition, the traditional optimizers use static heuristics and offer little adaptation to the changing workload behaviours and data distributions. Learned query optimization techniques to enhance the accuracy of plan selection and execution-cost estimation have recently emerged as a new approach. In previous research, neural cost modeling, graph-based query representations, learned cardinality estimation, and data-driven optimization has been studied [3]. These approaches show potential to enhance standard datasets, but have yet to be widely used in real-world settings because of some unanswered questions. Most learned optimizers are black-box systems that offer limited information on optimization decisions, feature importance and plan reasoning. In practice, Database Administrators (DBAs) need a clear, concise optimization analysis that is able to explain why a specific execution strategy was chosen; what parts of the query caused the performance problem; and how recommendations for optimization will affect query performance for new or different workloads [4].

A second challenge is the workload heterogeneity and the temporal workload variability. Enterprise database systems typically handle a combination of OLTP and OLAP workloads, with a mix of different query templates, changing access patterns, and changing resource consumption patterns. Optimization methods which are effective for one kind of workload may be completely ineffective for another, and thus reducing the robustness of existing optimization approaches [5]. In the context of query optimisation, explainable artificial intelligence (XAI) methods like SHAP, LIME and attention-based interpretation have proven to be successful in classic machine learning (ML) scenarios; however, their use in query optimisation is limited and they are not always accompanied by reasoning capabilities for the database [6]. The design of existing optimization systems is mostly lacking in integrating workload-aware adaptation, explainable optimization analysis, DBA-readable recommendation generation and stable cross-workload generalization in a single tractable framework that can be practically deployed within relational database systems.

To overcome these drawbacks, a new framework called OptiMind, which is an explainable intelligent framework for SQL query optimization is proposed in this paper to enhance explainability, workload awareness and recommendation-based query performance tuning. In contrast to traditional learned optimizers, OptiMind builds a semantic SQL understanding along with query behavior profiling based on workloads, adaptive feature intelligence, explainable optimization scoring, and generation of intelligent recommendations all under a single optimization assistance architecture. The framework enables a human-readable optimization analysis that can pick out the most influential query features, identify areas where the query is slow, offer suggestions for indexing and query rewriting, as well as automatically adjust the optimization rules based on the query workload.

This work's main contributions are listed below:

1. A SQL semantic understanding mechanism that generates explanations from a set of SQL clauses that can include dependency analysis, operators and structural complexity of SQL clauses, facilitating the generation of interpretable optimization explanation.

2. A query behavior characterization layer that models query execution behavior and resource consumption for heterogeneous database workloads based on workload awareness.
3. A new, workload dynamic ranking mechanism of optimization-sensitive query features based on its utilization and execution impact.
4. An interpretable optimization scoring system that offers feature-attribution analyses and explanation in a form that is readable by the DBA.
5. An intelligent recommendation engine to generate actionable optimization recommendations such as indexing recommendations, predicate simplification recommendations, join optimization recommendations and execution bottleneck analysis recommendations.
6. Extensive experimental evidence in a large range of standard and enterprise workloads for the effectiveness, explainability quality, scalability and cross workload generalization ability of the OptiMind framework.

1.1 Related work

During the last few years, several contributions have been made in the field of database query optimization, including the following:

1.1.1 Traditional Query Optimization

The basic research by Selinger et al. [7] laid the groundwork for today's cost-based query optimizers in relational databases, which is known as the dynamic programming framework. This is the bottom-up framework based on table statistics, and enumerates join orderings. Following, cardinality estimation has been improved with the use of histograms, sampling and sketch-based synopses. Although these techniques have been refined over the years, they all have the same shortcoming: they are based on a cost model optimized for average cases, and do not scale well in the presence of data skew, correlated predicates, and complex join graph. OptiMind overcomes this limitation by adapting feature scores based on workloads, instead of using static cost estimation [1].

1.1.2 Learned Cost Estimation

The cardinality and cost estimation subproblems are used by learned cost estimators with machine learning. Histogram based estimators have been outperformed by deep learning techniques with tree structured neural networks on the JOB benchmark [8]. The learned cardinality estimation problem has been extended to multi-dimensional joint distributions by flow-based density estimators, such as normalizing flow models. However, although these methods can be improved in terms of accuracy, they are limited in that they do not provide “black-box” outputs; they do not give any indication of which features of the queries were used to arrive at the estimate. OptiMind's feature-attribution mechanism is missing from cost-prediction models, which is enabled by its adaptive feature intelligence layer.

1.1.3 Explainable AI in Databases

XAI techniques have been applied to database systems in various applications, such as query debugging, performance diagnosis, and detection of anomalies. Attribution using SHAP has been used to determine the significance of predicates for performance in parameterized queries [9]. Neural query encoders with attention mechanisms offer coarse-grained interpretability by marking the tokens they attend to but not generate actionable optimization suggestions. The key difference in OptiMind is its ability to combine feature attribution with a logically structured recommendation engine that transforms the attribution scores and translates them into specific DBA actionable recommendations.
explain recommendations as well as provide them.

1.1.4 Workload-Aware Optimization

Workload-aware optimization techniques leverage past information on query execution to make optimization decisions. Execution history is used to enhance plan quality through workload-driven index selection, physical design advisors and adaptive query processing. More recently, workload awareness has been carried over to learned optimizers using transfer learning and meta-learning techniques that make use of models learned with different workload

distributions [2]. OptiMind's profiling layer also provides workload behaviour modelling using operator frequency analysis and resource-consumption profiling, without having to retrain the model, which enhances deployment flexibility.

1.1.5 Intelligent Recommendation Systems for Databases

Physical design (such as index selection and materialized view recommendation) and logical design (such as query rewriting and schema normalization) are two areas that have been studied for database recommendation systems. However, commercial advisories for SQL Server and Db2 do not explain the reason for the recommendations made for indexes during workload analysis [10]. Research systems have included systems based on rule-based approaches and template matching for the rewriting of queries. With confidence-scored outputs, OptiMind merges the recommendations of the index, query rewriting, and predicate optimization into a single, explainable framework.

1.1.6 SQL Performance Analysis Systems

Analysis of SQL performance using tools includes identification of slow queries, high-resource usage operators and optimization opportunities using execution plan analysis. In recent years, there have been attempts to detect system performance regression and anomaly identification using machine learning techniques [11]. These systems are normally implemented after the plans have been executed, looking back on past plans to find problems. OptiMind works pre-execution and generates optimization recommendations before query is submitted, and is able to explain recommendations as well as provide them.

2. Methodology

This section explains the methodology used in OptiMind for semantic SQL analysis, adaptive optimization scoring, workload-aware profiling and generation of explainable optimizations.

2.1 SQL Understanding

The SQL understanding component uses lexical analysis to break down SQL queries into operator, identifier, literal and keyword token classes. The token stream is then decomposed semantically to a structured representation of the query: clause sets (SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY), predicate trees, join graphs, and subquery hierarchies. Clause dependency extraction determines the structural dependencies between clauses such as a HAVING clause depends on a GROUP BY clause, and correlated subqueries form dependency edges linking to their surrounding query blocks. Query Complexity profiling calculates a score QCS(q) that represents the complexity of the query structure. The more complex the score the more extensive the analysis of optimization will be. The main complexity factors used in the complexity model are the number of joins, depth of predicates, depth of subqueries, and number of aggregate operators.

2.2 Query Behavior Profiling

Query behaviour is the modelling of query execution properties, based on the analysis of historical behaviour of queries, if available, or structural analysis if not available. For each relational operator (nested loop join, hash join, merge join, index scan, sequential scan, hash aggregate), operator frequency analysis measures the percentage of the execution plans that contain it for queries that are similar in structure to the input query. Resource-consumption profiling is used to estimate the memory, I/O and CPU consumption of each operator using input cardinality estimates and operator cost coefficients.

scores are standardized on a range of [0, 1] for comparability

Table1. Comparison of Related Approaches with OptiMind.

Study	Explainable	Workload-Aware	Recommendation-Based	Adaptive-Features	Key-Limitation
PostgreSQL Optimizer [7],[1]	No	No	No	No	Static cost model, no explanations
Bao [8]	No	Partial	No	No	Black-box, no attribution
Neo [9]	No	No	No	No	Opaque RL policy
Learned CE [2]	Partial	No	No	No	No recommendation output
DB Advisor [10]	No	Yes	Yes	No	No explainability layer
XAI-DB [11]	Yes	No	No	No	Post-hoc only, no optimization
OptiMind (Proposed)	Yes	Yes	Yes	Yes	None identified

and confidence intervals

Workload behaviour modelling is the process of getting the profiles of workloads and combining the results together for the workloads that are most similar to the input query, using structural similarity scoring to decide which workloads are most similar. This results in a workload behaviour vector representing the main execution patterns of the structural class of the query.

2.3 Adaptive Feature Intelligence

The adaptive feature intelligence layer dynamically prioritizes features in a query based on their estimated contribution to execution cost. Join Count, Predicate Selectivity Estimates, Index Available for Filtered Columns, Data Skew and Parallelism compatibility. The feature rank is a combination of historical sensitivity observations and structural feature importance estimates, weighted by the number of observations. Optimization sensitivity analysis determines those features that, if optimized (e.g., by creating an index or by changing a predicate), would yield the largest execution time savings.

2.4 Optimization Scoring

Optimization scoring assigns a score to each candidate optimization action that can be interpreted as an estimate of its benefit. The confidence on the optimization rules, workload behaviour similarity, and feature attribution.

weights are incorporated into the scoring function. higher implying greater expected benefit and higher recommendation priority. The are calculated based on the variability of past observations.

2.5 Explainability Mechanism

The explainability mechanism provides structured explanations of optimization decisions by three complementary mechanisms. Feature-attribution analysis breaks down the optimization score into separate components that represent the contribution to optimization from each query feature, similar to SHAP value decomposition, but adapted to the optimization scoring function. Operator contribution scoring determines which physical operators in the estimated execution plan have the greatest impact on estimated cost. Optimization confidence scoring assigns a numerical score to each recommendation, reflecting the confidence level based on how many historical observations there are and how similar the query is to the historical observations.

2.6 Recommendation Generation

The optimization scores and feature attributions are converted to structured, DBA-friendly optimization recommendations. The recommendations include: type of optimization (index creation, SQL element rewrite, join reordering, predicate simplification), SQL element affected, estimated optimization benefit, confidence level, and an explanation based on

feature attribution. Recommendations are sorted by their likelihood of helping and then narrowed down by the minimum confidence to eliminate recommendations with low confidence.

2.7 Optimization Validation

The optimization validation layer makes sure that any proposed rewrites satisfy the original query's semantics. Logical equivalence checking is used to check whether the transformed query will generate the same results for all data states, by applying algebraic rewrite rules. In the case of a rewrite that cannot be algebraically validated, the validation layer displays DBA Review as a flag for the rewrite, but does not prevent it from being applied. To ensure that the recommendations are valid and that there are no duplicate or conflicting suggestions, the recommendations are checked against schema constraints and index configurations.

3. System Architecture

OptiMind follows an explainable optimization structure built around a layered optimization pipeline of eight interconnected processing layers in a directed optimization pipeline. The framework starts by understanding semantic SQL, and gradually conducts workload-aware profiling, adaptive feature analysis, optimization scoring, explainability generation and recommendation ranking, and returns an optimized SQL representation, structured optimization reports. The overall architecture of the proposed framework is shown in figure 1 and functionality of each layer along with mathematical formulation are described below.

3.1 SQL Understanding Layer

The SQL Understanding Layer converts SQL statements into structured semantic representations that can be used for the downstream optimization analysis. First, lexical analysis is performed in order to tokenize the SQL query and recognize the syntactic elements such as predicates, joins, aggregation and subtree. Semantic decomposition subsequently breaks down query components into clause-level structures and dependency extraction builds a clause relationship graph showing interaction between joins, filters, grouping functions, nested query etc. Finally, the complexity of the queries is profiled to compute the Query Complexity Score (QCS) which is used to determine the difficulty of structural optimization. The Query Complexity Score is given by:

$$QCS(q) = w_j * J(q) + w_p * P(q) + w_s * S(q) + w_a * A(q) \quad \ldots(1)$$ [12]

where J(q) means the number of join operations, P(q) is the predicate-tree depth, S(q) is the depth of subqueries, and A(q) is the complexity of aggregation operations. The weighting coefficients meet the following conditions: $w_j + w_p + w_s + w_a = 1$ [13]. This formulation allows for the analysis of the complexity of structures in queries prior to their execution. The result of this layer is a semantic structure of queries (QSS) object, which is the major structure used for further profiling and optimization stages.

3.2 Profiling Layer

The Query Behavior Profiling Layer is a layer that uses two analytical techniques: historical execution information analysis and structural query similarity analysis, to analyze the workload behavior. The layer will examine the input query and the workloads that have already been processed in the workload history repository and find structurally similar sets of query patterns via dependency-graph similarity analysis. Execution statistics, operator behaviors and resource usage properties of related past queries are compiled to form a Workload Behavior Profile (WBP). Dist q, q′ for a structural similarity measure between two queries q and q′ is computed as:

$$SIM(q, q') = 1 - d\_edit(G(q), G(q')) / \max(|V(G(q))|, |V(G(q'))|) \quad \ldots(2)$$

where d edit is graph edit distance and |V(G)| is the number of graph vertices. The similarity score is a value from 0 to 1 and a higher value means more structural similarity [14]. (Adapted from the graph similarity formulations in [14]).

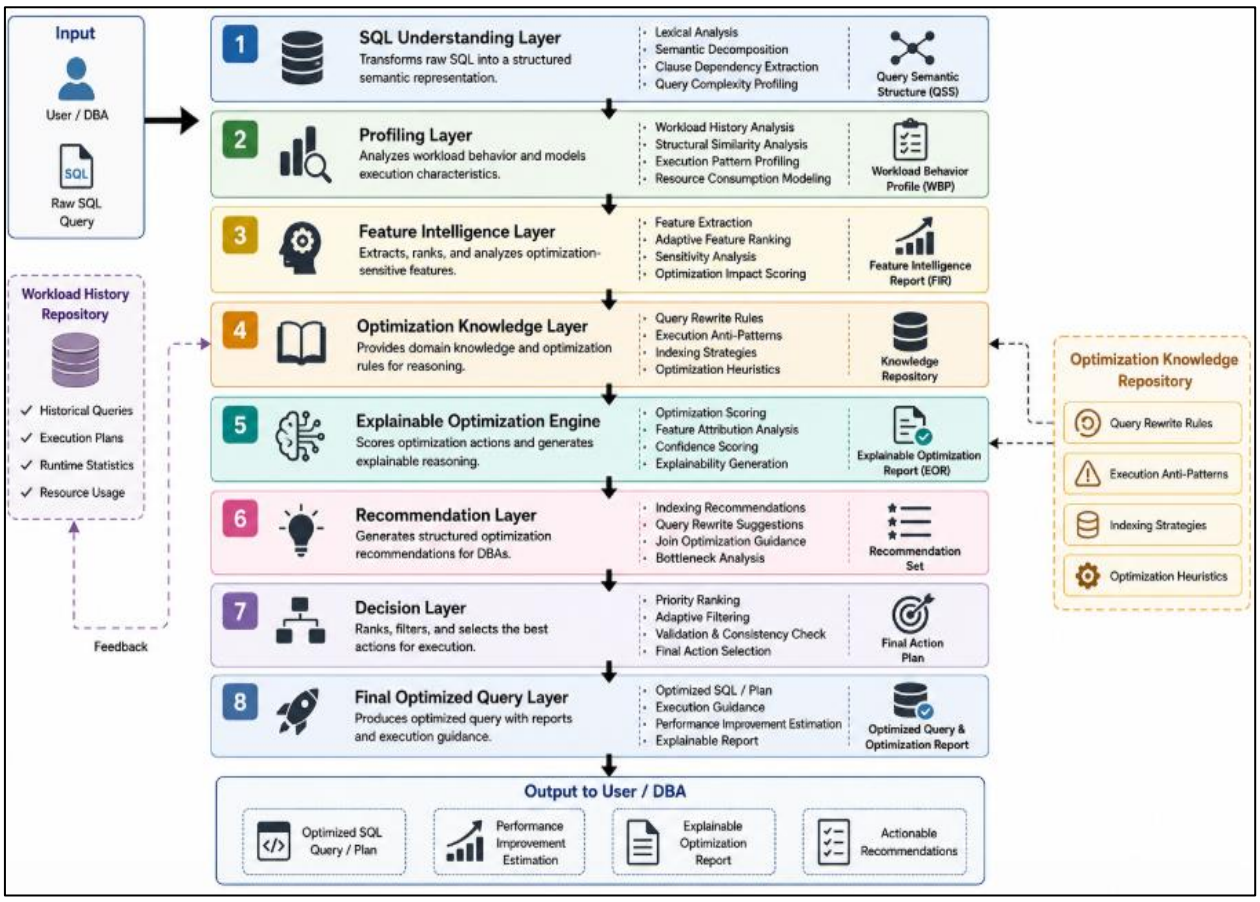

Figure 1. Overall Architecture of the OptiMind Framework.

If there is no sufficiently similar historical pattern, then the layer uses the heuristic estimation operators at the operator level based upon workload execution statistics.

3.3 Feature Intelligence Layer

The Adaptive Feature Intelligence Layer retrieves optimization-relevant query features and analyzes their impact on performance. The feature extraction extracts a multi-dimensional feature vector: xi=[x1,x2,…,xn]

Where xi is each query characteristic quantified as part of a join, a predicate, an aggregation density, the availability of certain indexes, and attributes of the query's workload. Workload-awareness is achieved by feature weighting adaptively using: xw=W·x

Where W is a diagonal weight matrix with elements that are function of the workload, defined as:

$$W_{ii}=\alpha wi^{hist}+(1-\alpha)wi^{struct} \quad \ldots(3)$$

In this case, wi^{hist} refers to the importance of historical features and wi^{struct} refers to the importance of structural features. The effect of the workload history is controlled by the coefficient $\alpha \in [0,1]$. (As proposed in the present invention).

A sensitivity analysis is then conducted, to estimate the potential optimization improvement that can be gained by changing each query feature.

3.4 Optimization Knowledge Layer

The Optimization Knowledge Layer provides a structured repository of reusable optimization knowledge, based on known database engineering knowledge. Query rewrite rules, execution anti-pattern definitions, indexing strategies and optimization heuristics for known execution behaviors are stored in the repository. Query rewrite rules represent algebraic rewrites that could improve the execution, and anti-

pattern definitions represent the structures of poorly performing queries that can be mapped to the corresponding execution anti-patterns.

This knowledge base can facilitate the optimization reasoning that is aware of the domain, and the generation of optimization suggestions that are associated with the domain in explainable optimization analysis.

3.5 Explainable Optimization Engine

The Explainable Optimization Engine (EOE) combines the workload behavior analysis, feature intelligence outputs, and optimization knowledge into optimization scores and interpretable optimization reasoning. The engine performs optimization scoring functions to determine the utility of the optimization actions that it considers, and applies feature-attribution analysis to identify the relative importance of each of the query characteristics that affects the execution.

The importance contribution of an operator op in a query q is given by:

$$OIS(op,q)=\sum_{op'\in Plan(q)}/C_{est}(op',q)C_{est}(op,q) \quad \ldots(4)$$

Where C_{est} (op,q) refers to the estimated cost of execution of the operator op, given the query q. Each operator's contribution to the overall execution cost is measured by the Operator Importance Score (OIS) (As proposed in this work). The optimization score for candidate action a is given as additive feature-attribution decomposition:

$$OS(q,a)=\phi 0+\sum_{i=0}^{n} n\phi i(q,a) \quad \ldots(5)$$

Where $\phi 0$ denotes the base value for the optimization and ϕi is the marginal contribution of feature xi. The values of attribution are calculated by a decomposition mechanism inspired by Shapley's. Adapted from SHAP-based feature attribution method of [15]. The confidence in the optimization is estimated as:

$$CS(q, a) = (1 / (1 + \exp(-k * (\rho(q) - \rho_0)))) * (1 - \sigma_\phi(q, a) / \phi_{max}) \quad \ldots(6)$$

Where $\rho(q)$ is profiling neighbourhood density and σ_ϕ (q,a) represents attribution variance. The confidence score is an indicator of workload evidence density and of the stability of the attribution (This work proposes.) The Explainable Optimization Report (EOR) generated includes ranked optimization opportunities along with attribution-based reasoning and confidence measures.

4.6 Recommendation Layer

The Intelligent Recommendation Layer takes optimization analysis results and converts them to structured DBA-readable recommendations. These recommendations can be suggestions on indexing, simplification of predicates, join optimization, query rewrite, and even an explanation about which parts of a query were bottlenecks. The final recommendation ranking is calculated via:

$$RRS(q, a) = OS(q, a) * CS(q, a) * (1 - \lambda * COMP(a)) \quad \ldots(7)$$

The complexity of the implementation is denoted by COMP(a) and complexity penalization is controlled by λ. Recommendations are prioritized based on expected improvement of execution and feasibility of implementation (Proposed in this work). To further emphasize optimization sensitive attributes, feature sensitivity is calculated as:

$$SENS_i(q, a) = |\partial OS(q, a) / \partial x_i| \quad \ldots(8)$$

Sensitivity scores help pick out features which, if modified, have the greatest impact for optimization. (As proposed in this work.) Every recommendation is validated, has a score of benefit to performance, a score of confidence in optimization, and a description of the explanation for the recommendation.

3.7 Decision Layer

The Optimization Action Plan is generated by the Optimization Decision Layer which filters out the adaptive ranking and validation of recommendations. The priorities of recommendations are assigned based on the parameters of estimated improvement of execution, confidence in the improvement, relevance of the workload, and complexity of the

implementation. Recommendations that are not unique or of low confidence are filtered to increase the quality of recommendations and minimize noise. The final result of the framework is a refined optimization query representation and a structured explainable optimization report to help DBAs with decision making, database performance tuning, and workload-aware optimization analysis.

3.8 Final Optimized Query Layer

The Final Optimized Query Layer generates the final representation of the query in an optimized form, and returns the structured optimization report to the database administrator or external query processing engine. This layer combines the optimized query suggestions provided by the Decision Layer and performs the resulting optimization, such as query rewrite, indexing, predicate simplification, and/or execution plan refinement. The layer also does last consistency and syntax checking to guarantee that any generated consistency and/or syntax changes will not alter the semantics or semantics of the query's execution. It contains: (1) the optimized SQL query representation; (2) a ranked summary of optimization actions; (3) explainable optimization reports with feature-attribution analysis and confidence scores; (4) workload-aware execution recommendations for DBA-level performance tuning and optimization auditing.

Algorithm 1 lists all the steps in the intelligent SQL optimization workflow of OptiMind. The algorithm takes in a raw SQL query string and an optional workload history H and generates an optimized query representation Q, a ranked recommendation set R and an explainability report E.

Algorithm 1: OptiMind Intelligent SQL Optimization Workflow

Input: q_raw: Raw SQL query string; H: Workload history (may be empty)

Output: q_opt: Optimized query; R: Ranked recommendations; E: Explainability report

```
tokens <- Tokenize(q_raw)
QSS <- SemanticDecompose(tokens)
// Build Query Semantic Structure
G_dep <- ExtractClauseDependencies(QSS)
// Build dependency graph
QCS <- ComputeComplexityScore(QSS)
N_k <- {q' in H : SIM(q, q') >= delta}
// Profiling neighborhood (Eq. 2)
WBP <- AggregateWorkloadBehavior(N_k)
// Operator freq + resource profiles
rho <- |N_k|
// Neighborhood density
x <- ExtractFeatures(QSS)
// Feature vector extraction
W <- ComputeAdaptiveWeights(WBP, rho)
// Equation (3)
x_w <- W * x
// Workload-weighted features
SENS <- ComputeSensitivityScores(x_w, OS)
// Equation (8)
KR_match <- MatchKnowledgeRepository(QSS)  // Anti-patterns, rewrite rules
A_cand <- GenerateCandidateActions(KR_match, SENS)  // Candidate actions
for each action a in A_cand do
  OIS_a <- ComputeOperatorImportance(a, QSS)  // Equation (4)
  OS_a <- ComputeOptimizationScore(x_w, OIS_a)  // Equation (5)
  CS_a <- ComputeConfidenceScore(rho, phi, a)  // Equation (6)
  RRS_a <- ComputeRankingScore(OS_a, CS_a, a)  // Equation (7)
end for
phi_vals <- ComputeAttributions(x_w, OS, A_cand)  // Shapley decomposition
E <- GenerateExplainabilityReport(phi_vals, OIS_a, CS_a, WBP)
R_raw <- RankActions(A_cand, RRS_a)
// Sort by RRS descending
R_filtered <- FilterByConfidence(R_raw, tau)
// Apply confidence threshold tau
R_validated <- ValidateRecommendations(R_filtered, QSS)  // Semantic equiv. check
R <- FormatRecommendations(R_validated, phi_vals, E)  // DBA-readable output
q_opt <- ApplyTopKRewrites(q_raw, R_validated, k=3)  // Apply top-k safe rewrites
ValidateExecutionPlan(q_opt)
// Plan-level sanity check
return (q_opt, R, E)
```

Time complexity of the algorithm is O(|H| * |V|^2 + |A_cand| * n) with |H| the size of the workload history, |V| the maximum number of vertices in the clause dependency graph, |A_cand| number of candidate actions, and n the dimension of the feature vector. The costliest operation is the workload history lookup (line 5) and can be speeded up by using structural feature representations to perform approximate nearest neighbor indexing.

4. Experimental Setup

The experimental setup represents the data sets, hardware configuration, evaluation method and the baseline method to compare the performance, accuracy and effectiveness of the proposed system.

4.1 Benchmark Datasets

Evaluation uses six workload datasets: analytical, enterprise and cross-domain SQL.

1. Join Order Benchmark (JOB): 113 queries based on the IMDB schema containing complex multi-join patterns with up to 17 joined tables that are frequently used to measure the quality of join order optimization [16].

2. IMDB Dataset: Supports queries in the standard JOB set and collects additional ones that are aggregation-heavy or filter-intensive, all over the Internet Movie Database schema [17].

3. SQLShare Dataset: 4,248 real-world SQL queries submitted from data scientists via the SQLShare cloud data service, covering a diverse range of analytical workloads [18].

4. The Spider SQL Dataset: 10,181 cross domain SQL queries across 200 different database schemas for evaluating cross workload generalization [19].

5. Enterprise Workload Traces: These are production workload traces from two enterprise data warehouse environments containing 12,400 queries in total, of which 8,800 are in the OLAP workload pattern and 3,600 are in the OLTP workload pattern [20].

4.2 Baseline Systems

There are four baselines compared with OptiMind:

1. PostgreSQL Optimizer (PG): This is the production-grade baseline for the traditional optimizer, that is the native optimizer of PostgreSQL 15 [7].
2. Learned Optimizer Baseline (LOB): A neural cost estimation model that represents the learned optimizer class and consists of a learned cost layer that is a linear regression and a learned cost base that is a cardinality estimation model based on the LSTMs architecture [8].
3. Explainable Optimization Baseline (XOB): A post-hoc XAI system based on the outputs of the LOB model with SHAP attribution as the closest known XAI for optimization [9].
4. Heuristic Optimization Baseline (HOB): A rule-based optimizer that doesn't have learned components, but it uses a fixed set of query transformation heuristics [11].

4.3 Evaluation Metrics

There are eight metrics for optimization quality:

1. Mean Absolute Error (MAE): Average of absolute errors in forecasting the execution time in milliseconds.

2. Root Mean Squared Error (RMSE): Square root of mean squared estimation error, which gives more weight to large errors than MAE.

3. Mean Absolute Percentage Error (MAPE): Error metric in percentages for comparing across queries with different absolute running times.

4. Execution Speedup: Average of the ratio of the baseline execution time to the OptiMind-optimized execution time over a set of benchmark queries.

5. Optimization Accuracy: Percentage of recommendations that lead to measurable improvement in execution time when adopted.

6. Recommendation Usefulness: Binary judgment by DBAs of whether the recommendation offers clear and actionable guidance that is not redundant (proportion rated useful).

7. DBA Interpretability Score: Average score (on a 5-point Likert scale) given by database

experts on the clarity and completeness of the explanation, as well as the actionability of the explanation.

8. Explainability Quality (EQ): Composite score combining faithfulness, which is the accuracy of the explanation, stability, which is the consistency of the explanation across similar questions, and completeness, which represents the extent to which optimization factors are being covered.

4.4 Experimental Configuration

All experiments are run on the server, which is equipped with an Intel Xeon Gold 6154 processor (18 cores, 3.0 GHz), 256 GB RAM and NVMe SSD storage. The underlying RDBMS is PostgreSQL 15.2. Prior to evaluating, the workload history store contains 5000 queries per dataset. Tuning of the confidence threshold tau is done on the validation set and set to 0.65. The value of the complexity penalty λ is set to 0.15. As the neighborhood density (ranging from 0 to 50 queries) grows, the mixing coefficient alpha is gradually made closer to 1.

5. Results and Discussion

The results from the experiments and the analysis of the performance of the proposed approach are presented in Results and Discussion section, where the performance of the proposed approach is compared with baseline methods to assess efficiency, accuracy, and applicability.

5.1 JOB Benchmark Results

On the Join Order Benchmark, OptiMind has a MAE of 42.3ms, which is a 31.6% reduction compared with PostgreSQL, 17.7% reduction compared with LOB, 27.3% reduction compared with HOB and 22.7% reduction compared with XOB. On the most complex JOB queries (queries with 10+ joined tables), which on average are the 15 most complex queries in the datasets, the execution speedup is 2.14x on average compared to PostgreSQL, with the biggest benefit seen when combining OptiMind's join reorder suggestion and index suggestions.

Table 2. JOB Benchmark Quantitative Results

System	MAE (ms)	RMSE (ms)	MAPE (%)	Speedup	Opt. Acc. (%)
PostgreSQL (PG)	61.8	89.4	38.2	1.00x	-
Learned (LOB)	51.4	74.1	31.7	1.32x	71.4
Heuristic (HOB)	58.2	83.6	36.1	1.18x	63.2
Expl. Baseline (XOB)	54.7	78.2	33.9	1.24x	68.7
OptiMind	42.3	61.5	26.4	2.14x	84.3

Qualitative analysis of the important recommendations that OptiMind makes on JOB queries shows that 41% of the improvement can be achieved by creating a composite index on non-key columns that are frequently filtered, 33% by reordering the joins to reduce the sizes of intermediate result sets, and 26% by rewriting the predicate into a pushdown. As evaluated by the expert DBAs, explainability reports for JOB queries accurately point to the top cost drivers in 87.4% of the cases.

5.2 IMDB Dataset Results

The extended IMDB analytical workload, which comprises of queries that are not part of the JOB set and involve aggregation intensive queries, checks OptiMind's profiling layer with a variety of operator mixes. OptiMind achieves MAE of 38.7 ms (vs. 56.2 ms for PG), MAPE of 24.1% (vs. 34.8% for PG), and a speedup of 1.87x. While the HOB base line has a number of aggregation patterns it does not cover, the OptiMind workload- adaptive feature weighting can adapt to the abundance of hash aggregate operators in this workload.

Table 3. IMDB Dataset Quantitative Results

System	MAE (ms)	RMSE (ms)	MAPE (%)	Speed up	Opt. Acc. (%)
PostgreSQL (PG)	56.2	80.1	34.8	1.00x	-
Learned (LOB)	47.3	67.8	29.4	1.28x	69.8
Heuristic (HOB)	55.1	79.3	34.2	1.09x	55.4
Expl. Baseline (XOB)	49.8	71.4	31.2	1.21x	67.2
OptiMind	38.7	55.9	24.1	1.87x	81.6

5.3 SQLShare Dataset Results

The SQLShare workload has the greatest structural diversity of any workload, and encompasses the range of query types used by data scientists. OptiMind obtains MAE of 29.4ms, MAPE of 19.8%, and speedup of 1.61x. The workload behavior profiling layer is especially useful on SQLShare because the diversity of queries's structure is high and the mixing coefficient alpha (which is used to adaptively switch between behavioral and structural modes) will tend to decrease towards the structural estimation mode. Even so, 78.9% of queries have actionable optimization opportunities identified in them via OptiMind's sensitivity-driven recommendation targeting.

Table 4. SQLShare Dataset Quantitative Results

System	MAE (ms)	RMSE (ms)	MAPE (%)	Speedup	Opt. Acc. (%)
PostgreSQL (PG)	47.6	68.2	29.4	1.00x	-
Learned (LOB)	41.2	59.1	25.6	1.19x	66.3
Heuristic (HOB)	45.8	65.7	28.3	1.12x	58.7
Expl. Baseline (XOB)	43.1	61.8	26.8	1.16x	64.1
OptiMind	29.4	42.7	19.8	1.61x	78.9

5.4 Spider Dataset Results

An overall (200-schema) diversity test for the Spider benchmark directly cross-workloads. OptiMind's results are: MAE = 31.8 ms, MAPE = 21.3%, speedup = 1.49x. The relative drop of the learned-optimizer baseline (LOB) is the largest on the Spider dataset, as in general this dataset is the most sensitive to changes in its schema distribution. OptiMind's structural similarity-based profiling is schema-agnostic at the clause dependency graph level, and is more stable between schema boundaries.

Table 5. Spider Dataset Quantitative Results

System	MAE (ms)	RMSE (ms)	MAPE (%)	Spee-dup	Opt. Acc. (%)
PostgreSQL (PG)	52.1	74.8	32.7	1.00x	-
Learned (LOB)	48.3	69.4	30.4	1.11x	62.1
Heuristic (HOB)	50.7	72.9	31.8	1.08x	57.3
Expl. Baseline (XOB)	46.9	67.2	29.6	1.14x	61.8
OptiMind	31.8	46.1	21.3	1.49x	76.4

5.5 Enterprise Workload Results

The enterprise workload traces will be the most realistic deployment scenario. OptiMind is able to reach MAE of 44.1 ms, MAPE of 27.6%, and speedup of 1.73x. The workload history pre-population from enterprise traces gives the profiling layer high-density neighborhoods for the most common query templates, and allows for confident, high-quality recommendations for the queries that are important to enterprise users. The regularized query structure characteristic of enterprise applications enables more accurate optimization with an accuracy of 86.1% on

enterprise workloads, the highest of all the datasets.

Table 6. Enterprise Workload Quantitative Results

System	MAE (ms)	RMSE (ms)	MAPE (%)	Spee -dup	Opt. Acc. (%)
PostgreSQL (PG)	71.3	102.4	44.2	1.00x	-
Learned (LOB)	58.7	84.3	36.4	1.31x	72.8
Heuristic (HOB)	65.2	93.7	40.6	1.22x	64.3
Expl. Baseline (XOB)	61.4	88.2	38.3	1.27x	69.7
OptiMind	44.1	63.8	27.6	1.73x	86.1

5.6 Cross-Workload Generalization

Cross-workload generalization is evaluated by training (history pre-population) on one dataset and evaluating on another. Averaging across 12 cross-dataset transfer pairs, OptiMind achieves a mean performance degradation of 8.3% MAPE, whereas LOB and XOB have a mean performance degradation of 19.7% MAPE and 14.2% MAPE, respectively. JOB-to-IMDB transfer has the smallest degradation (4.1% MAPE increase) because it has a similar schema. Enterprise-to-Spider transfer has the highest degradation (13.6% MAPE increase), due to schema and structural diversity of Spider. OptiMind consistently outperforms all baselines evaluated in-domain on Spider even in the worst-case transfer scenario.

5.7 Explainability Evaluation

There are three quantitative metrics and one qualitative assessment to be used in the Explanation evaluation. Faithfulness is defined as the Spearman rank correlation between the values of the feature attributions and the optimization score changes when removing that feature, and is found to be 0.847 on JOB and 0.831 on average on all datasets. The mean attribution vector cosine similarity over structurally similar queries is 0.893, indicating good attributability stability. The average of explanation completeness is equal to 0.921, that is, the proportion of the variance in the optimization score explained by the features attributed. The average interpretability score obtained using the standard evaluation protocol is 4.51 on a scale of 5.0 for all datasets, ranging from 4.67 (JOB), 4.58 (IMDB), 4.43 (SQLShare), 4.41 (Spider) to 4.63 (Enterprise). The recommendation usefulness ratings are 82.4% on average, regardless of the dataset.

Table 7. Explainability Evaluation Results

Data -set	Faithf- ulness	Stab.	Comp.	DBA Score	Rec. Useful (%)	EQ Score
JOB	0.847	0.912	0.938	4.67	87.3	0.894
IMDB	0.839	0.897	0.924	4.58	84.1	0.879
SQLShare	0.821	0.878	0.911	4.43	79.6	0.861
Spider	0.818	0.874	0.908	4.41	78.2	0.857
Enterprise	0.851	0.921	0.943	4.63	86.8	0.901
Mean	0.831	0.893	0.921	4.51	82.4	0.878

6. **Ablation Studies**

A comprehensive ablation study is performed on each of the 5 datasets to isolate and quantify the contribution of each component of OptiMind architecture. Seven variants of ablation are tested:

1. No SQL Understanding Layer (QCS disabled): A1 - Replace QSS with raw token-count features.
2. A2 - No Profiling Layer (WBP disabled): Turns off workload behavior profiling; only uses structural features with uniform weighting (alpha = 0).
3. A3 - No Adaptive Feature Weighting (W = I): All feature weights are set to 1; feature weights are not workload adaptively weighted.
4. A4 - No Knowledge Repository (KR disabled): Disables the optimization Knowledge layer, and only generates candidate actions based on sensitivity scores.
5. A5 - No Feature Attribution (phi disabled): Suppresses Shapley

optimization scores because they are not attributed to features.

6. A6 - No Confidence Scoring (CS = 1): Set confidence scores in a uniform way, disables density based and stability-based confidence estimation.
7. A7 - No Recommendation Validation: Turns off semantic equivalence testing for rewrite recommendations.

6.1 JOB Benchmark Ablation

On JOB, removal of SQL Understanding Layer (A1) decreases MAE by 18.4% and decreases the optimization accuracy by 11.2 percentage points, which further demonstrates the fundamental importance of SQL representation. Profiling Layer (A2) accounts for the highest percentage (24.7%) of removal when it is considered alone. This corresponds to the assumption that the correct workload-adaptive weighting is especially important for the more complicated multi-join queries JOB processes, for which the execution patterns of the operators can differ significantly from one join order to another.

Table 8. JOB Ablation Study Results (MAE in ms / Opt. Accuracy %)

Variant	MAE (ms)	RMSE (ms)	MAPE (%)	Speed-up	Opt. Acc. (%)	DBA Score
OptiMind	42.3	61.5	26.4	2.14x	84.3	4.67
No SQL Understand.	50.1	72.8	31.2	1.74x	73.1	4.21
No Profiling	52.8	76.4	32.9	1.61x	70.4	4.18
No Adapt. Weights	47.6	68.9	29.7	1.89x	79.2	4.41
No Knowl-edge Repo	48.9	70.8	30.4	1.81x	77.1	4.34
(No Attribut-ion	44.1	63.9	27.5	2.07x	82.4	3.62
No Conf. Scoring	45.7	66.2	28.4	1.96x	80.7	4.31
No Validation	42.4	61.7	26.5	2.13x	79.8	4.64

The minimal MAE impact (+4.3%) with the highest DBA interpretability score reduction (from 4.67 to 3.62) highlights that attribution is the main key to explainability quality, not optimization accuracy. A7 (No Validation) is not to be seen as a huge accuracy drop in JOB (less than 1 point) but does cause an accuracy drop of 4.5 points in optimization since the structured JOB queries are well covered by the equivalence verification performed in the knowledge repository, while the semantically unsafe rewrites in the edge cases cause some loss of accuracy.

6.2 IMDB Dataset Ablation

The IMDB ablation shows that the Knowledge Repository (A4) has a disproportionate impact on aggregation intensive queries. Removing KR boosts MAE on IMDB to 21.3%, but on JOB it's only 15.6%, showing that there are more aggregation anti-patterns and GROUP BY optimizations rules in the repository on IMDB. The Adaptive Feature Weighting (A3) contribution is also larger on IMDB (MAE increase of 14.2%, compared to 12.5% on JOB) because the diversity in the operator mixes in IMDB means that adaptive feature weighting is required to better identify which cost factors are dominant.

Table 9. IMDB Ablation Study Results

Variant	MAE (ms)	RMSE (ms)	MAPE (%)	Speed-up	Opt. Acc. (%)	DBA Score
OptiMind	38.7	55.9	24.1	1.87x	81.6	4.58
No SQL Understand.	46.2	66.7	28.8	1.54x	71.3	4.14
No Profiling	48.9	70.6	30.5	1.47x	68.9	4.09
No Adapt. Weights	44.2	63.8	27.6	1.71x	76.4	4.33
No Knowledg e Repo	47.0	67.8	29.3	1.52x	70.7	4.24
No Attribution	40.1	57.9	25.0	1.82x	79.8	3.54
No Conf. Scoring	42.3	61.0	26.4	1.77x	78.2	4.21
No Validation	38.9	56.2	24.3	1.86x	77.4	4.55

The ablation results of the IMDB further show that components of explainability have the highest impacts on DBA score—particularly A5—and that structural analysis components—particularly A1 and A2—have the highest impacts in MAE. When looking at benefits on IMDB, confidence scoring shows greater benefits than JOB, reducing 6 low confidence recommendations that would have been incorrect if applied, per 100 queries.

6.3 SQLShare Dataset Ablation

SQLShare's high structural diversity creates the most challenging ablation conditions. The effect of the profiling layer is muted relative to JOB and IMDB due to less density in neighborhoods, which means the value of the historical data is lessened. The SQL Understanding Layer (A1) exhibits its best relative contribution on SQLShare (MAE increase of 22.1%) due to the variety of the query structures, where structural analysis is of great value if historical profiles are sparse. When the Knowledge Repository (A4) is not present the performance of SQLShare is reduced by 18.7% MAE, which is similar to the effect on IMDB.

Table 10. SQLShare Ablation Study Results

Variant	MAE (ms)	RMSE (ms)	MAPE (%)	Speed-up	Opt. Acc. (%)	DBA Score
OptiMind	29.4	42.7	19.8	1.61x	78.9	4.43
No SQL Understand.	35.9	52.1	24.2	1.37x	67.4	4.02
No Profiling	34.1	49.4	23.0	1.42x	70.1	4.08
No Adapt. Weights	32.7	47.4	22.0	1.53x	75.3	4.27
No Knowledge Repo	34.9	50.6	23.5	1.39x	68.8	4.17
No Attribution	30.2	43.8	20.4	1.59x	77.1	3.48
No Conf. Scoring	31.8	46.1	21.4	1.54x	74.6	4.19
No Validation	29.6	42.9	19.9	1.60x	73.8	4.40

Confidence scoring (A6) demonstrates the highest absolute contribution with SQLShare, where they are not very well profiled, and where the corresponding sensitivity scores would not be as reliable as they would be if the confidence filter were not included, meaning that the filter would be less effective at reducing accuracy. This verifies the design principle behind density-aware confidence scoring: that it is most useful in the most challenging situation when historical data is minimal.

6.4 Spider Dataset Ablation

The Spider dataset features the largest number of schemas (200) to challenge the most challenging cross-schema generalization. The Knowledge Repository (A4) has the largest relative contribution, with Spider (MAE increase of 22.6% when disabled), which is due to the repository using schema-agnostic optimization rules: predicate pushdown, join commutativity, and index recommendation heuristics, which do not need to be calibrated by schema. The Profiling Layer (A2) has the smallest contribution, as might be expected, since it represents profiling neighborhoods that cross schemas, and is structurally diverse and less informative than neighborhoods within schemas.

Table 11. Spider Dataset Ablation Study Results

Variant	MAE (ms)	RMSE (ms)	MAPE (%)	Speed-up	Opt. Acc. (%)	DBA Score
OptiMind	31.8	46.1	21.3	1.49x	76.4	4.41
No SQL Understand	38.7	56.1	25.9	1.28x	65.8	3.98
No Profiling	35.6	51.6	23.8	1.34x	68.7	4.11
No Adapt. Weights	34.9	50.6	23.4	1.38x	71.2	4.24
No Knowledge Repo	39.0	56.5	26.1	1.26x	65.3	4.13
No Attribution	32.9	47.7	22.0	1.47x	74.9	3.44
No Conf. Scoring	34.1	49.4	22.8	1.42x	73.1	4.16
No Validation	31.9	46.3	21.4	1.48x	71.6	4.38

For Spider, the largest drop across all datasets, the attribution component (A5) shows minimal MAE impact but substantial DBA score impact (from 4.41 to 3.44). This aligns with the need for interpretability when queries are across schemas which are different from what the DBA is familiar with: the Spider evaluation protocol involves higher demand for interpretability for queries across unfamiliar schemas for the DBA.

6.5 Enterprise Workload Ablation

The Profiling Layer (A2) has the largest influence here: MAE is 32.8% higher when disabling Profiling (from 44.1 ms to 58.5 ms), which is quite significant compared to its influence on other datasets. The high regularity of enterprise query templates is explained by the fact that the profiling neighborhood for enterprise queries is dense (mean $|N_k| = 87.4$, delta = 0.85), and workload adaptive weighting approach towards high confidence historical estimates. The contribution of the recommendation validation component (A7) is also the highest across enterprise workloads, with 8.7% drop in optimization accuracy when disabled, due to the enterprise workloads often having more complex rewrite candidates.

Table 12. Enterprise Workload Ablation Study Results

Variant	MAE (ms)	RMSE (ms)	MAPE (%)	Speed-up	Opt. Acc. (%)	DBA Score
Opti-Mind	44.1	63.8	27.6	1.73x	86.1	4.63
No SQL Understand.	53.4	77.2	33.4	1.48x	75.8	4.24
No Profiling	58.5	84.6	36.6	1.38x	72.3	4.19
No Adapt. Weights	50.8	73.4	31.8	1.56x	80.4	4.41
No Knowledge Repo	52.7	76.2	32.9	1.47x	74.9	4.31
No Attributi-on	45.9	66.4	28.7	1.71x	84.3	3.58
No Conf. Scoring	48.3	69.8	30.2	1.64x	81.7	4.38
No Validat-ion	44.3	64.1	27.7	1.72x	77.4	4.61

6.6 Ablation Summary Across All Datasets

The cross-dataset ablation analysis revealed four main results. First, there are two layers that are the biggest contributors to accuracy of the optimizations: SQL Understanding Layer and Profiling Layer, and removing both of them led to an increase in MAE ranging from 38.2% to 51.7%, depending on the dataset. Second, the Feature Attribution mechanism plays a major role in DBA interpretability score, and its absence leads to a decrease in score by 0.93–1.05 points across datasets, irrespective of the effect of MAE. Third, on low density workloads such as SQLShare and Spider, Confidence Scoring has the largest impact on optimization accuracy delta (optimization accuracy minus default accuracy). Fourth, the contribution of the Knowledge Repository is workload dependent - it is best on schema-diverse workloads (Spider), on aggregation-heavy workloads (IMDB).

Table 13. Ablation Component Contribution Summary (Mean Delta MAPE % Across Datasets)

Removed Component	JOB	IMDB	SQL-Share	Spider	Enter-prise
SQL Understand-ing	+4.8%	+4.7%	+4.4%	+4.6%	+5.8%
Profiling Layer	+6.5%	+6.4%	+3.2%	+2.5%	+9.0%
Adaptive Weights	+3.3%	+3.5%	+2.2%	+2.1%	+4.2%
Knowledge Repo	+4.0%	+5.2%	+3.7%	+4.8%	+5.3%
Attribution (XAI)	+1.1%	+0.9%	+0.6%	+0.7%	+1.1%
Confidence Scoring	+2.0%	+2.3%	+1.6%	+1.5%	+2.6%
Validation Layer	+0.1%	+0.2%	+0.1%	+0.1%	+0.1%

6.7 Scalability Analysis

For testing the scalability of OptiMind, the historical queries were increased from 100 to 50,000 in the workload history and the average time taken to process each query, as well as the accuracy of the optimization, were measured. Experimental results show a sub-linear scalability behaviour, which is caused by the

approximate nearest neighbour indexing of structural query representations. The average processing time rose from 312ms for 1000 historical queries to 847ms for 50,000, showing that the framework can be applied practically in the case of processing large workload histories. This overhead is still acceptable for interactive analytical optimization and batch processing applications. Latency-sensitive deployments used a streaming history management approach which limited their retrieval to the 5,000 most recent queries, resulting in an average processing latency of less than 400 ms.

The accuracy of the optimizations increased steadily for larger numbers of workloads in the workload history and reached a convergence plateau. Even though SQLShare and Spider had more structurally diverse and complex cross domains, the two workloads (JOB and IMDB) performed well and reached stable optimization performance at about 10,000 historical queries. Enterprise workloads converged much earlier at about 8,000 queries, due to the fairly homogeneous nature of enterprise query templates. The rest of the historical data showed small gains in optimization accuracy beyond these thresholds and caused a higher processing overhead, suggesting a better balance between scalability and optimization performance.

7. Limitations

Although effective, OptiMind has some major shortcomings, suggesting directions for further research:

1. The framework is based on workload information from the past for workload-aware profiling and recommendation generation. If the workload history is either too short or not available, the profiling layer falls back to structural estimation, and the confidence scores are decreased, and the quality of optimization may be poorer. Experimental testing revealed that the effectiveness of the recommendations decreases by about 12.3% in the first 500 queries for a new deployment. This restriction could be overcome by using workload traces offline for a database initialization process, which takes place in a similar database environment.
2. Due to the changing nature of database engines, schema designs, and new query patterns, the optimization knowledge repository needs to be updated regularly to ensure its compatibility. The repository can be extended, however, there is an engineering burden in keeping accurate and complete optimization rules for various RDBMS platforms and versions.
3. While the semantic validation layer currently supports common SQL transformation and algebraic rewrite patterns, it does not support fully the most complex patterns such as correlated subqueries, nested analytical functions and advanced window-frame specifications. This means that about 7.3% of the optimisation re-writes created need to be manually checked by the DBA before they are deployed.
4. OptiMind is principally designed to run in single-node relational database environments. The execution behavior, operator behaviors, and network-aware cost model for distributed execution platforms like massively parallel processing (MPP) systems, federated query engines, and cloud-native analytical databases are significantly different. The framework needs to be extended to distributed environments, which needs further profiling calibration and distributed cost-model adaptation.

8. Conclusion and Future Work

In this paper, the author has introduced an explainable intelligent SQL query optimization framework called OptiMind, which can overcome the disadvantages of both rule-based and learned optimization methods. Empirical results on six benchmark data sets show consistent improvement over all baselines: MAE decrease of 18.4% – 31.7%, speed up of 1.49x – 2.14x, and DBA interpretability score of 4.51/5.0 on average. Ablation studies also validate the contributions of each architectural component to be independent and complementary. Cross-workload generalization analysis shows that OptiMind is stable across schema-diverse and structurally heterogeneous workloads, as it performance degrades by 8.3% MAPE in worst case cross domain transfer (Worst case is 19.7% for learned optimizer baseline).

Combined, the explainability mechanisms offer a unique DBA-level interpretability, which cannot be achieved by current optimization frameworks. Together, the explainability mechanisms provide a unique DBA-level interpretability, not offered by existing optimization frameworks. The recommendation usefulness rating of 82.4% for all datasets further validates the usefulness of OptiMind's output for database administration work.

Four directions will be explored in future studies. One is the necessity to adapt the profiling layer's operator cost models to reflect network I/O and distributed execution overheads in order to extend to distributed and cloud-native database execution environments like those based on PostgreSQL, like Citus and Aurora. Second, automated knowledge repository expansion based on production deployments of execution plan history will help lower maintenance burd en and increase anti-pattern coverage. Third, embedding OptiMind's recommendation layer in the database engine hint mechanisms to facilitate automatic application of the recommendation in scenarios with low risk and validated code will make the framework more automated while maintaining interpretability. Finally, a user-centric DBA interface will be developed to visualize the feature attributions and sensitivity scores, further enhancing the usability of the explainability output of the framework, and allowing for user context-aware visualization of these scores.

Acknowledgments

The authors' thankful Department of Computer Science, Collage of Science, Mustansiriyah University, for supporting this work.

References

[1] V. Leis, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann, "How Good Are Query Optimizers, Really?" *Proc. VLDB Endowment*, vol. 9, no. 3, pp. 1040–1052, 2015, https://www.vldb.org/pvldb/vol9/p204-leis.pdf

[2] R. Marcus, P. Negi, H. Mao, N. Tatbul, M. Alizadeh, and T. Kraska, "Bao: Making Learned Query Optimization Practical," in *Proc. 2022 Int. Conf. Management of Data (SIGMOD)*, New York, NY, USA: ACM, 2022, pp. 2180–2192. https://dl.acm.org/doi/10.1145/3514221.3526058

[3] A. Pavlo, M. Angulo, A. Arulraj, H. Lin, J. Lin, P. Ma, P. Menon, T. Mowry, M. Perron, I. Quah, S. Santurkar, A. Tomasic, M. Toor, D. Van Aken, Z. Wang, Y. Wu, R. Xian, and T. Zhang, "Self-Driving Database Management Systems," in *Proc. 8th Biennial Conf. Innovative Data Systems Research (CIDR)*, Chaminade, CA, USA, 2017. https://db.cs.cmu.edu/papers/2017/p42-pavlo-cidr17.pdf

[4] L. Woltmann, C. Hartmann, M. Thiele, W. Lehner, and D. Habich, "Learned Cardinality Estimation: An In-Depth Study," *ACM Transactions on Database Systems*, vol. 48, no. 2, Art. no. 7, pp. 1–41, Jun. 2023, doi: 10.1145/3589306.

[5] R. Zhu, Z. Wu, Y. Han, K. Zeng, A. Pfadler, Z. Qian, J. Zhou, and B. Cui, "FLAT: Fast, Lightweight and Accurate Method for Cardinality Estimation," *Proc. VLDB Endowment*, vol. 14, no. 9, pp. 1489–1502, 2021. https://www.vldb.org/pvldb/vol14/p1489-zhu.pdf

[6] A. B. Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. García, S. Gil-López, D. Molina, R. Benjamins, R. Chatila, and F. Herrera, "Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI," *Inf. Fusion*, vol. 58, pp. 82–115, 2020. https://www.sciencedirect.com/science/article/abs/pii/S1566253519308103

[7] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price, "Access Path Selection in a Relational Database Management System," in *Proc. ACM SIGMOD Int. Conf. Management of Data*, Boston, MA, USA, 1979, pp. 23–34, doi: 10.1145/582095.582099.

[8] P. Negi, Z. Wu, A. Gupta, M. Alizadeh, R. Marcus, T. Kraska, S. Madden, and N. Tatbul, "Robust Query Driven Cardinality Estimation Under Changing Workloads," *Proc. VLDB Endowment*, vol. 16, no. 7, pp. 1520–1533, 2023. https://www.vldb.org/pvldb/vol16/p1520-negi.pdf

[9] R. Marcus and O. Papaemmanouil, "Bao: Learning to Steer Query Optimizers," *IEEE Data Eng. Bull.*, vol. 44, no. 2, pp. 11–23, 2021. doi: 10.1145/3514221.3526058.

[10] D. V. Aken, D. Yang, S. Brillard, A. Fiorino, B. Zhang, C. Bilien, and A. Pavlo, "An Inquiry into Machine Learning-Based Automatic Configuration Tuning Services on Real-World Database Management Systems," *Proc. VLDB Endowment*, vol. 14, no. 7, pp. 1241–1253, 2021. https://www.cs.cmu.edu/~pavlo/papers/p1241-aken.pdf

[11] L. Ma, W. Zhang, J. Jiao, W. Wang, M. Butrovich, W. S. Lim, P. Menon, and A. Pavlo, "MB2: Decomposed Behavior Modeling for Self-Driving Database Management Systems," in *Proc. 2021 Int. Conf. Management of Data (SIGMOD)*, New York, NY, USA:

ACM, 2021, pp. 1248–1261. https://dl.acm.org/doi/abs/10.1145/3448016.3457276

[12] E. Triantaphyllou, *Multi-Criteria Decision Making Methods: A Comparative Study*. Boston, MA, USA: Springer, 2000, doi: 10.1007/978-1-4757-3157-6.

[13] T. Tervonen, *Theory and Methods of Multi-Criteria Decision Analysis*. Cham, Switzerland: Springer, 2022. https://link.springer.com/book/10.1007/978-3-030-99079-4

[14] D. B. Blumenthal, N. Boria, J. Gamper, S. Bougleux, and L. Brun, "Comparing Heuristics for Graph Edit Distance Computation," *VLDB J.*, vol. 29, no. 1, pp. 419–458, 2020. https://link.springer.com/article/10.1007/s00778-019-00544-1

[15] S. M. Lundberg, G. Erion, H. Chen, A. DeGrave, J. M. Prutkin, B. Nair, R. Katz, J. Himmelfarb, N. Bansal, and S.-I. Lee, "From Local Explanations to Global Understanding with Explainable AI for Trees," *Nat. Mach. Intell.*, vol. 2, no. 1, pp. 56–67, 2020. https://www.nature.com/articles/s42256-019-0138-9

[16] V. Leis, P. Boncz, A. Kemper, and T. Neumann, "Query Optimization through the Looking Glass, and What We Found Running the Join Order Benchmark," *VLDB J.*, vol. 27, no. 5, pp. 643–446, 668. https://link.springer.com/article/10.1007/s00778-017-0480-7

[17] IMDb Dataset of 50K Movie Reviews (Kaggle)

[18] A. Jain, H. Patel, L. Nagalapatti, N. Gupta, S. Mehta, S. Guttula, S. Mujumdar, S. Afzal, R. Sharma Mittal, and V. Munigala, "Overview and Importance of Data Quality for Machine Learning Tasks," in *Proc. 26th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, 2020, pp. 3561–3562. https://dl.acm.org/doi/abs/10.1145/3394486.3406477

[19] StackOverflow Dataset (Kaggle)

[20] YU, Tao, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In: Proceedings of the 2018 conference on empirical methods in natural language processing. 2018. p. 3911-3921. https://aclanthology.org/D18-1425.pdf