
Journal of Informatics and Web Engineering

Vol. 5 No. 3 (October 2026)

eISSN: 2821-370X

Adaptive and Learned Query Optimization: Techniques, Architectures, and Emerging Trends

Rehab Mohammed Abdelhalim Mokhtar^{1,2*}, Palanichamy Naveen³

¹ Department of Computer Science, Ibn Sina University, Aljerif West Block 5, Building No. 88, PO Box 10995, Khartoum, Sudan

^{2,3} Faculty of Computing and Informatics, Multimedia University, Persiaran Multimedia, 63100 Cyberjaya, Malaysia

**corresponding author: (rehabmoh33@gmail.com; ORCID: 0009-0004-8904-6335)*

Abstract - As databases continue to grow and complexity there is an urgent need to achieve more efficient and faster query processing. This paper presents a technical review of methods that work towards optimization of various types of queries. It progresses by focusing on basic approaches like rules-based optimization and cost-based optimization. Based on this basis, the paper presents a synthesized reference architecture for adaptive and learned query optimizers as a generic framework derived from prior system. It explains on how the classical optimizer components can be enhanced through runtime monitoring integration and feedback mechanisms. From this perspective, it becomes possible to explain how learning modules and execution feedback should be added to the standard optimization pipelines instead of requiring any revolutionary changes in the current database systems. Next, various Adaptive Query Processing (AQP) approaches are reviewed to highlight their strengths and limitations. A fundamental aspect to consider with respect to decision-making process is covered in how effective these methods are from a re-optimization point of view, as well as with adaptive operators. The final section discussed the emerging trends in this field, including some current developments and what is expected to come regarding query optimization alongside machine learning integration, and distributed queries handling advancement issues. By synthesizing insights from existing literature, this review equips researchers with a robust understanding of the current state and future potential of query optimization and adaptive processing techniques.

Keywords—Query Optimization, Adaptive Query Processing, Database Management System, Dynamic Execution, Query Optimizer, Query Cost, Query Plan

Received: 14 June 2025; Accepted: 23 December 2025; Published: 16 October 2026

This is an open access article under the [CC BY-NC-ND 4.0](#) license.



1. INTRODUCTION

The growth of data today is accelerating at an unprecedented rate and in many different forms. This is evident in the era of complex information systems and the rise of distributed databases, where making queries fast and efficient is a must for Database Management Systems (DBMSs) performance [1].

According to IDC, by 2027, 45% of Asia/Pacific organizations will adopt performance-intensive and software-driven infrastructures to support AI and analytics. This growth amplifies optimizer brittleness due to stale statistics, skewed data, and rapidly changing execution environments, highlighting the urgent need for more adaptive and scalable query optimization techniques. [1], [2]

There are various forms of machine learning methods [3], [4] but some of the most common ones include rule based and cost-based approaches which work on assumptions or prior statistics. These techniques are fine in case everything is stable and manageable however they have a number of issues as follows: (i) Data distribution gradually varies with time, (ii) Cost estimates and cardinality may not be accurate enough, (iii) Workloads can be unpredictable or very high and/or (iv) Databases become distributed systems. Hence, constraints lead to an ineffective data retrieval process with real-time applications [5], [6]. As a result of this, the deployment of resources is below optimal levels and the system's scalability is limited.

Adaptive Query Processing (AQP) has emerged as a solution to overcome the limitations of traditional optimization methods. Unlike static optimizers, AQP learns from real-time feedback while a query is being executed and consequently makes changes in the execution plan. This makes it possible for systems to better handle the variations in data characteristics, improve deployment of resources and overall performance and reliability of systems [6], [7]. Table 1 depicts the comparison of traditional optimizers as compared to AQP in terms of execution plan, performance, resource usage and so on.

Table 1. Comparison of Traditional Optimizers and AQP

Feature	Traditional Optimizer	AQP
Execution Plan	Fixed at compile time	Adjusts dynamically at runtime
Performance	Slow on dynamic data	Faster due to adaptation
Resource Usage	Often inefficient	Optimized based on current load
Suitability	Stable, predictable datasets	Dynamic, distributed datasets
Runtime Awareness	Fixed plan, no runtime adaptation	Adjusts execution plan at runtime
Flexibility under changes	Low	High
Handling estimation errors	Poor performance if estimates fail	Corrects plan using actual feedback
Need for pre-training	Not required but limited	Not required, avoids overfitting
Adoption in systems	Common in legacy DBMS	Spark SQL and the TPC-DS benchmark, PostgreSQL, SQLite

There has been evidence from the recent research that introduction of Artificial Intelligence (AI) and Large Language Model (LLM) into AQP can result in higher efficiency in adaptation and execution. AI models deliver more precise estimations of costs, rhythms, while LLMs may provide query hints generation or even transformation of an execution strategy. Eventually such technologies make it possible for the adaptive frameworks to deal with high reliability with the large-scale, complex, and dynamic workloads' management [8], [9], [10].

From Table 1, AQP solves these issues by adding a runtime flexibility to the problem. Instead of static optimizers, AQP modifies the plan execution during the query evaluation based on statistics that are observed rather than outdated estimations. This makes it support in case of cardinality estimation errors and dynamic workload changes. Also, AQP does not require pre-trained models which eliminates the problem of using models that are too precise about certain specifics – something that often causes big problems when using machine learning-based predictors. Thanks to its adaptability AQP has become an inherent feature for such modern systems as Spark SQL, TPC-DS benchmark, PostgreSQL SQLite thus proving its application in solving current large and dynamic data workloads' issue.

Dynamic query optimizers have been modernized to reduce the hydrostatic bias of these algorithms by allowing the optimizer to obtain dynamic statistics while executing queries. That is acceptable since the methods can deliver good performance in a steady state environment but can still supply plans that are far from optimal once actual conditions deviate significantly from initial estimates. For example, if cardinality estimates are not accurate then joins ordering will be poor and thus long query run times will be experienced. It is observed in case study of distributed system like Mariposa that workload distribution and system dynamics often get ignored by standard optimizers which results into performance decay. Research shows that even the most advanced optimizers can suffer significant failures when runtime data distributions are too different from precomputed statistics.

In contrast to the previous surveys that emphasized mainly on the learned optimizers or execution models, this review introduces a unified approach to the classical, adaptive learning based, and LLM-enhanced query optimization. Hence, this paper contributed to the following:

- Outline the classical query optimization approach and problems
- Review various AQP approaches based on the five main proposed groupings
- Visualize the emerging trends of AQP.

As such, this study is a consolidation of both the traditional and existing methods of query optimization, with interesting insights into the implications for adaptive frameworks. Survey shows that as more interests in distributed database systems rise, generic optimizers are failing to address issues of rapidly changing or large-scale workloads complex environment dynamically [11], [12], [13].

2. CLASSICAL QUERY OPTIMIZATION

Classical query optimization continues to serve as the backbone of modern DBMSs [14]. This type of optimizers heavily relies mainly on two optimization paradigms that are rule-based optimization and Cost Based Optimization (CBO). Specifically, rule-based optimization applies a given set of transformation rules which help to rewrite queries into logically equivalent forms this time more performance can be expected in their execution. In contrast, cost-based optimization if compared judges among possible execution plans looking at their costs judged based on statistical information like table cardinalities, data distribution and operation selectivity. Table 2 shows the comparison of rule-based optimization and CBO.

Table 2. Comparison between Rule-Based and Cost-Based Query Optimization

Feature	Rule-Based Optimization	Cost-Based Optimization
Optimization principle	Applies a fixed set of heuristic rules to rewrite queries	Evaluates multiple execution plans and selects the one with the lowest estimated cost
Use of statistics	Does not rely on database statistics	Relies heavily on statistics such as cardinality, selectivity, and data distribution
Plan selection	Deterministic; same query usually produces the same plan	Adaptive to statistics; plan may change when statistics change
Handling of complex queries	Limited effectiveness for complex joins and nested queries	More effective for complex queries when statistics are accurate
Sensitivity to data skew	Low awareness of skewed data distributions	Highly sensitive to inaccurate estimates caused by data skew
Runtime adaptability	No adaptation once the plan is chosen	Limited; decisions are made at compile time
Computational overhead	Low optimization overhead	Higher optimization overhead due to plan enumeration and cost estimation
Robustness in dynamic environments	Performs poorly when workloads or data evolve	Degrades when statistics become stale or inaccurate
Typical usage	Early relational DBMSs and simple query workloads	Widely adopted in modern commercial and open-source DBMSs

Despite cost-based optimization being the primary method used by both commercial and open-source DBMSs, its effectiveness mainly depends on how good the statistics are. The statistic in practice is often represented by these numbers which might be outdated and incomplete, which does not reflect the complex correlations within the data. Thus, classical optimizers may have problems with skewed data distributions, deep join trees, or fast evolving workloads. It has been proven quite recently that even for well-tuned query optimizer cardinality estimates deviating from actual runtime behaviour can potentially lead to much less efficient plans [15], [16] and, consequently, to severe performance degradation

These are particularly challenging in the time of big data analytics or when dealing with a variety of workloads, where most of the computations are performed on parallel and distributed systems. It is not new that decisions made statically

at compile time cannot handle dynamic variations at runtime (volume of data, availability of resources, query patterns). Consequently, researchers start to question the applicability of cost-based approaches and suggest the emergence of the possibility to apply dynamic optimizations or learning mechanisms into the decision process [11], [17]. It is apparent that need for self-adaptive and learning-oriented query optimizers has become one of the central motivations for further query optimization technology [18].

Modern classical optimizers partially mitigate these limitations through (i) richer statistics, such as multi-column statistics and extended histograms, (ii) adaptive re-planning when runtime behaviour deviates from optimizer estimates, (iii) improved cardinality estimation models, and (iv) plan management mechanisms, including plan baselines and regression safeguards. However, these extensions still struggle under rapid workload shifts and in highly distributed environments, where feedback often arrives too late or system conditions change during execution.

3. GENERIC ARCHITECTURE OF QUERY OPTIMIZER

Figure 1 shows a generic architecture that is characteristic of almost all the approaches to adaptive and learning-based query optimization. The process starts off with the query parser, who converts an SQL query into logical form or relational algebra tree. This part is mostly unchanged from classical systems. From there the logical query goes to the optimizer core which develops candidate execution plans. Unlike the traditional optimized algorithms that use only heuristics or cost models, the run time feedback or learned guidance can be employed by adaptive optimizers during this stage.

This module can provide predictions or recommendations based on past executions, e.g. cost estimates, plan rankings or operator level insights but only if a learner chooses to do so. Instead of replacing the optimizer it most often just complements it to improve decision making. The runtime statistics is acquired by the execution engine that executes the selected plan including cardinalities, latencies, and resource usage. This information thus forms a feedback loop which is used in future optimization of the system.

Overall, the key distinction between adaptive and classical query optimizers lies in the use of feedback-driven refinement, not in fundamental architectural changes. This incremental integration model enables many adaptive

For instance, in PostgreSQL-like architectures, the ‘optimizer core’ corresponds to the planner that enumerates join orders and access paths, while the ‘execution engine’ corresponds to the executor that can expose runtime signals (e.g., actual row counts, operator timing). The proposed feedback loop can be implemented by logging execution feedback and using it to update learned components (e.g., plan ranking or cost correction) that guide future planning decisions.

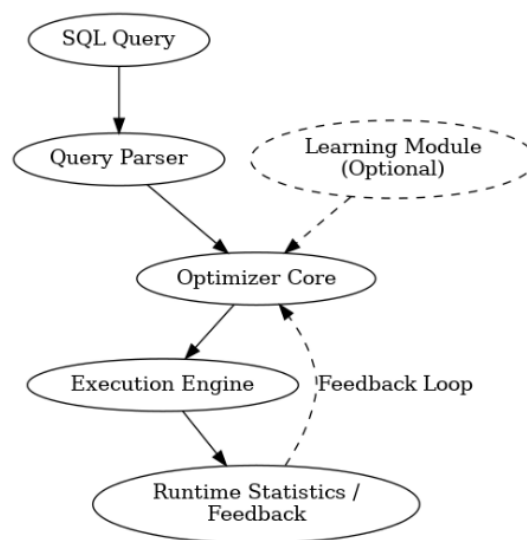


Figure 1. Typical Architecture of Query Optimizer

4. ADAPTIVE QUERY PROCESSING (AQP)

As noted earlier, the study of techniques for improving the efficiency of dynamic AQP is broad and covers a variety of frameworks. In this section, we review recent studies conducted between 2020 and 2025. After analysing these works, we provide comparisons that highlight the strengths and weaknesses, as well as the advantages and limitations, of the algorithms applied in each study. In this review, we group the studies thematically.

AQP is a lightweight, machine learning–based system designed to speed up aggregate queries by providing approximate answers with acceptable accuracy, significantly reducing both cost and response time [19], [20]. Rather than building models or samples over massive base tables, ML-AQP trains small, simple models using only a limited set of previously executed queries and their results which can predict the outcomes of new queries [21]. ML-AQP offers a fast alternative for large-scale analytics, energy-efficient and cost-effective; hence, it is a practical choice for organizations that require fast and accurate query processing. The grouping of AQP is shown in Figure 2, while the typical flow of the AQP is depicted in Figure 3.

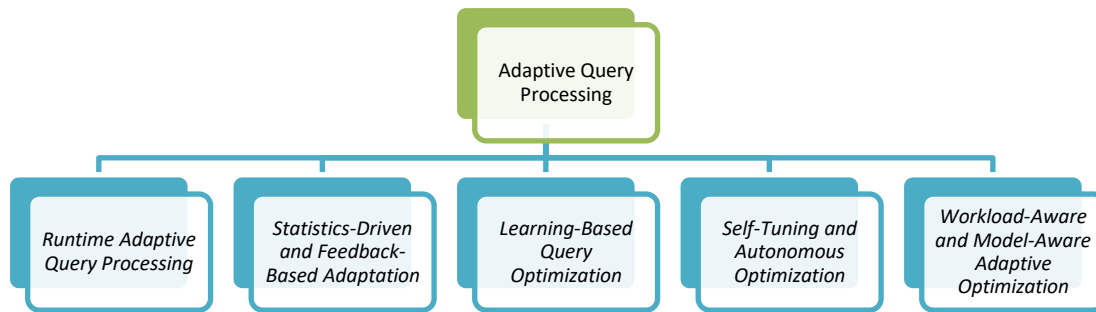


Figure 2. The grouping of AQP

4.1 Runtime Adaptive Query Processing

These works focus on adapting query execution decisions during runtime using observed execution behaviour, sketches, or execution feedback, without relying heavily on long-term learning. Izenov et al. [22] presented COMPASS, which is an online interactive method for sketch-based querying in main memory databases. It only deals with one certain type of statistics: Fast-AGMS graph – it connects selection predicates with downward graph updates and does their evaluation during execution dynamically. This approach provides fast computing of Fast-AGMS graphs and their application to affiliated sets so that incrementally the accuracy of element count estimation and feature-level statistics is improved. The results of experiments revealed that COMPASS improved gradually, cumulative query execution speed increasing from 1.35 to 11.28 times when compared with other methods [23].

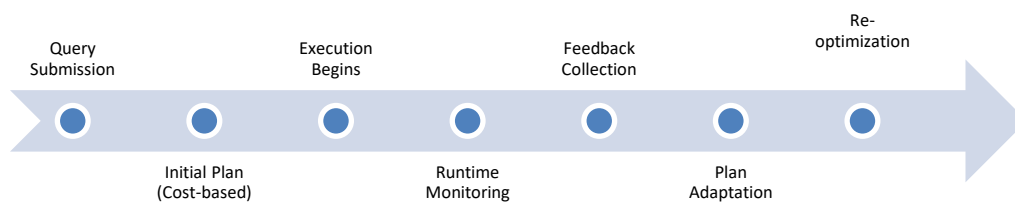


Figure 3. Typical Flow of the AQP

While runtime-adaptive approaches are promising, they may face several practical limitations. These include: (i) delayed or unreliable feedback in highly dynamic environments, (ii) exploration risks that can temporarily lead to suboptimal execution plans and tail latency spikes, and (iii) non-negligible overhead introduced by runtime monitoring and feature extraction.

Marcus et al. [24] introduced Bao, a bandit-based optimizer that combines reinforcement learning and classic query optimizers. It uses three deep neural networks in conjunction with the Thompson sampling to facilitate query optimizations. Rather than reinventing the wheel, it takes advantage of the knowledge embedded in existing optimizers and learns how to adapt on-the-fly as workloads change and databases schemas evolve. The authors noted that a Bao agent can be trained to learn optimal policies in a short amount of time, during which both median and tail latencies for queries decrease, overall cost efficiency is improved for cloud environments. Moreover, their agent shows better generalization than those from previous machine learning-based solutions. Nevertheless, there are some drawbacks of the current approach such as training overhead, integration complexity with multi-tenant cloud systems or a need for further refinement before it could be used in practice by developers or database operators working on regular optimization process.

Lindner [25] proposed a new model for AQP by combining three optimization techniques used in the field across five DBMSs. They then used SQL rewriting to develop engines that discover hidden data dependencies and their origins, such as function dependencies and inclusions within the database, and then used this knowledge to rewrite and optimize SQL queries. By utilizing these hidden dependencies, they achieved a set of results, the most important of which was that the model significantly improved query performance, achieving a 5% - 33% reduction in execution time across different database platforms. One of the drawbacks of this model is that the cost of discovering dependencies is reduced after a single workload is executed.

Table 2 depicts the summary of these approaches. Analysis of COMPASS has indicated that although sketch-based runtime techniques are able to enhance steadily over time the estimation accuracy and execution speed during query processing, their efficiency can be affected by the frequency of the updates. This also may cause some problems since it might be difficult if one tries to perform same with an unstable connectivity. Bao clarifies how to judiciously apply opportunistic learning paradigms over existing optimizers for dynamic workloads yet does so at additional training complexity and deployment practicalities. Dependency-driven optimization, on the other hand, benefits from uncovering hidden relationships in data to improve query rewriting but requires extra effort to discover these dependencies and offers limited improvement once a workload becomes stable.

Table 2. Key Findings on Runtime AQP

Researcher	Findings	Advantages	Drawback
Izenov et al. [22]	Sketch-based optimization using Fast-AGMS graphs for dynamic query execution	Achieved 1.35–11.28× faster execution, improved accuracy of estimations	Limited ability to update data under weak or interrupted internet connections
Marcus et al. [24]	The study showed that Bao can learn useful optimization strategies in a relatively short time. It managed to cut down both the average and the slower query latencies, and it helped lower overall costs when running in cloud environments. Another important point is that Bao seemed more flexible and responsive to workload or schema changes compared to earlier machine learning approaches	One of the main strengths of Bao is that it doesn't need to start learning from zero every time it builds on what the traditional optimizer already knows. This allows it to adapt faster. It also gave clear performance improvements, especially in query speed and efficiency. On top of that, it proved to be greater adaptability when conditions changed suddenly	Requires initial training phase, adds overhead. Integration into multi-tenant cloud systems is complex. Not yet mature to replace traditional optimizers.
Lindner [25]	optimization techniques across five DBMSs and uses SQL rewriting to discover hidden data dependencies, improving query performance with a 5%–33% reduction in execution time across platforms	Leverages hidden dependencies to optimize queries. Works effectively across multiple DBMS platforms. Achieves notable reductions in execution time.	The cost of discovering dependencies is only amortized after executing a single workload.

4.2 Statistics-Driven and Feedback-Based Adaptation

This group improves optimization quality by refining statistics, correcting estimates, or combining feedback with classical cost models, while largely preserving traditional optimizer architectures.

Yang et al. [26], proposed Balsa to show how a query optimizer could learn by itself without the need for expert demonstrations. This model acquires knowledge through self-feedback. It repeatedly plans a particular set of queries, digest their results executed and learns from their response time for building better and faster execution plan in the future incrementally. To model such process Balsa had three main phases: simulation booting, safe execution and safe exploration. This paper gives an insight on using current machine learning methods to make query optimization more efficient. With introduction of innovative tools and approaches it can be used for achieving goals which earlier seemed impossible. The ability of Balsa to bring query optimization at a comparable or even higher level compared with present-day ‘ideal’ optimizers is obtained via automatic trial-and-error learning cycle unsupervised by any expert. Nonetheless, although proficient this approach may appear there is a major drawback – it depends too much on feedback: if the feedback is wrong or unreliable then all following predictions are false.

Zhu et al. [27] proposed Lero, a learning to rank approach for query optimization that is designed to supplement rather than replace traditional cost-based optimizers. Instead of providing an exact value of the execution cost or latency as the input to rank query plans, Lero uses a pairwise approach which trains a binary classifier for each pair of candidate plans to determine which one should be selected. This makes it possible for the model to learn more efficiently and accurately compared to other machine-learning models with previous efforts on regression analysis. Lero can be used with no hassle within current RDBM like PostgreSQL as the only thing that has to be done – its minor modification, at that; moreover, it will be adapted for enquire workloads so that it could keep dealing with novel data types and usage patterns. The method proposed possesses some benefits including significantly faster runtimes of up to 70% less than native optimizer’s runtime, low deployment overhead and adaptiveness towards changing workload. However, the model has its own set of limitations such as; it requires retraining very often when there is a huge change in the data distributions also it depends on DBMS to generate and test multiple plans and consuming time for plan exploration could be introduced.

Eraser system was introduced by Weng and Zhu [28] to mitigate problems of performance regressions in learned query optimisers. These regressions often limit their utility in practice even though they are better than the median performer. Eraser operates in two phases - the first phase is filtering out execution plans that appear risky because they incorporate features not seen before, and the second phase is grouping remaining plans, estimating their prediction accuracy, and selecting the most reliable plan. The main advantage of Eraser is that it can be connected to any learned query optimizer and directs its search for safer plans, all the while being able to enhance the performance of those plans. Tests run on PostgreSQL and Spark revealed the fact that Eraser is capable to adapt to various workloads effectively and provide better satisfaction. Nonetheless, there are some shortcomings related to it: extra overheads caused by the filtering and grouping steps as well as results influenced by underlying enquiries predictions. Consequently, when Eraser decreases regressions issues in learned optimizers, it still preserves these [7] trustworthy issue.

These approaches can be sensitive to noisy, incomplete, or biased feedback. Practical mitigations include: (i) applying outlier filtering and robust aggregation methods (e.g., median or trimmed mean) to stabilize runtime signals, (ii) confidence-aware learning, where low-confidence feedback is down-weighted, (iii) safe fallback mechanisms that revert to the native optimizer when predicted gains are uncertain, and (iv) periodic revalidation on representative workloads to detect model drift.

Table 3 shows the key findings of the approaches reviewed. Balsa demonstrates that query optimizers can learn effective execution strategies through self-feedback without expert guidance, but its heavy dependence on accurate runtime feedback makes it vulnerable to noisy or misleading signals. Lero shows that learning-to-rank can significantly enhance traditional cost-based optimizers with relatively low integration effort, although frequent retraining and plan exploration overhead remain concerns under shifting data distributions. Eraser addresses a practical weakness of learned optimizers by reducing performance regressions through risk-aware plan selection yet introduces additional overhead and continues to rely on the accuracy of underlying prediction models.

Table 3. Key Findings on Statistics-Driven and Feedback-Based Adaptation

Researcher	Findings	Advantages	Drawback
Yang et al. [26]	Yang designed a model called Balsa, the first query optimizer that can learn without any guidance from experts. It plans and executes queries, then learns from their response times to create better execution plans in the future. It works in three stages: starting with a simulator, safe execution, and safe exploration. Tests showed it can reduce the complexity of query optimization and reach performance equal to or better than the best real-world optimization	Learns by itself through trial and error without needing expert training, matches or outperforms state of the art optimizers, uses a safe design that reduces risks to the system during learning	If the feedback data it learns from is wrong, all its predictions will also be wrong, its performance depends heavily on the quality of the data observed during execution.
Zhu et al. [27]	Proposed Lero, a learning-to-rank query optimizer that works alongside native optimizers using a pairwise classification approach to rank query plans.	- Reduces plan execution time by up to 70% compared to PostgreSQL's native optimizer. - Improves performance over other learned optimizers by up to 37%. - Non-intrusive design with minimal integration efforts. - Continuously adapts to query workloads and data changes.	- Requires retraining when data distributions shift significantly. - Depends on the DBMS to generate and test multiple plans. - May incur overhead from additional plan exploration.
Weng and Zhu [28]	Reduces performance regressions in learned query optimizers	Safer plans, works with any optimizer, adapts to workloads.	Extra overhead, depends on base optimizer accuracy, does not fully eliminate regressions

4.3 Learning-Based Query Optimization

The LOGER system was proposed by Chen et al. [29] in 2023, which is a learned query optimizer that is based on Deep Reinforcement Learning (DRL). LOGER employs at first a Graph Transformer which learns to represent the relationships between tables and their predicates, then it rearranges the search space. It doesn't make the job of directly choosing operators, but it reduces the number of operator choices and depend on the internal optimizer of DBMS to take detailed decisions. For achieving higher quality plans, ϵ -beam search is used with LOGER during exploration which keeps multiple promising plans, and a loss function specially designed with reward weighting and log transformation to stabilize training is used. The primary merit of LOGER lies in the fact that it provides both efficient and resilient execution plan alternatives. On JOB, TPC-DS, and Stack Overflow it provided up to $2.07\times$ speedup as compared to PostgreSQL benchmarks, as well as proved satisfactory adaptability to commercial DBMSs. Nevertheless, its drawbacks include dependence on internal optimizer, additional training time frame and restrictions with regards to more intricate queries such as sub-queries handling ability.

Anneser et al. [30] proposed a learning-based framework named AutoSteer that extends the Bao optimizer by automatically generating dynamic hint-sets rather than predicting exact execution cost or latency. It works by steering native SQL optimizers through these hints, making it easily adaptable to multiple systems like PostgreSQL, Presto DB, SparkSQL, MySQL, and Duck DB, with minimal human integration. The main benefit is that AutoSteer improves query performance significantly—up to 40% faster execution in benchmarks and over 20% lower tail latency in real-world Presto DB workloads—while also increasing adaptability and usability with an open-source tool for query experts. However, its drawbacks include dependence on the effectiveness of the generated hints, some overhead during the learning phase, and the fact that it enhances rather than replaces traditional optimizers, so it inherits their limitations.

Panwar [31] advocated the adoption of NoSQL and graph databases in adaptive query management employing AI. The research surveyed all possible forms of enhancing AI driven queries, concentrating on describing how machine learning powered algorithms work and how to employ them for predicting the best ontological plans and execute it correctly based on historical as well as real time data for these queries. Even though focusing on different distributed database systems, he carried out advanced experimental analysis to compare the results of AI with standard optimization techniques. The outcome demonstrate that AI forms an important part in improving performance and efficiency of databases in such models further, making those models more responsive, cost-effective, scalable and agile so they become future trends. One drawback of this model is its applicability only to unstructured databases.

Yao et al. [32] proposed a novel and innovative framework called LLMOpt that is based entirely on LLMs to address these challenges. The framework consists of two main components. The first is a candidate key generation LLM (LLMOpt(G)), which eliminates traditional heuristic search by leveraging the capabilities of LLMs and then using the results to generate candidate keys to generate high-quality, accurate, and straightforward query plans. The second is candidate selection LLM (LLMOpt(S)), which is a cost-oriented, list-based model that compares candidates to test the accuracy of the model. To adapt LLMs to optimize queries, experimental results on the JOB, JOB-EXT, and Stack benchmarks showed that LLMOpt(G) and LLMOpt(S) outperformed the previously mentioned state-of-the-art approaches, including PostgreSQL, BAO, and HybridQO. Notably, LLMOpt(S) achieved the best practical and application performance, striking a balance between plan quality, inference efficiency, and execution speed. A drawback of this model is its complexity, particularly when the data are large.

Table 4 depicts the key findings of the approaches reviewed in this section. LOGER shows that reinforcement learning can be used to restructure the plan search space and improve robustness while still relying on native optimizers, though this dependence limits flexibility and introduces training overhead. AutoSteer demonstrates that learned hint generation can effectively guide existing SQL optimizers across multiple systems, but its performance remains tied to hint quality and inherited limitations of classical optimizers. Panwar's work highlights the potential of AI-driven optimization in NoSQL and graph databases, yet its applicability is largely restricted to unstructured data settings. LLMOpt represents a more radical shift by using LLMs to replace heuristic search entirely, achieving strong performance gains, but at the cost of increased complexity and computational overhead, particularly for large-scale data.

4.4 Self-Tuning and Autonomous Optimization

Eppinger and Störl [33] suggested using machine learning to solve the problem of adjusting NoSQL databases in a way that would address many interdependency issues surrounding configuration parameters. They utilized ensemble methods like Random Forest (RF) and Gradient Boosted Decision Trees (GBDT) as an approach, trained on massive sets representing system properties including replication and sharding, supporting its automatic optimization process to suggest good configurations. The results depicted increased throughput (up to 4%) and both read (43%) and write (39%) latencies reduction in comparison to defaults settings. Nevertheless, its application was rare as it had a reduced scope when employed in hardware elements, low generalization into novel designs as well as insufficient node granularity utilization [38]. This further indicates the necessity of wider tuning domains as well as more open encoding strategies.

Du et al. [34] introduced the genetic Artificial Bee Colony (ABC) approach which is a hybrid of the Gene and ABC algorithms for distributed database systems' query optimization. They have improved the searching ability of the algorithm by introducing selection, cross-over and mutation procedures from the genetic algorithm. This facilitates more efficient search in distributed databases as well as determination of better query plans. The approach was tried in six clusters of varying sizes and results indicated that plans for Top-k queries generated by the algorithm were cheaper to execute and faster converging, thus increased query efficiency. Conversely, the major constraint was that the proposed method required more processing time than the other existing methods which still is a disadvantage.

The comparison on the approaches reviewed is shown in Table 5. Eppinger and Störl [33] showed that machine learning can be effectively used to automate configuration tuning in NoSQL systems and improve throughput and latency, but the approach remains limited in scope and does not generalize well to new architectures or finer-grained system settings. Du et al.'s [34] hybrid genetic-ABC method demonstrates stronger search capability for distributed query optimization and achieves faster convergence for certain workloads, although the added computational cost reduces its practicality for time-sensitive environments. Overall, these self-tuning approaches highlight the potential of autonomous optimization, while also revealing trade-offs between adaptability, scalability, and computational overhead.

Table 4. Key Findings on Statistics-Driven and Feedback-Based Adaptation

Researcher	Findings	Advantages	Drawback
Chen et al. [29]	LOGGER is a DRL-based query optimizer that models table–predicate relationships, narrows operator choices, and stabilizes training using ϵ -beam search and a custom loss function	Generates efficient and robust plans, achieves up to $2.07\times$ speedup over PostgreSQL, and adapts well to commercial DBMSs	Relies on the built-in optimizer, adds training overhead, and struggles with complex queries like subqueries
Anneser et al. [30]	Proposed AutoSteer, a learning-based framework that extends the Bao optimizer by automatically generating dynamic hint-sets instead of relying on static expert-picked ones. It steers native SQL optimizers through these hints, making it applicable to PostgreSQL, PrestoDB, SparkSQL, MySQL, and DuckDB with minimal integration	Improves query performance (up to 40% faster in benchmarks). - Achieves over 20% lower tail latency in Meta’s PrestoDB workloads. - Increases adaptivity by discovering hint-sets automatically. - Provides an open-source tool for query experts.	Dependent on the quality of generated hints. - Incurs some learning overhead. - Enhances but does not replace traditional optimizers, so it inherits their limitations.
Panwar [31]	AI-powered AQP using NoSQL and graph databases can intelligently predict and execute the most efficient query plans by leveraging historical and real-time data, outperforming traditional methods in responsiveness, scalability, cost-efficiency, and agility	Uses AI to predict optimal query execution plans. Improves database performance, scalability, and cost-effectiveness. Works well with unstructured data in distributed systems	Limited to unstructured databases; cannot be applied to relational systems.
Yao et al. [32]	LMOpt, an LLM-based framework, eliminates traditional heuristic search by using LLMs for candidate key generation and selection, achieving superior query plan quality, efficiency, and execution speed compared to PostgreSQL, BAO, and HybridQO on JOB, JOB-EXT, and Stack benchmarks.	Outperforms state-of-the-art query optimizers in both quality and speed. Balances plan quality, inference efficiency, and execution speed. Removes reliance on traditional heuristic search methods.	High complexity, especially when processing large datasets.

While many self-tuning systems have primarily targeted NoSQL configurations, similar principles can be extended to relational DBMSs by tuning: (i) memory and I/O parameters, (ii) parallelism and join execution settings, and (iii) maintenance policies such as auto-vacuum. However, in relational systems, the tighter coupling between optimizer decisions and physical design introduces additional complexity, which necessitates stronger safety constraints and regression control mechanisms.

4.5 Workload-Aware and Model-Aware Adaptive Optimization

Zhang et al. [35] proposed Multi-Agent Database (MADB) that falls within the artificial intelligent systems which brings- about collaboration of multiple learned components to enhance query performance in database systems. The approach integrates a learned query scheduler and a learned query optimizer, both trained within the context of a cooperative multi-agent reinforcement learning framework. This design enables these components to exchange decision contexts during training and execution, working together to reduce query latency as an overall goal. The results showed that MADB was able to decrease query response time significantly which means that it is better than integrations of learned components. Learning to be co-operative did not have good performance, so the approach did

not work well. Also, MADB is more efficient at adjusting to changing data scenarios and finding ways for performance enhancement that older approaches overlooked completely. However, there are pitfalls in this approach: The obtained results are still preliminary and proof-of-concept, there is no large-scale validation, even predefined tables or models should be used limiting the range of optimizable queries and/or extra complexity caused by coordinating multiple learned components during both training and execution [14].

Table 5. Key Findings on Self-Tuning and Autonomous Optimization

Researcher	Findings	Advantages	Drawback
Eppinger and Störl [33]	Applied ML (RF, GBDT) to automatically tune NoSQL DBs, guiding a black-box optimizer to select effective configurations.	Improved throughput (+4%), reduced read latency (43%) and write latency (39%) vs. default settings.	Strong dependence on specific hardware, limited generalization to unseen designs, no node-level customization.
Du et al. [34]	Improved ABC algorithm for distributed DBs; combines GA factors to enhance search and generate optimal query plans.	Lower execution costs for generated Top-k plans. Significant improvement in query efficiency across tested clusters.	Requires relatively longer execution time compared to baseline methods.

Kakkar et al. [23] have proposed a unique framework called Hydro, an AQP framework that addresses all traditional machine learning query problems. This developed framework is dedicated for an efficient processing of machine-learning type of queries in modern database systems. The framework (some of its features) include; UDFs to quickly and accurately generate given query predictions, proper ordering consideration in prediction evaluations and improving UDF implementations scalability. The process proceeds by regularly monitoring those experiments, when a pattern is observed the data will be directed to the predictions in a correct order dynamically changing allocated resources for prediction evaluation Hydro has been tested as shown its remarkable figure giving response time 11.52 times faster than that in baseline model. On the other hand, this framework has limitation: not all its features are supportable by databases [20].

In their research, Nguyen et al. [36] presented a novel model for the problem of self-adapting query execution over distributed databases using reinforcement learning and eXplainable AI (XAI). This approach is one of the most promising ways to tackle this challenge and to avoid some gaps of conventional systems. In that work, a variety of sophisticated methods are employed by the researchers to enhance query processing and optimize partitioned database system. AI being at the initial stage of its evolution, this tool is used for using knowledge that lets it become a self-managed system that adapts to changes in the environment dynamically and improves query performance gradually. Regarding their use of ML methods, they are doing such activities as; performing routine tasks with less importance than modifying configurations or issuing commands among those system elements which do not have a human operator and hence greatly improving query process efficiency. From the results obtained, it was found that the current model enhanced the iterative learning process of system through self-discovering optimum query plans and data/workload dynamic as improved performance with efficient resource utilization and value creation, all without human intervention. Cost effectiveness feature constitutes one disadvantage among others in full intelligent base model development [37].

Table 6 provides the main summary of the two methods. MADB represents the fact that combining several learned parts to enhance the performance and adaptability of a query does not make the approach more effective, although it is still at an experimental stage and introduces some additional complexity owing to coordination and limited validation. Hydro talks about machine learning workloads as it optimizes UDF-based predictions and resource allocation to get significant accelerations even if its functionality is restricted by database support for various features. Work by Nguyen et al. pointed out the possibility of implementation of reinforcement learning in combination with explainability for a self-adjusting distribution query execution system, while the high cost and complexity of fully intelligent systems create difficulties in practical realization.

Table 6. Key Comparison on Semantic-, Workload-, and Model-Aware Adaptive Optimization

Researcher	Findings	Advantages	Drawback
Zhang et al. [35]	MADB (Multi-Agent Database) enhances query performance by enabling collaboration between multiple learned components (a query scheduler and a query optimizer) trained using cooperative multi-agent reinforcement learning	- Significantly improves query response time. - Outperforms non-cooperative learned components. - Adapts flexibly to dynamic data environments. - Identifies optimization opportunities missed by traditional methods.	- Still a proof-of-concept with preliminary results. - Requires large-scale validation. - Relies on pre-defined tables/models, limiting scope of optimization. - Introduces added complexity in coordinating multiple components.
Kakkar et al. [23]	Hydro is an AQP framework designed to handle machine learning queries in modern database systems, generating UDF-based query predictions quickly, optimizing prediction order, and dynamically allocating resources, achieving 11.52× faster response time than the baseline.	- Optimizes prediction order for efficiency. - Dynamically allocates resources based on detected patterns. - Significantly faster (11.52×) than baseline approaches.	Not compatible with relational database systems.
Nguyen et al. [36]	Reinforcement learning combined with explainable AI was used to create an AQP framework for distributed databases that learns iteratively, identifies the most effective query plans, and automatically adapts to changing workloads and data without human intervention, resulting in improved performance and optimal resource use.	- Self-adapts to changing data and workload conditions. Automates configuration changes, resource management, and routine, optimization tasks. - Improves performance and resource utilization without human input.	High development cost for building a fully intelligence-based model.

Figure 4 summarizes the evolution of query optimization techniques discussed in this section, highlighting how systems differ in their reliance on learning mechanisms and runtime adaptivity. The figure shows a clear progression from static cost-based optimizers to adaptive and learning-based approaches, with recent LLM-assisted methods occupying the upper-right region of the design space.

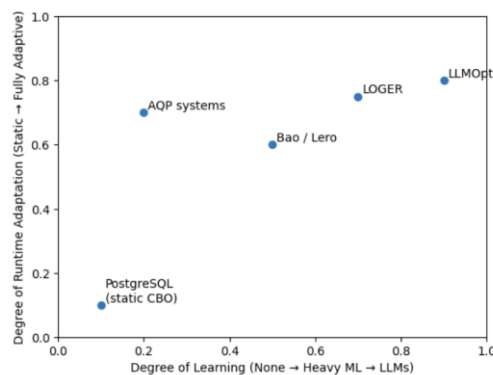


Figure 4. Normalized Progression of Various Query Optimizer

In practice, explainable RL introduces additional complexity at three levels: (i) computational cost, including training overhead and inference latency, (ii) system integration complexity, such as instrumentation, feature pipelines, and safe deployment mechanisms, and (iii) operational overhead, including continuous monitoring, drift detection, and retraining schedules. As a result, many proposals remain at the proof-of-concept stage unless they provide explicit guarantees for tail latency, robust rollback mechanisms, and well-defined cost budgets for online inference.

Table 7 summarizes the datasets and evaluations employed by various researchers.

Table 7. Summary of Dataset and Evaluation

Researcher	Dataset	Evaluation
Yang et al. [26]	Join Order Benchmark (JOB) queries executed on PostgreSQL and a commercial engine.	Measured runtime performance versus PostgreSQL and a commercial optimizer, demonstrating close or superior performance after learning.
Panwar [31]	Collected configuration datasets of NoSQL systems, including key parameters such as replication and sharing. Data used to train ensemble models (RF, GBDT).	- A reduction in query execution time by up to 40% compared to traditional optimization methods was observed on AI-optimized queries. - Lower resource utilization for AI-optimized queries (25% to 30% reduction). - High accuracy in predicting query costs (exceeding 90%).
Zhang et al. [35]	Based on pre-defined tables containing frequently applied queries.	Preliminary experiments with prototype implementation; shows MADB outperforms non-cooperative learned components
Izenov et al. [22]	In-memory database workloads	Achieved $1.35\times - 11.28\times$ cumulative query execution speedup compared to competitors
Chen et al. [29]	JOB, TPC-DS, Stack Overflow	- Experiments show $2.07\times$ speedup on JOB - Competitive results on TPC-DS & Stack Overflow - Similar performance improvements observed on a commercial DBMS
Eppinger and Störl [33]	Performance dataset from Apache Cassandra, capturing replication/sharding configurations and throughput/latency measures.	Compared tuned vs. default Cassandra settings, reporting measurable performance gains. Highlighted dependence on dataset/environment and identified gaps for future research.
Zhu et al. [27]	Benchmarks on PostgreSQL (TPC-H, JOB, and other query workloads).	Experiments show near-optimal performance, with faster execution plans and adaptive learning over time.
Anneser et al. [30]	The dataset is generated synthetic data tailored to their experimental setup. The researcher evaluated their approach across multiple DBMSs.	This data was used to evaluate the proposed approach across multiple DBMSs, with the aim of assessing the impact of leveraging data dependencies on query execution performance.
Du et al. [34]	Six distributed clusters with different sizes (used to test the improved ABC algorithm)	The algorithm generated Top-k query plans that achieved lower execution costs and faster convergence speeds compared to baseline methods, proving improved query efficiency
Kakkar et al. [23]	Benchmarks used for ML query workloads (details not explicitly provided)	Achieved $11.52\times$ faster response time compared to baseline model, proving significant performance gains
Weng and Zhu [28]	Workloads tested on PostgreSQL and Spark using different learned query optimizers, synthetic and real-world query sets.	- Eliminated most regressions. - Improved underperforming optimizers ($1.1\times - 2.9\times$ worse $\rightarrow$ comparable to baseline). - Minimal negative impact when optimizers already perform well. - Adaptive in static & dynamic settings.
Marcus et al. [24]	Standard query benchmarks such as TPC-H and JOB were used to evaluate Bao's performance. Real-world cloud database workloads were also tested to measure adaptability.	- Bao learned effective strategies within a short training period. - It reduced median and tail latencies, improved cost efficiency in cloud environments, and showed stronger adaptability compared to earlier ML-based optimizers.
Yao et al. [32]	JOB, JOB-EXT, and Stack benchmarks were used for evaluation.	LLMOpt consistently achieved better execution time, plan quality, and cost-efficiency when compared to traditional query optimization methods.

5. EMERGING TRENDS

Since the evolution of databases query optimization was and remains the main area in database research. Generally, these methods are based on static cost models and offline selection of the best execution plans. Nevertheless, such techniques usually have bad performance when the data distribution or workload changes with time. This limitation was highlighted by contemporary database systems that have now mostly turned distributed, and their query workloads are unpredictable.

In trustability of learned costs, this is one of the key areas that are still rapidly growing and challenging. For instance, some of the most recent works have demonstrated that expressing uncertainty, and risk in cost models can help to prevent those plans which may have low expected cost but are very sensitive to estimation errors [38]. This way is especially important for data with skewed or evolving distributions since a minor estimation error may result in considerable performance degradation; therefore, it is necessary to perform more detailed studies within different types of systems and workloads.

In another direction, the learning-based optimizers are improved by encoding semantic and structural information. The applications using schema relationships from the training data to learn how to obtain missing information given a partially specified query [39], [40]. Particularly, heuristic approaches have been outperformed in specific scenarios. However, there is room for further research on their scalability and robustness with dynamic changes in schemas and workloads over time. Moreover, besides individual queries, optimization that is more towards workload centric and system-wide approach becomes more important. So, it has been observed that interest in the community on autonomous database systems' research direction is growing steadily [41].

In the realm of research, there is another very tentative area that is worth noting, namely the use of LLM. When it comes to learned optimizers which are normally based on purely numerical features and execution statistics, their thinking is unlike these LLMs as they can reason over symbolic inputs like SQL queries as well as schema descriptions and plan structures [42], [43]. This form of reasoning can be quite useful while performing tasks such as query rewriting, workload characterization and providing high level optimization guidance in contexts where either statistics are non-existent or unreliable. Nevertheless, current limitations related to inference cost, reproducibility, and system integration suggest that LLMs are unlikely to replace core optimization components in the near term [44]. Instead, they are more naturally positioned as auxiliary tools that operate outside the optimizer's critical execution path, for example by generating hints or supporting offline analysis, while primary planning decisions continue to rely on established optimization mechanisms [44], [45], [46].

Generally, the most important trend for real-world use is not only making the optimizer smarter but making it safe and reliable. The production systems appreciate more steady and predictable performance, which includes tail latency rather than some marginal improvements in average performance. It is better to perform slightly slower but always be on the safe side than to run fast from time to time and then show bad results.

In short, there are still important open challenges. First, learning methods should be able to perform well even when the feedback is noisy and the workloads keep changing. Second, monitoring and inference cost should not exceed a certain level for the optimizer itself to become a burden. Third, it must not be evaluated depending only on average latency but also on tail latency metrics, besides considering concurrent workloads, since these better reflect real production scenarios.

6. CONCLUSION

This paper provided the evolution of query optimization from the classical methods to adaptive and learning methods. The survey showed that conventional optimizers still work well for constant workloads, but they often do not perform well in case of data characteristic or execution condition change. To overcome these issues, adaptive processing of queries utilizes runtime feedback for reevaluating execution plans, while methods based on learning enhance optimization decisions through exploitation of past execution behaviour. Majority of these approaches augment classical optimizers instead of replacing them which allows their integration into current DBMSs. The direction of the future work should concentrate on two major issues: (i) the evaluation of how robust a learned optimizer is under varied workloads and (ii) exploration of integrating language models into query optimization pipelines.

ACKNOWLEDGEMENT

The authors would like to thank the anonymous reviewers for the suggestions to improve the paper.

FUNDING STATEMENT

The authors received no funding from any party for the research and publication of this article.

AUTHOR CONTRIBUTIONS

Rehab Mohammed Abdelhalim Mokhtar: Conceptualization, Methodology, Validation, Writing – Original Draft Preparation;

Palanichamy Naveen: Supervision, Writing – Review & Editing;

CONFLICT OF INTERESTS

No conflict of interests were disclosed.

ETHICS STATEMENTS

Our publication ethics follow The Committee of Publication Ethics (COPE) guideline. <https://publicationethics.org/>

DATA AVAILABILITY

Data availability is not applicable to this paper as no new data were created or analysed in this study.

REFERENCES

- [1] Y. Zhang, Y. Chronis, J. M. Patel, and T. Rekatsinas, “Simple Adaptive Query Processing vs. Learned Query Optimizers: Observations and Analysis,” *Proceedings of the VLDB Endowment*, vol. 16, no. 11, pp. 2962–2975, Jul. 2023, doi: 10.14778/3611479.3611501.
- [2] R. Heinrich, M. Luthra, J. Wehrstein, H. Kornmayer, and C. Binnig, “How Good are Learned Cost Models, Really? Insights from Query Optimization Tasks,” *Proceedings of the ACM on Management of Data*, vol. 3, no. 3, pp. 1–27, Jun. 2025, doi: 10.1145/3725309.
- [3] E. A. Anaam, S.-C. Haw, and P. Naveen, “Applied fuzzy and analytic hierarchy process in hybrid recommendation approaches for E-CRM,” *JOIV International Journal on Informatics Visualization*, vol. 6, no. 2–2, p. 553, Aug. 2022, doi: 10.30630/joiv.6.2-2.1043.
- [4] J. Shajeena, R. M. Shiny, K. Gayathri Devi, and M. Mary Vespa, “A Review of Query Optimization Techniques: From Traditional to Reinforcement Learning Approaches,” in *2026 International Conference on Electronics and Renewable Systems (ICEARS)*, Tuticorin, India, Feb. 11–13, 2026, pp. 1185–1189, doi: 10.1109/ICEARS67481.2026.11416629.
- [5] I. Trummer, “Cost-Based Query Optimization for Quantum Computation,” in *Proceedings of the 2nd Workshop on Quantum Computing and Quantum-Inspired Technology for Data-Intensive Systems and Applications*, New York, NY, USA: ACM, Jun. 2025, pp. 1–2. doi: 10.1145/3736393.3736690.
- [6] S.-J. Qiao *et al.*, “Learning database optimization techniques: the state-of-the-art and prospects,” *Front. Comput. Sci.*, vol. 19, no. 12, p. 1912612, Dec. 2025, doi: 10.1007/s11704-025-41116-7.

-
- [7] S. Subramaniam, S.-C. Haw, and L.-K. Soon, "Improved Centralized XML Query Processing Using Distributed Query Workload," *IEEE Access*, vol. 9, pp. 29127–29142, 2021, doi: 10.1109/ACCESS.2021.3058383.
- [8] M. Acosta, "Machine Learning for Query Optimization in Knowledge Graphs," 2026, pp. 183–191. doi: 10.1007/978-3-032-05727-3_18.
- [9] H. Su-Cheng and L. Chien-Sing, "Efficient Preprocesses for Fast Storage and Query Retrieval in Native XML Database," *IETE Technical Review*, vol. 26, no. 1, p. 28, 2009, doi: 10.4103/0256-4602.48466.
- [10] E. Song and S.-C. Haw, "XML-REG: Transforming XML Into Relational Using Hybrid-Based Mapping Approach," *IEEE Access*, vol. 8, pp. 177623–177639, 2020, doi: 10.1109/ACCESS.2020.3026006.
- [11] B. Milicevic and Z. Babovic, "A systematic review of deep learning applications in database query execution," *J. Big Data*, vol. 11, no. 1, p. 173, Dec. 2024, doi: 10.1186/s40537-024-01025-1.
- [12] S. Subramaniam, S.-C. Haw, and L.-K. Soon, "Performance Evaluation of XML Query Processing in Centralized and Distributed Environment," in *Proceedings of the 2019 2nd International Conference on Computational Intelligence and Intelligent Systems*, New York, NY, USA: ACM, Nov. 2019, pp. 129–133, doi: 10.1145/3372422.3372431.
- [13] "Improving Chatbot Performance using Hybrid Deep Learning Approach," *Journal of System and Management Sciences*, vol. 13, no. 3, May 2023, doi: 10.33168/JSMS.2023.0334.
- [14] N. Hettiarachchi and P. Yapa, "Advancements in SQL Query Optimization: A Review of Join Order and Index Selection," in *2025 5th International Conference on Machine Learning and Intelligent Systems Engineering (MLISE)*, IEEE, Jun. 2025, pp. 31–39, doi: 10.1109/MLISE66443.2025.11100192.
- [15] S. Salihoğlu, J. Chen, Y. Huang, M. Wang, and K. Salem, "Cardinality Estimation Graphs," *Communications of the ACM*, vol. 69, no. 2, pp. 99–109, Jan. 2026, doi: 10.1145/3780104.
- [16] L. Woltmann, J. Thiessat, C. Hartmann, D. Habich, and W. Lehner, "FASTgres: Making Learned Query Optimizer Hinting Effective," *Proceedings of the VLDB Endowment*, vol. 16, no. 11, pp. 3310–3322, Jul. 2023, doi: 10.14778/3611479.3611528.
- [17] B. Ding, R. Zhu, and J. Zhou, "Learned Query Optimizers," *Foundations and Trends in Databases*, vol. 13, no. 4, pp. 250–310, Sep. 2024, doi: 10.1561/19000000082.
- [18] R. Radha *et al.*, "AI based Self-Managed Query Optimization Framework for tightly coupled multistore big data systems," *SN Computer Science*, vol. 7, no. 6, Jun. 2026, doi: 10.1007/s42979-026-05170-7.
- [19] O. Gheibi, D. Weyns, and F. Quin, "Applying Machine Learning in Self-adaptive Systems," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 15, no. 3, pp. 1–37, Sep. 2020, doi: 10.1145/3469440.
- [20] P. Mu, A. C. Carniel, A. Barbalace, and A. Shaikhha, "[Experiment, Analysis, and Benchmark] Systematic Evaluation of Plan-Based Adaptive Query Processing," in *2026 IEEE 42nd International Conference on Data Engineering (ICDE)*, 2026, pp. 3501–3514, doi: 10.1109/ICDE65706.2026.00257.
- [21] S. A. Peious, M. Kaushik, M. Shahin, R. Sharma, and D. Draheim, "On Choosing the Columnar In-Memory Database Hyrise as High-Performant Implementation Platform for the GrandReport Tool," in *2025 International Conference on Next Generation Information System Engineering (NGISE)*, IEEE, Mar. 2025, pp. 1–8, doi: 10.1109/NGISE64126.2025.11085185.
- [22] Y. Izenov, A. Datta, F. Rusu, and J. H. Shin, "COMPASS: Online Sketch-based Query Optimization for In-Memory Databases," in *Proceedings of the 2021 International Conference on Management of Data*, New York, NY, USA: ACM, Jun. 2021, pp. 804–816, doi: 10.1145/3448016.3452840.
- [23] G. T. Kakkar, J. Cao, A. Sengupta, J. Arulraj, and H. Kim, "Hydro: Adaptive Query Processing of ML Queries," *arXiv preprint arXiv:2403.14902*, Mar. 2024, doi: 10.48550/arXiv.2403.14902.

- [24] R. Marcus, P. Negi, H. Mao, N. Tatbul, M. Alizadeh, and T. Kraska, “Bao: Making Learned Query Optimization Practical,” in *Proceedings of the 2021 International Conference on Management of Data*, New York, NY, USA: ACM, Jun. 2021, pp. 1275–1288. doi: 10.1145/3448016.3452838.
- [25] D. Lindner, D. Ritter, and F. Naumann, “Enabling Data Dependency-based Query Optimization,” *arXiv preprint arXiv:2406.06886*, Jun. 2024, doi: 10.48550/arXiv.2406.06886.
- [26] Z. Yang, W.-L. Chiang, S. Luan, G. Mittal, M. Luo, and I. Stoica, “Balsa: Learning a Query Optimizer Without Expert Demonstrations,” in *Proceedings of the 2022 International Conference on Management of Data*, New York, NY, USA: ACM, Jun. 2022, pp. 931–944. doi: 10.1145/3514221.3517885.
- [27] R. Zhu et al., “Lero: A Learning-to-Rank Query Optimizer,” *Proceedings of the VLDB Endowment*, vol. 16, no. 6, pp. 1466–1479, 2023, doi: 10.14778/3583140.3583160.
- [28] L. Weng, R. Zhu, D. Wu, B. Ding, B. Zheng, and J. Zhou, “Eraser: Eliminating Performance Regression on Learned Query Optimizer,” *Proceedings of the VLDB Endowment*, vol. 17, no. 5, pp. 926–938, Jan. 2024, doi: 10.14778/3641204.3641205.
- [29] T. Chen, J. Gao, H. Chen, and Y. Tu, “LOGGER: A Learned Optimizer towards Generating Efficient and Robust Query Execution Plans,” *Proceedings of the VLDB Endowment*, vol. 16, no. 7, pp. 1777–1789, 2023, doi: 10.14778/3587136.3587150.
- [30] C. Anneser et al., “AutoSteer: Learned Query Optimization for Any SQL Database,” *Proceedings of the VLDB Endowment*, vol. 16, no. 12, pp. 3515–3527, Aug. 2023, doi: 10.14778/3611540.3611544.
- [31] V. Panwar, “AI-Driven Query Optimization: Revolutionizing Database Performance and Efficiency,” *International Journal of Computer Trends and Technology*, vol. 72, no. 3, pp. 18–26, Mar. 2024, doi: 10.14445/22312803/IJCTT-V72I3P103.
- [32] Z. Yao, H. Li, J. Zhang, C. Li, and H. Chen, “A Query Optimization Method Utilizing Large Language Models,” *arXiv preprint arXiv:2503.06902*, Mar. 2025, doi: 10.48550/arXiv.2503.06902.
- [33] F. Eppinger and U. Störl, “Tuning Cassandra through Machine Learning,” *Lecture Notes in Informatics (LNI)*, Gesellschaft für Informatik, Bonn, Germany, 2023, pp. 93–104, doi: 10.18420/BTW2023-04.
- [34] Y. Du, Z. Cai, and Z. Ding, “Query Optimization in Distributed Database Based on Improved Artificial Bee Colony Algorithm,” *Applied Sciences*, vol. 14, no. 2, p. 846, Jan. 2024, doi: 10.3390/app14020846.
- [35] C. Zhang, O. Papaemmanouil, and J. Hanna, “Multi-agent Databases via Independent Learning,” *arXiv preprint arXiv:2205.14323*, 2022, doi: 10.48550/arXiv.2205.14323.
- [36] T.-G.-B. Nguyen et al., “Towards Explainable Polypharmacy Risk Warnings Using Reinforcement Learning on Knowledge Graphs,” 2025, pp. 582–600, doi: 10.1007/978-3-031-90194-2_41.
- [37] S. Milani, N. Topin, M. Veloso, and F. Fang, “Explainable Reinforcement Learning: A Survey and Comparative Review,” *ACM Comput. Surv.*, vol. 56, no. 7, pp. 1–36, Jul. 2024, doi: 10.1145/3616864.
- [38] A. Kamali, V. Kantere, C. Zuzarte, and V. Corvinelli, “Robust Plan Evaluation based on Approximate Probabilistic Machine Learning,” *Proceedings of the VLDB Endowment*, vol. 18, no. 8, pp. 2626–2638, 2025, doi: 10.14778/3742728.3742753.
- [39] S. Yue, Y. Shao, and S. Hayes, “OntoTune: Ontology-Driven Learning for Query Optimization with Convolutional Models,” *Arxiv*, Nov. 2025, <https://doi.org/10.48550/arXiv.2511.06780>
- [40] L. D. Krisnawati, A. W. Mahastama, S.-C. Haw, K.-W. Ng, and P. Naveen, “Indonesian-English Textual Similarity Detection Using Universal Sentence Encoder (USE) and Facebook AI Similarity Search (FAISS),” *CommIT*

- (Communication and Information Technology) Journal, vol. 18, no. 2, pp. 183–195, Sep. 2024, <https://doi.org/10.21512/commit.v18i2.11274>
- [41] J. R. Haritsa, “Robust Query Processing: A Survey,” *Foundations and Trends in Databases*, vol. 15, no. 1, pp. 1–114, Dec. 2024, doi: 10.1561/19000000089.
- [42] J. Zhu, L. Chen, X. Ke, Z. Fang, T. Li, Y. Gao, and C. S. Jensen, “Beyond Relational: Semantic-Aware Multi-Modal Analytics with LLM-Native Query Optimization,” *Proceedings of the ACM on Management of Data*, vol. 4, no. 1, Art. no. 14, pp. 1–25, 2026, doi: 10.1145/3786628
- [43] S.-C. Haw, J. W. May, and S. Subramaniam, “Mapping Relational Databases to Ontology Representation,” in *Proceedings of the 1st International Conference on Digital Technology in Education*, New York, NY, USA: ACM, Aug. 2017, pp. 54–58, doi: 10.1145/3134847.3134852.
- [44] D. Satriani, E. Veltri, D. Santoro, S. Rosato, S. Varriale, and P. Papotti, “Logical and Physical Optimizations for SQL Query Execution over Large Language Models,” *Proceedings of the ACM on Management of Data*, vol. 3, no. 3, pp. 1–28, Jun. 2025, doi: 10.1145/3725411.
- [45] L. Shi, Z. Tang, N. Zhang, X. Zhang, and Z. Yang, “A Survey on Employing Large Language Models for Text-to-SQL Tasks,” *ACM Comput. Surv.*, vol. 58, no. 2, pp. 1–37, Jan. 2026, doi: 10.1145/3737873.
- [46] J. Tan *et al.*, “Can Large Language Models Be Query Optimizer for Relational Databases?,” *Proceedings of the ACM on Management of Data*, vol. 3, no. 6, pp. 1–28, Dec. 2025, doi: 10.1145/3769771.

BIOGRAPHIES OF AUTHORS

	Rehab Mohammed Abdelhalim is a lecturer in Computer Science at Ibn Sina University in Sudan and a current PhD candidate at Multimedia University, Malaysia. Her research focuses on Adaptive and Learned Query Optimization: Techniques, Architectures, and Emerging Trends. She has authored a review paper on adaptive query processing and query optimization, contributing to the growing field of intelligent data management. She can be contacted at email: REHAB.MOHAMMED.ABDELHALIM@student.mmu.edu.my
	Naveen Palanichamy joined the Faculty of Computing and Informatics, Multimedia University after receiving PhD from Curtin University, Malaysia. She received her Bachelor of Engineering (CSE) and Master of Engineering (CSE) from Anna University, India. Her research interest includes Smart Grid, Cloud Computing, Machine Learning, Deep Learning and Recommender system. She is involved in multiple research projects funded by Multimedia University. She can be contacted at email: p.naveen@mmu.edu.my