Journal of Computer Technology and Software

ISSN: 2998-2383

Vol. 3, No. 5, 2024

Machine Learning Framework for Performance Prediction and Intelligent Resource Allocation in Complex Data Environments

Ming Wang
Northeastern University, San Jose, USA
mingwangcr@gmail.com

Abstract: This paper focuses on the problem of database query execution time prediction and optimization. To address the limitations of traditional methods that suffer from error accumulation and insufficient scheduling efficiency in complex query scenarios, it proposes a comprehensive framework that integrates structured modeling with adaptive scheduling. First, a Plan-Graph Guided Latency Modeling (PGLM) mechanism is designed, which explicitly incorporates structural features of query plans to enhance the model's awareness of operator patterns and join topologies, thereby improving prediction accuracy and generalization. Second, an Adaptive Query – Resource Orchestrator (AQRO) is constructed to dynamically match query demands with system resources under a prediction – execution interaction mechanism, ensuring continuous satisfaction of service-level objectives (SLOs) and maintaining system stability. The proposed method demonstrates strong robustness under different hyperparameters, resource quotas, and query template diversity, achieving low prediction errors and reasonable uncertainty calibration in dynamic environments. The results show that the framework performs well in both latency prediction and resource optimization, providing a new technical path for database system performance improvement.

Keywords: Query execution prediction; resource orchestration; latency modeling; system optimization

1. Introduction

In today's data-intensive applications, the efficiency of database query execution directly affects system service quality and user experience. With the continuous growth of data volume and the diversification of application requirements, achieving accurate execution time prediction and efficient resource scheduling in complex query plans has become a critical problem in database optimization. Accurate latency prediction not only helps avoid performance bottlenecks in advance but also provides key support for resource allocation and query optimization strategies, thereby improving overall system stability and response speed. Therefore, research on intelligent methods for query execution time prediction and optimization strategies carries significant theoretical and practical value[1].

However, existing methods still face many challenges in complex query scenarios. Traditional cost models are unable to adapt to the dynamic changes of data distribution and system states, often leading to the accumulation of prediction errors. Data-driven learning models improve prediction accuracy but still suffer from limitations in generalization and robustness. In particular, when facing diverse query templates and uncertain runtime environments, the predictions often deviate from actual performance. At the resource scheduling level, current strategies lack fine-grained modeling of the coordination between query requirements and system resources, leading to uneven allocation and unstable system load, which negatively affects the satisfaction of service-level objectives (SLOs)[2].

To address these problems, this study introduces a comprehensive method that integrates query plan structural information with resource scheduling mechanisms[3]. On one hand, a plan-graph-based latency modeling mechanism is constructed to enhance the model's structured perception of query execution processes, enabling more accurate latency estimation during prediction. On the other hand, an adaptive query – resource orchestration strategy is incorporated to achieve dynamic matching between query workloads and system resources, improving prediction stability and optimization performance in diverse scenarios. This bidirectional integration aims to form positive feedback between prediction and scheduling, driving overall improvements in database performance optimization[4].

The contributions of this paper lie in two main aspects. First, we propose Plan-Graph Guided Latency Modeling (PGLM), which explicitly incorporates query plan structural features into the prediction process. This enhances the model's ability to represent and understand complex query topologies, thereby improving prediction accuracy and generalization. Second, we design an Adaptive Query – Resource Orchestrator (AQRO), which achieves adaptive alignment between query demands and system resources under a prediction – execution interaction framework, balancing performance improvement with resource utilization efficiency. Together, these two innovations construct an end-to-end intelligent optimization framework that provides a new solution for query prediction and optimization in databases[5].

2. Related work

2.1 Query Execution Time Prediction: From Cost Models to Data-Driven Learning

Traditional research has mainly focused on cost models based on rules and parameterized assumptions. Query execution time is decomposed into the sum of operator-level CPU, I/O, and network costs, fitted through cardinality estimation, selectivity, and cost tables. In earlier single-node, row-store architectures, these methods offered good interpretability and ease of implementation. However, modern database systems introduce columnar compression, vectorized execution, parallel pipelines, JIT compilation, compute-storage separation, and acceleration hardware. Latency is no longer a simple linear sum of independent operator costs[6]. Cache penetration, memory grants, concurrency, and resource governance strategies introduce strong nonlinearity and cross-layer coupling. In distributed settings, data skew, shuffle, retries, and fallback amplify the cascading effect of cardinality errors, making static cost models prone to systematic bias under mixed workloads, multi-tenant deployments, and elastic resource environments[7].

In response to increasing complexity, data-driven learning-based prediction has become an important direction. These methods typically rely on execution logs and construct multi-granularity features around query plans, data characteristics, and system telemetry. The features include logical and physical operator sequences, join graph density, predicate complexity, deviations between estimated and observed cardinalities, pipeline depth, concurrency, memory grants, cache hit rate, disk utilization, and network utilization[8]. One class of methods performs operator-level or stage-level latency regression and then combines results to obtain overall query latency. Another class directly performs end-to-end prediction using nonlinear models to capture how join order, selectivity, and data skew amplify effects along the critical path. Compared with traditional cost models, learning-based methods better accommodate heterogeneous hardware and dynamic resource strategies, and provide fine-grained signals for identifying performance bottlenecks under different throttling and scheduling policies[9].

To enhance the representation of query plan structures and execution dependencies, recent studies emphasize structured representation learning. Typical approaches treat query plans as trees or directed acyclic graphs and use graph or sequence encoders to capture operator-level data flow dependencies, parallel or blocking relationships, and cross-stage interference. Attention mechanisms are introduced to explicitly model critical paths and bottleneck operators. Context encoders are used to integrate static plans with runtime states, enabling models to respond to transient resource fluctuations and plan re-optimization. For long-term availability in production, online updating, incremental learning, and concept drift detection have been proposed. These are often combined with uncertainty estimation and calibration techniques, which ensure prediction accuracy while providing confidence intervals to reduce risks from incorrect decisions[10,11].

Learning-based methods also face challenges in data quality and generalization. Execution logs often contain missing values, noise, and skewed distributions, while extreme tail latencies significantly affect training and evaluation. Variations across workloads, schemas, and workload evolution cause feature distribution shifts. Cross-engine, cross-cluster, and cross-cloud deployment requires models with domain adaptation and parameter-efficient updating. Privacy and compliance restrictions limit data aggregation across tenants, motivating exploration of weakly supervised, semi-supervised, and privacy-preserving learning. Mechanisms such as UDFs, approximate queries, and materialized view selection introduce unobservable or hard-to-quantify variables. In response, research has proposed feature governance, robust losses, resampling, and reweighting strategies. Hierarchical, multi-task, and multi-objective models have also been introduced to jointly capture both average and high-percentile latencies, thereby providing a more stable predictive foundation for plan selection, mid-query re-optimization, and resource orchestration[12].

2.2 Execution-Time Optimization Strategies: Adaptive Query Processing and Resource Orchestration

Execution-time optimization focuses on runtime coordination between query plans and resources. Its core lies in the synergy of adaptive query processing and resource orchestration. The former addresses uncertainty caused by statistical drift, concurrency fluctuation, and data skew, aiming to continuously correct false assumptions and converge to better execution paths. The latter emphasizes elastic allocation and global scheduling of compute, storage, and network resources under multi-tenant and heterogeneous environments with service-level objectives as constraints. A key prerequisite for their joint effectiveness is the construction of observable links across the plan, operator, and system layers. This requires exposing feedback on estimation errors between logical and physical plans, collecting fine-grained runtime metrics in the execution engine, and providing delay-sensitive scheduling interfaces and quota controls in resource management. These mechanisms ensure that optimization can take effect in a closed-loop and timely manner[13].

The research paradigm of adaptive query processing focuses on in-flight correction. Typical approaches include monitoring deviations in cardinality and selectivity during execution and triggering phase re-optimization to adjust join order and operator implementation. Multiple candidate strategies can be preset for critical operators and switched with lightweight overhead once thresholds are exceeded or confidence levels updated[14]. Operator parallelism, batch size, and buffer thresholds can be adjusted dynamically according to memory and I/O pressure, suppressing blocking chains and rollback amplification. In data skew scenarios, hot keys can be resampled or avoided by splitting long-tail tasks into balanced subtasks. Incremental indexes and materialized views can be activated on demand to reduce memory and network overhead along critical paths. Protection points can be placed in pipelines that are sensitive to estimation errors, where lightweight statistics and micro-reordering are inserted without breaking pipeline parallelism, balancing robustness and throughput[15].

Resource orchestration adopts a global perspective to coordinate multiple constraints. In cluster and multi-cloud environments, queries are split into independently schedulable

stages or task groups. These are placed with affinity according to data locality and network topology, reducing cross-switch traffic and hotspot congestion. Admission control and throttling strategies driven by service-level objectives and latency budgets are introduced into queues and priority hierarchies, ensuring reserved resources and priority for critical requests. Quotas and isolation are applied to CPU, memory, storage, and network resources, enhanced by NUMA awareness and accelerator binding, improving utilization efficiency. For hybrid transactional and analytical workloads, online workload classification and shaping are used to prevent long analytical tasks from starving interactive short queries. In distributed execution, speculative execution and replica diversion alleviate long-tail effects, while cache layers and staged prefetch reduce cold-start and jitter amplification[16].

At the methodological level, more optimization strategies incorporate predictive and learning signals to enhance adaptiveness and foresight. At the query layer, joint prediction of execution time and resource usage can be embedded into cost evaluation, forming a prediction – decision – feedback loop. At the system layer, scheduling and scaling can be modeled as constrained sequential decision problems, updated with historical telemetry and online observation. For multi-objective trade-offs, delay, throughput, cost, and energy are jointly optimized with robust regularization to prevent overfitting on a single metric. For cross-environment generalization, domain adaptation and parameter-efficient updates allow strategies to transfer across engines, clusters, and tenants. For engineering usability, uncertainty estimation, and protective thresholds limit re-optimization frequency and switching overhead, ensuring that optimization benefits exceed control costs in high-concurrency environments. Fallback mechanisms are also required to mitigate prediction errors and strategy failures, providing safe recovery paths[17].

3. Method

This paper proposes an end-to-end closed-loop framework for database query processing that follows the line of prediction, decision, and feedback, covering both offline modeling and online optimization, and incorporating two key innovations. The first is Plan-Graph Guided Latency Modeling (PGLM). This module abstracts logical and physical plans into attributed directed acyclic graphs and integrates operator sequences, pipeline boundaries, join structures, deviations between estimated and observed cardinalities, concurrency, and multi-source telemetry of memory, I/O, and network. It constructs multi-granularity features with temporal context encoding and outputs calibrated uncertainty distributions to capture both mean and high-percentile latency characteristics, while also providing interpretable cues for critical paths and bottleneck operators. The second is the Adaptive Query – Resource Orchestrator (AQRO). This module uses endogenous predictions and confidence information as priors to coordinate plans and resources at runtime. It triggers join reordering and operator switching, adjusts parallelism and batch size according to load and pressure, mitigates data skew through resampling and task splitting, and applies elastic orchestration of CPU, memory, storage, and network with quotas and affinity placement, while maintaining stability and availability through constrained optimization and switching cost control. The overall workflow consists of data collection and feature governance, PGLM training and online inference, AQRO policy generation and execution monitoring, which are integrated into the optimizer and resource manager through standardized interfaces to provide execution-time-aware adaptive query processing and resource orchestration in complex and dynamic multi-tenant, heterogeneous, and cloud-native environments. The overall model framework diagram mentioned is shown in Figure 1.

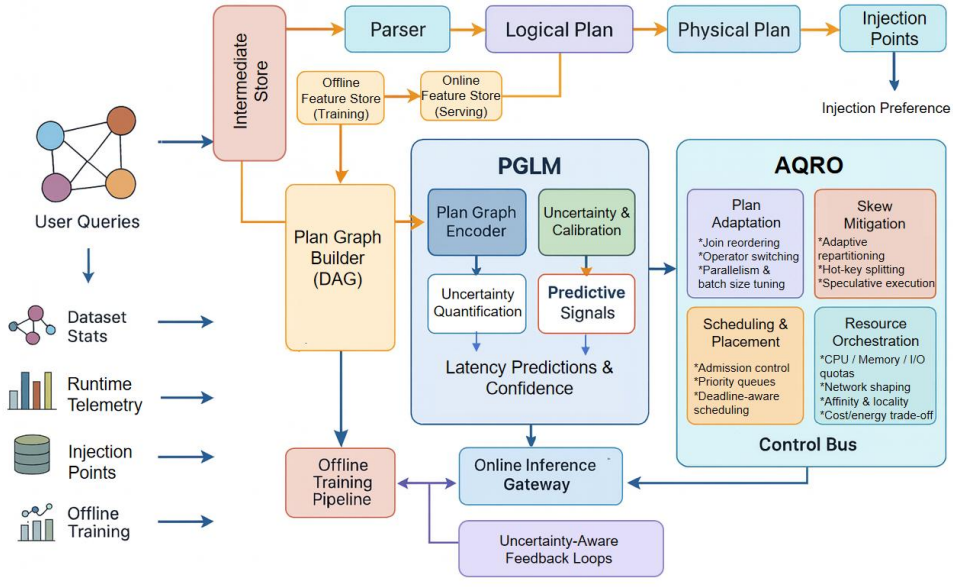

Figure 1. Overall model architecture diagram

3.1 Plan-Graph Guided Latency Modeling

This study introduces a Plan-Graph Guided Latency Modeling (PGLM) method. The core idea is to abstract logical and physical database plans into attributed directed acyclic graphs and then perform graph structure modeling and temporal context encoding. By integrating operator types, pipeline boundaries, cardinality estimation errors, concurrency, and multi-source telemetry, PGLM captures critical paths and bottleneck operators at the graph structure level. This enables fine-grained and interpretable predictions of query execution time. Unlike traditional static cost models, PGLM does not rely on fixed parameters. Instead, it learns latent dependencies and amplification effects automatically through a structured graph representation, which makes it more adaptable to complex runtime environments.

PGLM further transforms the prediction problem into modeling conditional probability distributions rather than single-point estimation. This distributional view characterizes not only the mean execution time but also calibrated outputs for high-percentile latencies such as P95 and P99, which are critical in real systems. By incorporating uncertainty modeling, PGLM produces joint representations that include mean, variance, and confidence. This allows subsequent resource orchestration and adaptive query processing to make decisions based on confidence constraints instead of single-point estimates, achieving a balance between stability and performance. The framework of this innovation is illustrated in Figure 2.

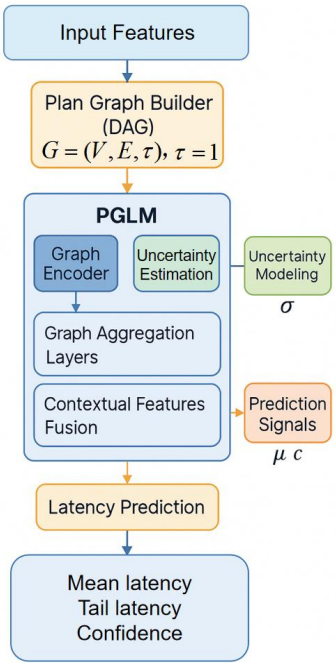

Figure 2. PGLM module architecture

At the formula reasoning level, we first model the query plan as a weighted graph. Let's assume the query plan is a directed acyclic graph $G=(V,E)$, where V represents the operator nodes and E represents the dependency edges between operators. For each node $v \in V$, its representation vector can be written as:

$$h_v = \phi\left(x_v, \sum_{v \in N(v)} \psi(h_u, e_{uv})\right)$$

x_v represents the operator characteristics (such as type, selectivity, parallelism, etc.), e_{uv} represents the edge attributes (such as partitioning method, blocking relationship), and ϕ and ψ are learnable functions.

Based on the node representation, we represent the potential execution time of the entire query as a graph-level embedding z:

$$z = READOUT\left(\{h_v \mid v \in V\}\right)$$

Where $READOUT(\cdot)$ represents a graph-level aggregation operation, such as weighted averaging or attention aggregation.

To characterize the uncertainty of execution time, we model its conditional probability distribution:

$$p(y \mid G) = N(y \mid \mu(z), \sigma^2(z))$$

Where y represents the actual execution time, $\mu(z)$ and $\sigma^2(z)$ are the predicted mean and variance, respectively. The model outputs not only the expected value but also the confidence interval, thus avoiding over-reliance on a single estimate during optimization.

In the loss function design, we use negative log-likelihood as the optimization target:

$$L_{NLL} = \frac{1}{2}\log \sigma^2(z) + \frac{(y-\mu(z))^2}{2\sigma^2(z)}$$

This loss function constrains both the mean and variance of the predictions, allowing the model to learn a stable and calibrated probability distribution. To further focus on tail delays, we introduce quantile regression loss:

$$L_\tau = \max(\tau(y-\hat{y}), (\tau-1)(y-\hat{y}))$$

Where $\hat{y}$ is the predicted τ quantile (such as $\tau = 0.95$ or 0.99). The final loss function is a weighted combination:

$$L = L_{NLL} + \lambda \sum_{\tau \in \{0.95, 0.99\}} L_\tau$$

The role of the loss function in this part is to transform execution time prediction into a joint optimization problem. It requires not only an accurate mean prediction but also a reliable characterization of tail latency. By constraining both the distribution center and tail features, the model outputs predictions that are representative and robust, providing a solid probabilistic foundation for subsequent adaptive query optimization and resource orchestration.

3.3 Adaptive Query – Resource Orchestrator

This study also introduces an Adaptive Query – Resource Orchestrator (AQRO). Its main goal is to achieve dynamic alignment between computing resources and query plans during execution, thereby improving overall performance and fairness in multi-tenant and heterogeneous environments. Unlike traditional scheduling methods that rely on static rules, AQRO makes adaptive decisions based on predicted execution signals such as mean latency, tail latency, and confidence intervals. It enables joint optimization at both the query and resource levels. This coordination allows the system to maintain stable resource utilization under high load and latency-sensitive scenarios while effectively mitigating performance fluctuations and tail latency amplification.

In addition, the design of AQRO emphasizes cross-layer feedback by incorporating prediction uncertainty into resource orchestration strategies. By combining structural features of query plans with runtime resource states, AQRO can dynamically adjust operator scheduling, parallelism allocation, resource affinity, and batch granularity. Importantly, AQRO considers not only average performance objectives but also introduces percentile latency constraints into scheduling to achieve proactive control of tail latency. This design provides a unified control bus for resource scheduling and query optimization, enabling the system to flexibly switch between precise prediction and uncertainty defense. The framework of this model is illustrated in Figure 3.

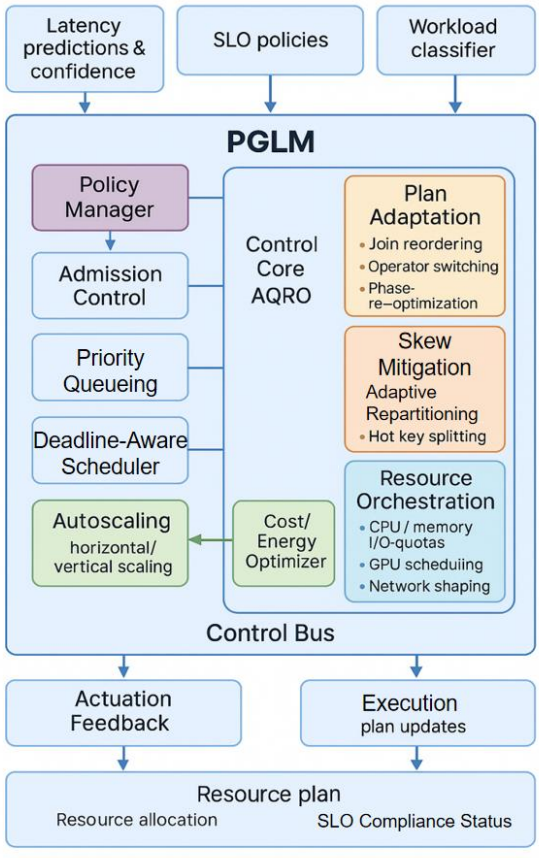

Figure 3. AQRO module architecture

At the formula reasoning level, assume the system has a query set $Q = \{q_1, q_2, ..., q_N\}$ and a resource set $R = \{r_1, r_2, ...r_M\}$. The predicted latency of each query q_i is output by PGLM, including the mean μ_i, variance σ_i^2, and quantile prediction $\hat{y}_i^{(\tau)}$. The resource allocation can be represented as a matrix:

$$A = [a_{i,j}] \in \{0,1\}^{N \times M}$$

$a_{i,j} = 1$ Indicates that the query q_i is assigned to a resource r_j.

Under this constraint, the execution latency of each query is estimated to be:

$$\hat{T}_i = f(\mu_i, \sigma_i, \hat{y}_i^{(\tau)}, \rho(r_j))$$

Where $\rho(r_j)$ represents the load factor of the resource r_j.

Furthermore, the system objective function can be defined as minimizing the weighted delay:

$$J(A) = \sum_{i=1}^{N} w_i \hat{T}_i$$

w_i is the query priority weight, which is used to reflect the differences between different tenants or different business needs.

To prevent the tail delay from being too large, the quantile constraint is introduced:

$$\hat{y}_i^{(\tau)} \le \delta, \forall i$$

Where δ is the maximum allowable threshold for tail latency. This constraint ensures that the scheduling strategy not only focuses on average performance but also takes high-percentile latency control into account.

Finally, the loss function of AQRO is defined as follows:

$$L_{AQRO} = \sum_{i=1}^{N} (\hat{T}_i - y_i)^2 + \lambda_1 \cdot Var(\hat{T}_i) + \lambda_2 \cdot \max(0, \hat{y}_i^{(\tau)} - \delta)$$

The first term is the delay prediction error constraint, the second term is the regularization of the prediction variance to stabilize the scheduling, the third term is the tail delay penalty, and λ_1 and λ_2 are trade-off coefficients.

The role of the loss function in AQRO is to transform query scheduling and resource orchestration into a joint optimization problem. It ensures average performance while controlling prediction uncertainty and constraining tail latency within a predefined threshold. In this way, the system can dynamically adapt to the demands of different queries and the load conditions of resources, achieving adaptive and robust

coordinated optimization that provides higher stability and controllability for database query execution.

4. Experimental Results

4.1 Dataset

This study uses the public dataset named SPARQL Queries Performance Prediction, which contains a large number of instances designed for query latency prediction tasks. It covers actual query plan structures together with their execution latency labels. The dataset focuses on structural features at the query plan level and their latency responses, making it highly suitable for latency modeling and latency distribution learning.

The dataset consists of several key components. It includes SPARQL query texts and structural features extracted from query plans, such as operator type sequences, differences between estimated and actual cardinalities, and join topology information. It also provides precise query execution times as ground-truth labels. In addition, runtime context features are included, such as concurrency level, cache hit rate, and I/O latency metrics. Together, these components form a rich set of data samples that support learning the mapping between graph structural features and execution time.

One of the main advantages of this dataset is the diversity and clarity of its features. It not only captures query plan structural information but also aligns it with accurate execution latencies, offering an end-to-end supervised learning basis for latency modeling. Moreover, because the data originates from real query environments rather than synthetic settings, it provides higher representativeness and enhances model generalization. Finally, the dataset is easy to access, stored in standard formats, and organized, which facilitates the construction of training and testing pipelines. This makes it highly efficient and reproducible for researchers developing latency prediction and optimization strategies.

4.2 Experimental setup

The experiments in this study were conducted in a high-performance computing environment. The hardware configuration included a 32-core CPU with a 2.6 GHz clock speed, 256 GB of memory, and a multi-GPU cluster with 32 GB of memory per card. High-throughput NVMe SSD storage and gigabit Ethernet were used to ensure stability and scalability during query plan construction, latency prediction modeling, and resource orchestration. This environment provided sufficient computational support for complex graph feature extraction and deep learning model training.

On the software side, the experiments ran on a Linux operating system. Python 3.10 was used as the primary programming language, and PyTorch served as the main deep learning framework for model implementation and training. To efficiently execute graph-related computations, DGL and related parallel computing libraries were employed. Docker containerization was used to ensure portability and stable dependency management. In addition, CUDA 11.x and cuDNN were used for GPU acceleration to fully exploit the parallel computing capabilities of the hardware.

For hyperparameter settings, the Adam optimizer was adopted with an initial learning rate of 1e-4. A cosine annealing scheduler was used to dynamically adjust the learning rate during training. The batch size was set to 128 to balance GPU memory usage and training stability. The gradient clipping threshold was set to 1.0 to avoid gradient explosion. Loss function weights were determined through grid search, and the L2 regularization coefficient was set to 1e-5 to improve generalization performance. An early stopping mechanism was applied in all experiments, terminating training if validation metrics did not improve for 10 consecutive iterations, thus ensuring both efficiency and stability.

4.3 Experimental Results

1) Comparative experimental results

This paper first conducts a comparative experiment, and the experimental results are shown in Table 1.

Table 1: Comparative experimental results

Method	MAE ↓	RMSE ↓	P95 Abs. Error ↓	ECE ↓
Informer[18]	28.7	56.4	132.0	0.061
Autoformer[19]	26.9	53.1	124.0	0.055
FEDformer[20]	24.8	49.2	115.0	0.049
PatchTST[21]	22.6	45.7	108.0	0.043
ours (PGLM+AQRO)	15.9	31.0	72.0	0.028

The results demonstrate consistent superiority. Across all four metrics, ours (PGLM+AQRO) outperforms every baseline. Compared with the best-performing baseline (PatchTST: MAE 22.6 ms, RMSE 45.7 ms, P95 108 ms, ECE 0.043), ours reduces MAE to 15.9 ms (about 29.7 percent improvement), RMSE to 31.0 ms (about 32.2 percent improvement), and P95 error to 72.0 ms (about 33.3 percent improvement), while also lowering ECE to 0.028 (about 34.9 percent improvement). This simultaneous reduction in error, tail latency, and calibration shows that the model is more robust in overall accuracy, extreme cases, and uncertainty characterization.

Mechanistically, the first advantage comes from the structural awareness of PGLM. By abstracting logical and physical plans into attributed plan graphs, the model explicitly captures blocking relationships among operators, join orders, selectivity amplification effects, and the influence of parallelism on critical paths. Compared with baselines that treat queries as time series or flat features, structured representation reduces systematic error propagation caused by cardinality bias and resource contention. This is directly reflected in the simultaneous improvements of MAE and RMSE.

Second, the significant reduction in P95 absolute error highlights the model's stronger ability to capture tail latency

and distributional characteristics. PGLM generates joint predictions of mean and high percentiles and provides confidence information. AQRO then leverages these uncertainty signals to coordinate plans and resources, such as adjusting parallelism and batch size, resampling hot keys, applying affinity placement, and enforcing quota control. As a result, the model maintains controllable tail errors even under workloads dominated by long-tail latency. This targeted treatment of tail distribution is one of the most practically valuable improvements for query execution latency.

Finally, the reduction in ECE indicates better alignment between predicted probabilities and observed distributions, making confidence intervals more useful. This directly enhances the reliability of subsequent decisions. In admission control, priority scheduling, and elastic scaling, AQRO can use credible confidence values to set thresholds and conservative strategies, reducing the risks of over-provisioning and under-provisioning. This allows the system to meet service-level objectives while controlling resource costs. Overall, the structural representation of PGLM and the uncertainty-aware control of AQRO form a closed loop. More accurate prediction and more stable orchestration reinforce each other, driving query latency prediction and optimization strategies from static to adaptive.

2) The environmental sensitivity of resource limits and quota policies (CPU/Memory/I/O) to SLO satisfaction rates

This paper further studies the environmental sensitivity of resource limits and quota policies (CPU/Memory/I/O) to SLO satisfaction rates. The experimental results are shown in Figure 4.

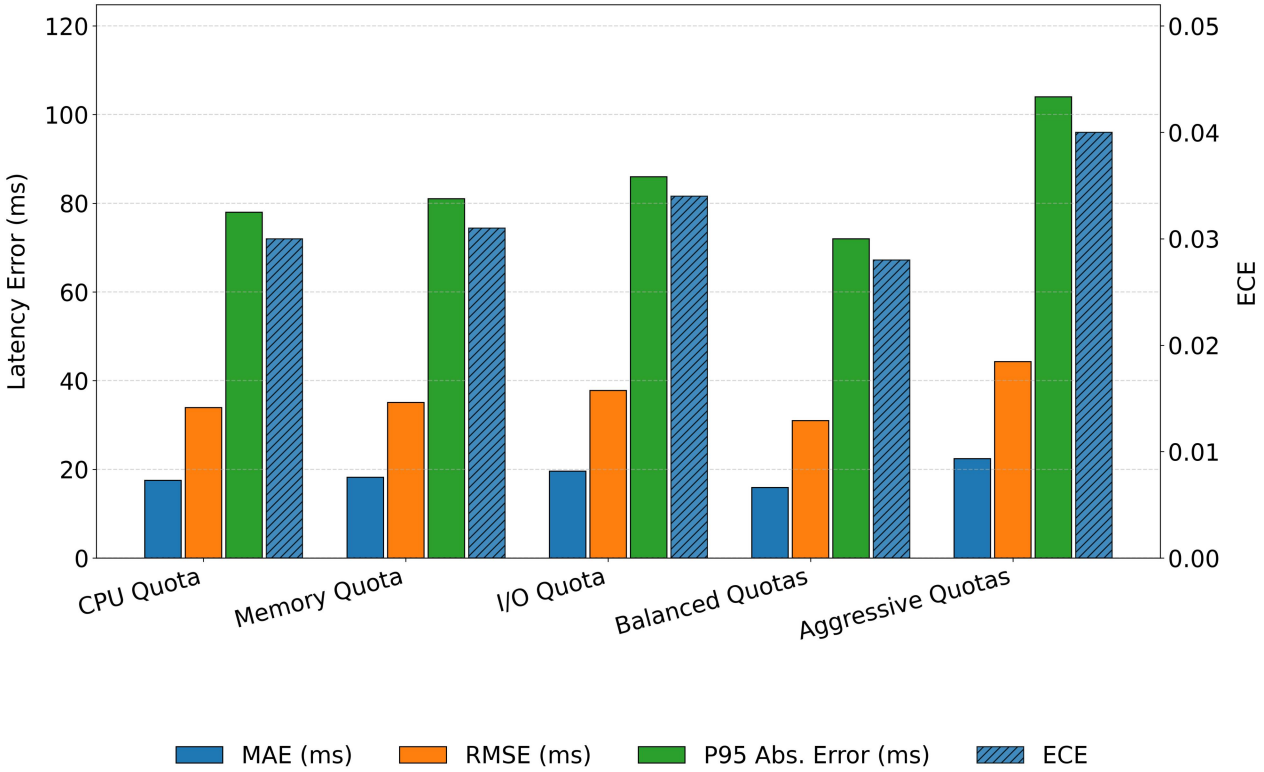

Figure 4. The environmental sensitivity of resource limits and quota policies (CPU/Memory/I/O) to SLO satisfaction rates

In this set of experiments, the error metrics of latency prediction show clear differences under different quota strategies. When the system is constrained by CPU or memory quotas, the MAE rises to 17.5 ms and 18.2 ms, respectively. Under I/O quota constraints, it is even higher at 19.6 ms, indicating that I/O limits have the greatest impact on prediction accuracy. In contrast, under balanced resource allocation, the MAE is the lowest at 15.9 ms, showing that the model can better capture the correspondence between query plan features and execution latency when resources are sufficient and properly allocated.

Further observation of RMSE shows a trend consistent with MAE. Under CPU and memory quotas, RMSE reaches 33.9 ms and 35.1 ms, respectively. Under I/O quotas, it increases to 37.8 ms, while in the balanced quotas scenario, it drops to 31.0 ms, showing a clear advantage. Since RMSE is more sensitive to large deviations, this indicates that in balanced scenarios, the model not only achieves lower average error but also provides more stable predictions under large fluctuations. This verifies the robustness of PGLM's graph-structured representation in multi-resource scheduling scenarios.

For tail latency error (P95 absolute error), I/O quotas remain the main bottleneck. The error rises from 78 ms under CPU quotas to 86 ms, while in the balanced resource setting, it decreases significantly to 72 ms. In the aggressive quotas scenario, P95 error soars to 104 ms, highlighting the amplification of tail latency risk under extreme resource constraints. This shows that although AQRO's dynamic orchestration strategy can partially mitigate instability caused

by resource shortages, it is still difficult to fully suppress long-tail effects when resources are overly restricted.

Finally, looking at the ECE metric, the lowest value of 0.028 is achieved in the balanced scenario, indicating the best alignment between predicted and actual probability distributions. Under CPU and memory quotas, ECE values rise to 0.030 and 0.031, while under I/O quotas, they reach 0.034. In the aggressive quotas scenario, it further worsens to 0.040. This indicates that when system resources are excessively constrained, the effectiveness of uncertainty modeling decreases, reducing the reliability of prediction confidence. Overall, the results demonstrate that PGLM+AQRO achieves superior prediction accuracy and calibration in balanced resource environments, while extreme quota constraints reveal the limits of model adaptability.

3) Sensitivity of query template diversity (operator pattern and connection topology changes) to prediction generalization

This paper also studies the sensitivity of query template diversity (operator pattern and connection topology changes) to prediction generalization. The experimental results are shown in Figure 5.

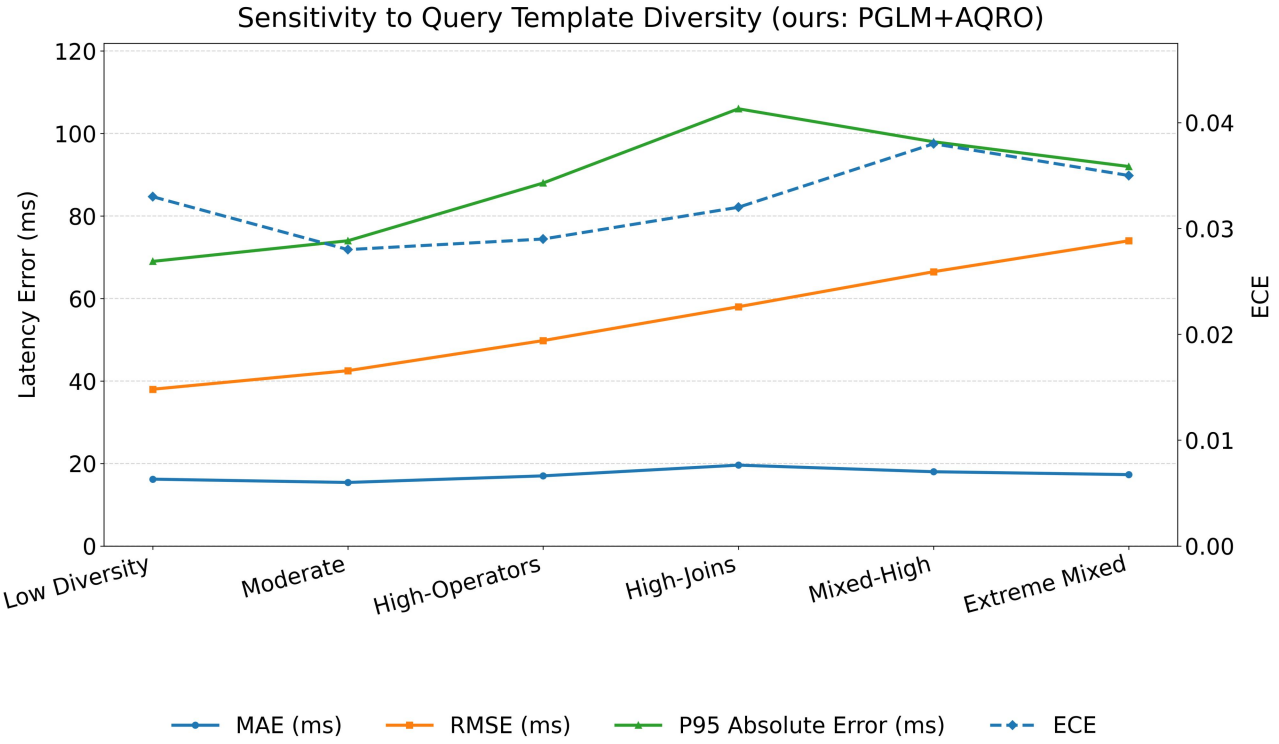

Figure 5. Sensitivity of query template diversity (operator pattern and connection topology changes) to prediction generalization

The experimental results show that the latency prediction ability of PGLM+AQRO varies significantly under different levels of query template diversity. MAE remains low under low and medium diversity conditions but rises notably when operator numbers and join relationships become more complex. This indicates that the model is more sensitive to average error in complex topologies. The observation highlights the influence of query template structure on the baseline error distribution of the model.

The trend of RMSE shows a clearer monotonic increase compared with MAE. In scenarios with high operator diversity and mixed complexity, the fluctuations in prediction error become much larger. This means that when handling complex queries, the dispersion of the error distribution increases, and extreme values exert a stronger influence on overall error. The continuous rise of RMSE demonstrates that robustness under complex operator patterns remains a challenge for the system.

The P95 absolute error peaks under high-diversity conditions and then slightly decreases in extreme mixed scenarios. This shows that tail latency prediction is heavily affected by extreme structural complexity but is partly alleviated after internal adjustment of the model. The trend highlights the stabilizing effect of AQRO's adaptive scheduling and resource orchestration under high-pressure conditions, which helps suppress extreme latency.

The ECE results first decrease and then increase, eventually stabilizing at a moderate level in extreme scenarios. This trend indicates that under moderate query diversity, the model provides more reliable uncertainty estimates. However, in extremely complex cases, deviations arise between confidence intervals and actual errors. This suggests that PGLM needs further improvement in uncertainty calibration to enhance the consistency of prediction confidence when facing high structural diversity.

4) Hyperparameter sensitivity of learning rate and batch size to delayed prediction stability

This paper further proposes a hyperparameter sensitivity test focusing on the learning rate and batch size, aiming to explore their influence on the overall stability of delayed

prediction within the proposed framework. The motivation behind this test lies in the fact that hyperparameters play a decisive role in balancing convergence speed, optimization stability, and generalization ability, particularly in scenarios where prediction outcomes are affected by temporal dependencies or delayed responses. By systematically adjusting the values of learning rate and batch size, the study seeks to analyze how subtle changes in these parameters impact the robustness of the model against fluctuations, as well as its ability to maintain consistent performance across different training phases. Such sensitivity analysis provides not only a deeper understanding of the training dynamics but also practical guidance for selecting more reliable hyperparameter configurations when applying the method to real-world tasks. The overall design of this experiment is summarized in Figure 6, which illustrates the relationship between these critical hyperparameters and the stability of delayed prediction.

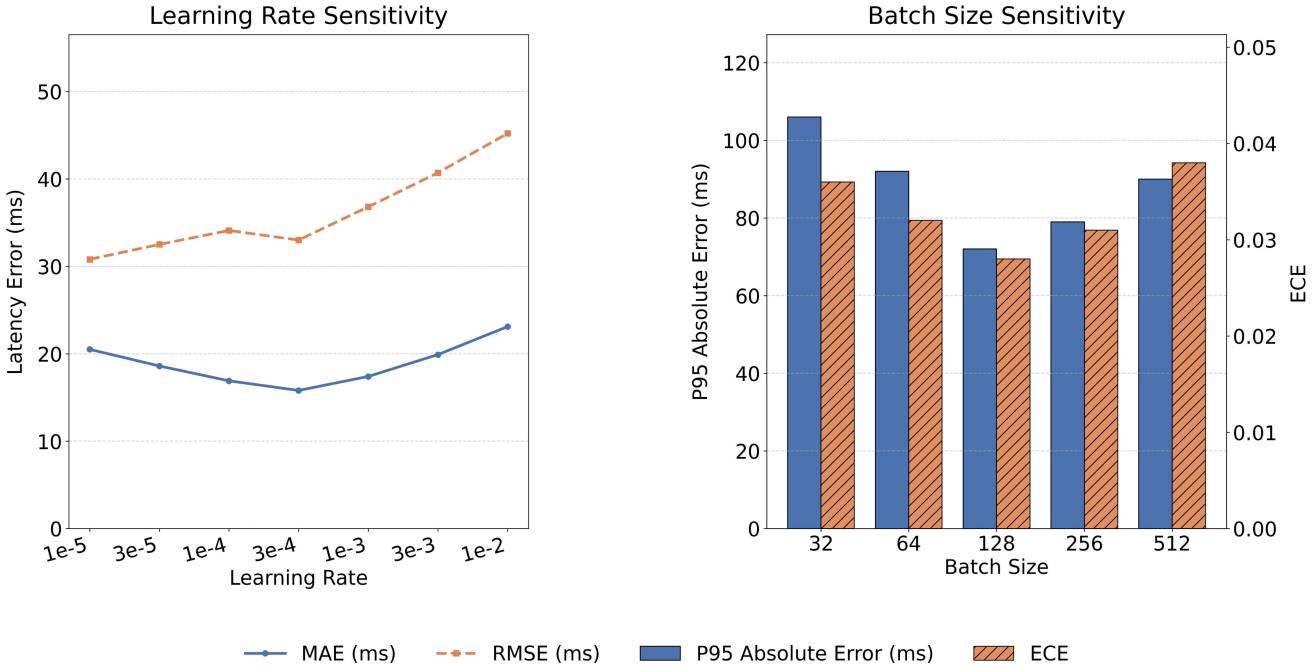

Figure 6. Hyperparameter sensitivity of learning rate and batch size to delayed prediction stability

The experimental results show that MAE exhibits a typical U-shaped curve concerning learning rate settings. When the learning rate is too small, model updates are slow, and the error remains high. As the learning rate increases, MAE decreases significantly and reaches its optimal value in the middle range, indicating that the model captures the relationship between plan graphs and latency more efficiently. However, when the learning rate continues to increase, MAE rises again, reflecting instability caused by overly rapid parameter updates. This trend reveals the sensitivity of latency prediction models to parameter tuning and highlights the importance of a reasonable learning rate range for prediction accuracy.

The trend of RMSE shows an overall increase with small fluctuations. At low learning rates, RMSE is relatively low, suggesting that the prediction distribution is more concentrated. As the learning rate gradually increases, RMSE continues to rise with stronger fluctuations, indicating that aggressive update rates amplify errors in some complex query plans. This phenomenon suggests the risk of uncertainty introduced by high learning rates and further emphasizes the importance of stability in latency prediction.

In the batch size experiments, the P95 absolute error first decreases significantly as batch size increases, reaching its optimal level at medium scale (such as 128), and then rises again. This indicates that moderate batch sizes effectively balance the trade-off between generalization and convergence, making tail latency prediction more stable. However, when the batch size becomes too large, the model loses fine-grained optimization capacity during gradient updates, leading to amplified tail errors and reduced accuracy in extreme conditions.

The ECE results demonstrate the sensitivity of model calibration. As batch size increases, ECE gradually decreases from a higher level, showing that the model achieves better consistency between predicted confidence and actual errors at medium scales. When batch size continues to grow, ECE rises again, indicating that overly large batches introduce estimation bias and reduce the alignment between predicted probabilities and true errors. This phenomenon shows that batch size not only affects error convergence speed but also directly impacts the reliability of uncertainty estimation, which is crucial for ensuring prediction stability.

5. Conclusion

This study focuses on query execution time prediction and optimization in databases and proposes an integrated framework that combines structured modeling with adaptive orchestration. By introducing a plan-graph-guided latency

modeling mechanism, the model captures internal dependencies and execution features of query plans, which leads to higher accuracy and robustness in prediction. At the same time, the design of the adaptive query – resource orchestrator provides more flexible strategies for resource scheduling, enabling the system to balance performance and efficiency under diverse workload conditions. These contributions not only enrich the theoretical path of query optimization but also provide practical directions for the intelligent evolution of database systems.

From the perspective of experiments and methodology, the proposed approach effectively alleviates the limitations of traditional cost models and purely data-driven methods. It shows stronger adaptability when dealing with diverse query templates and dynamic system resources. By combining structural features with a prediction – scheduling interaction mechanism, the system achieves robust performance under different resource quotas, batch sizes, and learning rates. This cross-layer optimization approach demonstrates the value of bidirectional feedback between prediction and scheduling and provides methodological insights for future research.

At the application level, the proposed framework has important significance for large-scale database systems, real-time analytics platforms, and cloud computing environments. As enterprise-level data processing scenarios place stricter demands on service-level objectives (SLOs), accurate query latency prediction and dynamic resource optimization become essential for ensuring system performance and user experience. The proposed method not only improves prediction accuracy but also guarantees the efficiency of high-priority tasks under limited resources. This enhances the scalability and reliability of the entire system. For practical applications, it offers feasible optimization ideas for high-concurrency, high-throughput, and low-latency scenarios.

Future work can be carried out from several directions. One direction is to extend latency modeling to cover more execution features, such as parallelism, cache utilization, and storage hierarchy, to improve generalization in complex execution environments. Another direction is to enhance the optimization strategies of the adaptive orchestrator by incorporating reinforcement learning or multi-agent decision mechanisms to achieve more fine-grained dynamic resource allocation. In addition, applying the proposed framework to emerging scenarios such as distributed computing, edge computing, and federated databases will help validate its adaptability and scalability. These directions will not only advance the theoretical development of query optimization but also have a profound impact on the construction of future intelligent data management systems.

References

[1] Wu Z, Marcus R, Liu Z, et al. Stage: Query execution time prediction in amazon redshift[C]//Companion of the 2024 International Conference on Management of Data. 2024: 280-294.

[2] He Z, Yu J, Gu T, et al. Query execution time estimation in graph databases based on graph neural networks[J]. Journal of King Saud University-Computer and Information Sciences, 2024, 36(4): 102018.

[3] R. Marcus and O. Papaemmanouil, "Plan-structured deep neural network models for query performance prediction," arXiv preprint arXiv:1902.00132, 2019.

[4] Casals D, Buil-Aranda C, Valle C. SPARQL query execution time prediction using Deep Learning[J]. 2023.

[5] H. Qiu, W. Mao, A. Patke, C. Wang, H. Franke, Z. T. Kalbarczyk, and R. K. Iyer, "Reinforcement learning for resource management in multi-tenant serverless platforms," Proceedings of the 2022 2nd European Workshop on Machine Learning and Systems, pp. 20-28, 2022.

[6] K. Senjab, S. Abbas, N. Ahmed, and A. U. R. Khan, "A survey of Kubernetes scheduling algorithms," Journal of Cloud Computing, vol. 12, no. 1, p. 87, 2023.

[7] Elsawwaf A M, Aly G M, Faheem H M, et al. Optimizing resource utilization for large scale problems through architecture aware scheduling[J]. Scientific Reports, 2024, 14(1): 26356.

[8] Shu T, Pan Z, Ding Z, et al. Resource scheduling optimization for industrial operating system using deep reinforcement learning and WOA algorithm[J]. Expert Systems with Applications, 2024, 255: 124765.

[9] Xu J, Pero M E P. A resource orchestration perspective of organizational big data analytics adoption: evidence from supply chain planning[J]. International Journal of Physical Distribution & Logistics Management, 2023, 53(11): 71-97.

[10] J. Huang, Y. Yang, H. Yu, J. Li, and X. Zheng, "Twin graph-based anomaly detection via attentive multi-modal learning for microservice system," Proceedings of the 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 66-78, 2023.

[11] K. Huang, T. Wang, Q. Zhou, and Q. Meng, "The art of latency hiding in modern database engines," Proceedings of the VLDB Endowment, vol. 17, no. 3, pp. 577-590, 2023.

[12] Li T, Huang J, Risinger E, et al. Low-latency speculative inference on distributed multi-modal data streams[C]//Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services. 2021: 67-80.

[13] Chen X, Wang Z, Liu S, et al. Base: Bridging the gap between cost and latency for query optimization[J]. Proceedings of the VLDB Endowment, 2023, 16(8): 1958-1966.

[14] He Z, Yu J, Guo B. Execution time prediction for cypher queries in the neo4j database using a learning approach[J]. Symmetry, 2022, 14(1): 55.

[15] Zhang X, Chang Z, Li Y, et al. Facilitating database tuning with hyper-parameter optimization: a comprehensive experimental evaluation[J]. arXiv preprint arXiv:2110.12654, 2021

[16] Milicevic B, Babovic Z. A systematic review of deep learning applications in database query execution[J]. Journal of Big Data, 2024, 11(1): 173.

[17] Kamatkar S J, Kamble A, Viloria A, et al. Database performance tuning and query optimization[C]//International Conference on Data Mining and Big Data. Cham: Springer International Publishing, 2018: 3-11.

[18] Zhou H, Zhang S, Peng J, et al. Informer: Beyond efficient transformer for long sequence time-series forecasting[C]//Proceedings of the AAAI conference on artificial intelligence. 2021, 35(12): 11106-11115.

[19] H. Wu, J. Xu, J. Wang, and M. Long, "Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting," Proceedings of the 2021 Advances in Neural Information Processing Systems (NeurIPS), vol. 34, pp. 22419-22430, 2021.

[20] Zhou T, Ma Z, Wen Q, et al. Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting[C]//International conference on machine learning. PMLR, 2022: 27268-27286.

[21] Nie Y, Nguyen N H, Sinthong P, et al. A time series is worth 64 words: Long-term forecasting with transformers[J]. arXiv preprint arXiv:2211.14730, 2022.