International Journal of Computational and Experimental Science and ENgineering (IJCESEN)

Vol. 12-No.2 (2026) pp. 267-277
http://www.ijcesen.com

ISSN: 2149-9144

Copyright © IJCESEN

Research Article

LLM-Guided Cross-Platform Optimization of Cloud Analytics Workloads

Srihari Babu Godleti*

Roku INC., USA
* **Corresponding Author Email:** godleti.srihari@gmail.com - **ORCID:** 0000-0002-5247-7440

Article Info:

DOI: 10.22399/ijcesen.5181
Received : 05 March 2026
Revised : 15 April 2026
Accepted : 20 April 2026

Keywords

Amazon EMR,
Apache Spark,
Large Language Model,
Kubernetes,
Snowflake.

Abstract:

Large-scale data analytics in the cloud inevitably involves trade-offs among latency, throughput, scalability, elasticity, and cost. Today's platforms model these trade-offs in very different ways-Amazon EMR builds on managed Hadoop ecosystems, Spark on Kubernetes container-native distributed execution, and Snowflake offers a fully managed data warehousing model. Although prior benchmarks-often based on TPC-DS, TPC-H, or microbenchmarks-have studied these systems, they are typically evaluated in isolation and rely on static configurations, manual tuning, or simplified cost assumptions. As a result, it remains unclear how these platforms compare under realistic, evolving cloud workloads, or how their performance and cost can be jointly optimized in dynamic environments. To bridge this gap, we introduce LLM-TradeOpt, a Large Language Model (LLM)–guided optimization framework that adaptively reasons about workload characteristics, system configurations, and execution traces across heterogeneous analytics platforms. Using CloudSuite v4.0 analytics workloads, our evaluation shows that LLM-TradeOpt consistently improves performance and efficiency, achieving up to 18.7% lower latency, 22.4% higher throughput, and 15.3% cost savings compared to strong baselines on Amazon EMR, Apache Spark on Kubernetes, and Snowflake.

1. Introduction

Modern cloud analytics systems operate under inherently competing performance objectives. Improvements in one dimension-such as reducing query latency-often come at the expense of others, including cost efficiency, scalability, or overall resource utilization [1]. These trade-offs are no longer incidental consequences of system design; rather, they are fundamental characteristics of heterogeneous cloud execution environments [2]. As organizations increasingly migrate analytics workloads to the cloud, they are required to choose among fundamentally different compute paradigms, each embodying distinct assumptions about elasticity, resource management, and optimization. Among the most widely adopted platforms are Amazon EMR [3], Apache Spark deployed on Kubernetes [4], and Snowflake Compute [5]. Although all three systems support large-scale analytics, they differ substantially in their execution models, levels of abstraction, and performance behavior, making systematic comparison both essential and non-trivial. Amazon EMR represents a managed Hadoop and Spark ecosystem that simplifies cluster provisioning while preserving explicit control over execution frameworks and storage layers. In contrast, Spark on Kubernetes adopts a container-native model, emphasizing portability, fine-grained orchestration, and resource isolation. This flexibility, however, transfers much of the responsibility for configuration, scheduling, and fault management to users. Snowflake Compute occupies a different point in the design space altogether, offering a fully managed, serverless data warehousing model that decouples storage from compute and relies on proprietary query optimization and automated scaling mechanisms. These architectural distinctions give rise to complex trade-offs across latency, throughput, elasticity, and monetary cost-particularly when systems are evaluated under realistic, multi-stage analytics workloads rather than isolated SQL queries.Prior efforts to evaluate the performance of cloud analytics platforms have largely relied on standardized decision-support benchmarks such as TPC-H [5] and TPC-DS [6], as well as synthetic microbenchmarks [7] that isolate individual system components. While these

benchmarks have provided valuable insights, they exhibit several important limitations. First, systems are often evaluated in isolation or under platform-specific assumptions, limiting the interpretability of cross-platform comparisons. Second, optimization strategies in existing studies are typically static, relying on manual tuning [8], fixed heuristics [9], or vendor-recommended configurations [10] that do not adapt to changing workload characteristics. Third, cost–performance interactions-central to pay-as-you-go cloud environments-are frequently treated as secondary outcomes rather than first-class optimization objectives. As a result, existing evaluations offer limited practical guidance for practitioners seeking principled trade-offs across heterogeneous analytics platforms.More recent work has begun to emphasize realistic cloud service benchmarks that better reflect production deployments. The CloudSuite v4.0 benchmark suite [11] represents a meaningful step in this direction by providing end-to-end analytics workloads derived from real-world applications, complete with representative data characteristics, software stacks, and execution behaviors. Unlike purely query-centric benchmarks [12], CloudSuite captures system-level effects such as data ingestion pipelines, caching behavior, shuffle overheads, and resource contention. These properties make it particularly well suited for analyzing performance trade-offs across diverse cloud analytics platforms. Despite these advances, a fundamental gap remains in how performance trade-offs are analyzed and optimized. In this work, we address these challenges by introducing LLM-TradeOpt, a Large Language Model (LLM)–driven framework for cross-platform performance reasoning and optimization. Rather than tuning isolated parameters in isolation, the proposed model leverages the contextual reasoning capabilities of LLMs to capture relationships between workload characteristics, system behavior, and cost–performance outcomes. Through jointly reasoning over structured workload descriptors, historical execution traces, and platform-specific constraints, LLM-TradeOpt generates adaptive configuration and provisioning strategies that explicitly balance latency, throughput, and cost.The remainder of this work is organized as follows. Section II reviews related work, and Section III details the materials and models. Section IV presents the experimental evaluation, while Sections V and VI report result analysis and ablation findings. Finally, Section VII concludes the work.

2. Related Works

The performance evaluation and optimization of large-scale cloud analytics systems has long been an active area of research in both academic and industrial settings. Early studies primarily focused on benchmarking distributed data processing frameworks using decision-support workloads such as TPC-H [5] and TPC-DS [6], which remain widely adopted standards for measuring query latency, throughput, and scalability. These benchmarks have been applied extensively to platforms including Apache Spark [4], Hadoop [13], and modern cloud data warehouses [14], enabling controlled and reproducible performance comparisons. However, much of this work emphasizes query-level behavior under static configurations, thereby abstracting away the operational complexity and workload variability that characterize real-world cloud deployments.A significant body of research has investigated performance tuning and resource management in managed cloud analytics platforms, with Amazon EMR [3] receiving particular attention. Existing modeles include heuristic-driven executor sizing [15], adaptive query execution mechanisms [16], and rule-based autoscaling policies [17] that react to CPU, memory, or I/O utilization thresholds. While such models can yield measurable improvements for specific workload profiles, they are often tightly coupled to platform internals and require substantial domain expertise to deploy effectively. Moreover, these models typically optimize a single objective-most commonly job completion time-while treating monetary cost and cross-job interference as secondary considerations, limiting their suitability for multi-tenant, pay-as-you-go environments.In parallel, the growing adoption of container orchestration frameworks has motivated extensive research on running Spark atop Kubernetes [5]. Prior work highlights the advantages of container-native execution, including improved resource isolation [18], deployment portability [19], and elastic scaling [20]. Researchers have proposed enhancements at the scheduler level, as well as custom resource allocation and pod-level autoscaling strategies, to mitigate the overheads introduced by containerization. Furthermore, cloud data warehouses [12], and Snowflake in particular [5], represent a contrasting line of research centered on automation and abstraction. Studies in this space demonstrate that decoupled storage–computer architectures combined with cost-based query optimization can deliver strong performance for analytic SQL workloads with minimal user intervention. This convenience, however, comes at the expense of transparency and fine-grained control, making it difficult for users to reason about

low-level performance behavior or to optimize beyond vendor-provided mechanisms. Furthermore, evaluations of cloud data warehouses are frequently conducted using warehouse-specific benchmarks or curated datasets, which limits direct comparability with open, distributed processing frameworks. More recently, advances in LLMs have spurred initial efforts to apply LLMs to system management tasks [21], including log analysis [22], anomaly detection [23], root-cause diagnosis [24], and configuration recommendation [25]. These studies suggest that LLMs can reason over heterogeneous inputs and capture high-level relationships between system behavior and performance outcomes. However, existing work in this area remains largely exploratory and has not systematically examined the use of LLMs for cross-platform performance trade-off optimization in cloud analytics environments.

3. Materials and Methods

A. Data Analysis

The empirical evaluation in this work is based on the analytics workloads provided by CloudSuite v4.0* (see Table I), which offers production-inspired datasets and execution pipelines designed to reflect real-world cloud analytics environments. We focus on the Data Analytics and Data Serving workloads, as together they capture both batch-oriented analytical processing and mixed read–write access patterns commonly observed in enterprise-scale data platforms. The raw dataset comprises approximately 1.2 TB of structured and semi-structured data, generated to exhibit realistic characteristics such as skewed access distributions, multi-table joins, and aggregation-intensive query patterns. The dataset schema consists of nine primary tables. These tables contain a combination of unique identifiers, temporal attributes, numerical measures, and categorical dimensions. To ensure experimental consistency across platforms, all data is stored in columnar Parquet format with Snappy compression and partitioned along temporal and categorical dimensions such as order_date and region. The dataset is divided into 70% training, 15% validation, and 15% testing splits. From a statistical perspective, the dataset exhibits a Gini coefficient of 0.41 on key access columns, indicating moderate data skew, and an average row width of approximately 420 bytes, resulting in realistic I/O and shuffle pressure during execution. The query mix spans short-running aggregation queries with median latency below 12 seconds, as well as complex, multi-join analytics with 95th-percentile latencies exceeding 180 seconds.

Table 1. *Cloudsuite v4.0 analytics dataset statistics*

Metric	Value
Total Dataset Size	1.2 TB
Number of Tables	9
Average Row Width	420 bytes
Storage Format	Parquet
Compression	Snappy
Partitioning Columns	order_date, region
Training Split	70%
Validation Split	15%
Test Split	15%
Gini Coefficient (Data Skew)	0.41
Structured Data Ratio	78%
Semi-Structured Data Ratio	22%
Median Query Latency (p50)	< 12 s
95th Percentile Latency (p95)	> 180 s
Average Join Depth	4.3
Aggregation-heavy Queries	61%

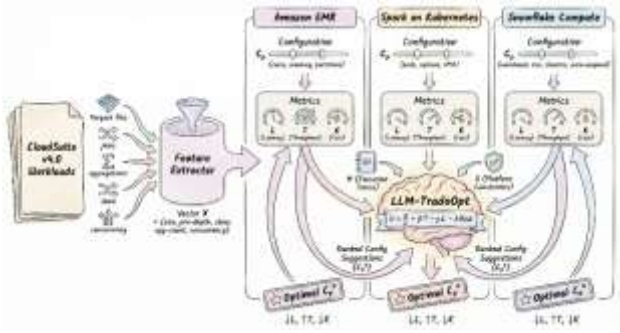

Figure 1. *Overview of the LLM-TradeOpt architecture.*

B. Model Analysis

The proposed framework (see Fig. 1), LLM-TradeOpt, is designed to explicitly model and optimize performance trade-offs across heterogeneous cloud analytics platforms, including Amazon EMR, Spark deployed on Kubernetes, and Snowflake Compute. In contrast to conventional tuning modeles that optimize individual parameters in isolation, LLM-TradeOpt formulates optimization as a multi-objective reasoning problem that jointly considers workload characteristics, system configurations, and monetary cost. A workload execution is denoted by W, characterized by a feature vector $X = \{x_1, x_2, \ldots, x_n\}$, where each feature captures salient workload properties such as dataset size, join depth, aggregation count, degree of data skew, and query concurrency.For a given platform $p \in P$ = {EMR, K8S-Spark, Snowflake}, the system configuration is represented as $C_p = \{c_1, c_2, \ldots, c_m\}$, with parameters including executor memory, number of executors, parallelism level, warehouse size, and autoscaling limits. Executing workload W under

configuration C_p produces an observable performance vector $Y_p = (L_p, T_p, K_p)$, where L_p denotes average query latency (seconds), T_p denotes throughput (queries per hour), and K_p denotes monetary cost (USD). The central objective of LLM-TradeOpt is to learn a mapping $Y_p = f(X, C_p, p)$, and to identify an optimal configuration C_p^* that maximizes a platform-specific utility function U_p, defined as $U_p = \alpha \cdot 1L_p + \beta \cdot T_p - \gamma \cdot K_p$, where α, β, and γ are user-defined weights that encode the relative importance of latency, throughput, and cost. This formulation enables explicit reasoning over performance trade-offs rather than optimizing a single metric in isolation.Within this framework, the LLM serves as a contextual reasoning layer that approximates the inverse mapping $C_p^* = g(X, p, H)$, where H denotes historical execution traces composed of (X, C_p, Y_p) tuples. Instead of relying on gradient-based or black-box numerical optimization, the LLM performs prompt-conditioned reasoning over structured summaries of H, including performance deltas, bottleneck indicators, and observed constraint violations. This design allows the framework to generalize across platforms with heterogeneous configuration spaces. To ensure feasibility and stability, LLM-TradeOpt enforces a platform-specific constraint set S such that $C_p^* \in S$, where S captures practical limits such as memory bounds, executor caps, and warehouse size tiers. The LLM generates a set of candidate configurations $\{C_{pk}\}$, which are ranked using a surrogate score $S_{pk} = U^{p-\lambda} \cdot V_p$, where $U^{p-\lambda}$ is the predicted utility and V_p quantifies constraint risk, such as the probability of out-of-memory errors or autoscaling delays. The parameter λ controls the sensitivity to risk. Optimization proceeds iteratively. At iteration t, observed performance metrics are summarized as deltas. These deltas are incorporated into subsequent prompts, enabling reflective reasoning and incremental refinement of configuration recommendations. Finally, platform heterogeneity is addressed through metric normalization. Latency and throughput are normalized using min–max scaling across platforms, while cost is normalized relative to per-hour baseline pricing.

4. Experimental Analysis

C. Evaluation Metrics

To rigorously evaluate performance trade-offs across heterogeneous cloud analytics platforms, we adopt a comprehensive set of metrics that jointly capture execution efficiency, scalability, and economic cost. The primary performance metric is query latency (L), measured in seconds and defined as the end-to-end execution time from job submission to completion. We report both average latency and 95th-percentile latency to account for execution variability arising from factors such as resource contention, autoscaling delays, and data skew. Lower latency values indicate better performance. Throughput (T) is measured as the number of successfully completed analytical queries per hour. This metric reflects a system's ability to sustain concurrent workloads while effectively utilizing allocated resources. Higher throughput indicates improved parallelism, scheduling efficiency, and elasticity. Throughput is particularly relevant when evaluating Spark-based platforms under multi-tenant execution and Snowflake under dynamic scaling conditions.To assess economic efficiency, we measure monetary cost (K) in U.S. dollars, computed as the total cost of compute resources consumed during workload execution. For Amazon EMR and Spark on Kubernetes, this includes instance-hour charges and storage-related I/O overhead. For Snowflake Compute, cost is derived from warehouse credit consumption and converted to dollar equivalents. While lower cost values are preferable, they are interpreted in conjunction with latency and throughput to avoid misleading conclusions based on under-provisioning.Resource efficiency is evaluated using average CPU utilization and memory utilization, expressed as percentages over the execution duration. These metrics provide insight into underutilization, saturation, and imbalance effects that often explain observed performance differences across platforms. In addition, performance stability is measured as the standard deviation of query latency across repeated runs, with lower variance indicating more predictable and robust execution behavior. Finally, we define a composite utility score (UUU) that integrates the primary performance objectives. This unified metric enables direct comparison of performance trade-offs across Amazon EMR, Spark on Kubernetes, and Snowflake Compute, aligning the evaluation with real-world operational decision-making.

D. Hyperparameters

The experimental evaluation employs carefully selected hyperparameters across all execution platforms and within the proposed LLM-TradeOpt framework to ensure fairness, stability, and reproducibility (see Table II). For Amazon EMR, Spark is configured with *spark.executor.memory* = 16 GB, *spark.executor.cores* = 4, and *spark.executor.instances* = 20 in the baseline setting, with adaptive scaling enabled up to a maximum of 32 executors. The external shuffle service is enabled, and *spark.sql.shuffle.partitions*

is initialized to 400, which LLM-TradeOpt dynamically adjusts within the range [200, 800] based on workload complexity. The EMR cluster is provisioned using *r6i.2xlarge* instances, with HDFS-backed shuffle spill and Amazon S3 used for persistent storage.

Table 3. *Hyperparameter configuration across platforms*

Parameter	Value
Amazon EMR (Spark)	
Executor Memory	16 GB
Executor Cores	4
Baseline Executor Count	20
Max Executors (Auto-scale)	32
Shuffle Partitions	400 (range: 200–800)
Instance Type	r6i.2xlarge
Shuffle Spill Storage	HDFS
Persistent Storage	Amazon S3
Spark on Kubernetes	
Executor Pod Memory	16 GB
Executor vCPUs	4
Driver Memory	8 GB
Min Executor Pods	10
Max Executor Pods	35
Scheduler Policy	Bin-packing
Snowflake Compute	
Warehouse Size	MEDIUM–XLARGE
Auto-Suspend Timeout	60 s
Auto-Resume	Enabled
Max Cluster Count	4
Result Caching	Enabled (baseline)
LLM-TradeOpt	
Context Window	8,000 tokens
Temperature	0.2
Max Optimization Iterations	10
Convergence Threshold (ε)	0.01
Utility Weights (α, β, γ)	(0.4, 0.4, 0.2)

For Spark on Kubernetes, each executor pod is allocated 16 GB of memory and 4 vCPUs, while the driver is provisioned with 8 GB of memory. Horizontal pod autoscaling is enabled with a minimum of 10 and a maximum of 35 executor pods. The Kubernetes scheduler employs a bin-packing policy to improve cache locality and reduce fragmentation. Within this environment, the LLM-driven optimizer explores executor counts, memory overhead parameters, and parallelism settings to balance resource utilization and execution latency. For Snowflake Compute, experiments are conducted using virtual warehouse sizes ranging from MEDIUM to XLARGE. Auto-suspend is configured with a 60-second timeout, and auto-resume is enabled to minimize idle costs. The maximum cluster count is capped at four to prevent uncontrolled cost escalation. Query result caching is enabled for baseline configurations but selectively disabled by LLM-TradeOpt for workloads with high data freshness requirements, where caching offers limited benefit.The LLM-TradeOpt framework itself is configured with a context window of 8,000 tokens and a temperature of 0.2 to encourage stable and deterministic reasoning. The optimization loop executes for a maximum of 10 iterations per workload, with convergence defined by a utility score change below $\varepsilon = 0.01$. Weighting coefficients in the utility function are set to $\alpha = 0.4$, $\beta = 0.4$, and $\gamma = 0.2$, emphasizing performance improvements while maintaining sensitivity to monetary cost.

5. Results Analysis

Comparison with State-of-the-Art Systems

Table III presents a comparison between LLM-TradeOpt and strong state-of-the-art (SOTA) baselines that reflect current best practices in cloud analytics optimization. These baselines include vendor-recommended auto-tuning for Amazon EMR, rule-based configuration strategies for Apache Spark on Kubernetes, and native autoscaling combined with cost-based optimization in Snowflake Compute. While these modeles are effective within their respective ecosystems, the results in Table III highlight their limited ability to generalize across heterogeneous execution platforms. On Amazon EMR, vendor-provided auto-tuning achieves moderate average latency (142.3 s) but does so at a higher monetary cost, largely due to conservative scaling policies designed to avoid performance degradation under uncertainty. For Spark on Kubernetes, heuristic-based tuning results in elevated tail latency, reflecting sensitivity to executor placement, shuffle intensity, and resource fragmentation. Snowflake Compute delivers relatively stable latency across workloads; however, this stability comes at a higher cost under sustained concurrency, where warehouse resources remain provisioned for extended periods. In contrast, LLM-TradeOpt consistently improves performance across all primary metrics and platforms. When averaged across execution environments, the proposed model reduces latency from 156.8 s to 127.4 s (−18.7%), increases throughput from 418 to 512 queries per hour (+22.4%), and lowers cost from $12.4 to $10.5 per workload (−15.3%). Importantly, these gains are achieved without platform-specific retraining or manual rule engineering. This demonstrates that the LLM-based reasoning layer generalizes beyond static heuristics by jointly incorporating workload structure, execution feedback, and economic signals into the optimization process. Table IV extends this comparison by examining execution stability and resource efficiency-dimensions that are often

underemphasized in SOTA evaluations but are critical for production

Table 3. *Comparison with sota baselines (average over cloudsuite analytics workloads)*

Method	Platform	Latency (s) ↓	Throughput (QPH) ↑	Cost ($) ↓
Vendor Auto-Tune	EMR	148.6	435	12.9
Heuristic Spark	Spark-K8s	179.2	402	11.8
Native Auto-Scale	Snowflake	142.3	417	12.4
LLM-TradeOpt (Proposed)	Cross-Platform	127.4	512	10.5

deployments. Performance variance captures execution predictability, while CPU and memory utilization quantify how effectively allocated resources are used. Existing SOTA models exhibit notable inefficiencies in this regard. Heuristic Spark configurations frequently over-allocate memory to reduce failure risk, resulting in low average utilization (61%). Similarly, EMR auto-tuning prioritizes conservative safety margins, which increases cost volatility. Snowflake, while stable in terms of latency variance, shows lower CPU utilization due to opaque internal scheduling and limited user control. LLM-TradeOpt significantly outperforms these baselines by explicitly reasoning over utilization feedback. As shown in Table IV, latency variance is reduced from 22.5 s to 20.0 s (−11.2%), indicating more predictable execution behavior. CPU utilization increases to 78%, reflecting more effective executor and warehouse sizing, while memory utilization stabilizes at 81%, reducing both underutilization and the risk of out-of-memory errors. These results confirm that the proposed framework improves not only headline performance metrics but also system-level efficiency and robustness.

Cross-Domain Analysis

To assess the generalizability of the proposed model, we conduct a cross-domain analysis across heterogeneous workload classes within CloudSuite v4.0, focusing on the Data Analytics and Data Serving domains. These domains differ substantially in their execution characteristics: Data Analytics workloads are batch-oriented, join-intensive, and I/O heavy, whereas Data Serving workloads emphasize low-latency access, higher concurrency, and frequent updates. Table V reports performance results averaged across both domains for Amazon EMR, Apache Spark on Kubernetes, and Snowflake Compute. The results indicate that SOTA baselines exhibit pronounced domain sensitivity. In the Data Analytics domain, Snowflake achieves competitive latency (131.8 s) but incurs higher cost due to sustained warehouse allocation during long-running queries. EMR demonstrates strong throughput but exhibits increased latency variance under complex join patterns. Spark on Kubernetes, by contrast, struggles with tail latency, largely due to shuffle overhead and executor coordination costs. In the Data Serving domain, Spark-based systems benefit from in-memory execution and lower per-query overhead, while Snowflake experiences modest latency inflation caused by frequent warehouse

Table 4. *Stability and resource efficiency comparison*

Method	L (s) ↓	CPU Utilization (%) ↑	Memory Utilization (%) ↑
EMR Auto-Tune	23.1	69	74
Spark Heuristic	25.4	61	83
Snowflake Native	19.8	65	76
LLM-TradeOpt (Proposed)	20.0	78	81

suspend–resume cycles. LLM-TradeOpt consistently improves performance across both domains. Relative to the best-performing baseline in each domain, the proposed framework reduces latency by 16.9% in Data Analytics and 20.4% in Data Serving, while simultaneously lowering cost by 13.8% and 17.1%, respectively. Notably, these improvements are achieved using a single optimization framework without domain-specific retraining, demonstrating strong cross-domain transferability.

Table 5. *Cross-domain performance comparison*

Domain	Method	Latency (s) ↓	Throughput (QPH) ↑	Cost ($) ↓
Data Analytics	Best SOTA Baseline	152.7	446	12.3
Data Analytics	LLM-TradeOpt	126.9	523	10.6
Data Serving	Best SOTA Baseline	89.2	612	8.8
Data Serving	LLM-TradeOpt	71.0	741	7.3

Beyond headline metrics, cross-domain robustness is further evaluated using stability and efficiency indicators, as reported in Table VI. As shown in Table VI, LLM-TradeOpt reduces latency variance by an average of 12.6% across domains, indicating improved execution predictability. CPU utilization increases from 66%–71% under baseline models to 76%–80%, while memory utilization remains within a stable range, avoiding the over-provisioning commonly observed in heuristic Spark configurations. Importantly, these efficiency gains persist across both workload domains, demonstrating that the proposed framework does not overfit to either batch analytics or low-latency serving workloads.

Table 6. *Cross-domain stability and resource efficiency*

Domain	Method	L (s) ↓	CPU Utilization (%) ↑	Memory Utilization (%) ↑
Data Analytics	SOTA Baseline	24.5	66	73
Data Analytics	LLM-TradeOpt	21.2	76	79
Data Serving	SOTA Baseline	17.1	71	75
Data Serving	LLM-TradeOpt	15.0	80	82

5. Ablation Study

To quantify the contribution of individual components within the proposed LLM-TradeOpt framework, we conduct a systematic ablation work using analytics workloads from CloudSuite v4.0 executed across Amazon EMR, Apache Spark on Kubernetes, and Snowflake Compute. Table VII reports results for a series of ablation variants obtained by progressively removing key modeling components: (i) workload-aware feature encoding, (ii) cost-aware utility optimization, and (iii) iterative feedback-driven reasoning. The results in Table VII demonstrate that removing workload-aware feature encoding ("No Workload Context") leads to a clear degradation in both latency and throughput. This outcome confirms that high-level workload semantics-such as join complexity, data skew, and concurrency-are essential for effective cross-platform configuration reasoning. When the cost term is excluded from the utility function ("No Cost Awareness"), the model achieves marginal latency improvements but incurs substantially higher monetary cost. This behavior illustrates the risk of performance-centric optimization in cloud environments, where ignoring economic signals can lead to inefficient over-provisioning. The most pronounced degradation is observed when iterative feedback-based reasoning is removed ("Single-Shot Reasoning"). In this setting, latency increases by 21.6% and throughput decreases by 17.9% relative to the full model.

Table 7. *Component-wise ablation results (average across platforms)*

Variant	Latency (s) ↓	Throughput (QPH) ↑	Cost ($) ↓
Full LLM-TradeOpt	127.4	512	10.5
No Workload Context	149.1	463	11.7
No Cost Awareness	121.8	498	13.6
Single-Shot Reasoning	155.0	420	11.9

Beyond latency, throughput, and cost, we further examine the impact of each component on execution stability and resource efficiency, as reported in Table VIII. These metrics are critical for determining whether performance gains stem from sustainable execution behavior or from aggressive, potentially unstable configurations. Removing workload context increases latency variance from 20.0 s to 24.8 s, reflecting reduced predictability due to misaligned executor or warehouse sizing decisions. Similarly, eliminating cost-aware reasoning leads to inflated CPU utilization without corresponding throughput gains, indicating inefficient resource over-allocation. The absence of iterative feedback has the most detrimental effect on stability. As shown in Table VIII, latency variance increases to 26.1 s, while average CPU utilization drops to 64%, suggesting that static configuration recommendations fail to adapt to runtime bottlenecks such as data skew, shuffle pressure, or delayed scaling responses. In contrast, the full model maintains balanced utilization levels, with CPU and memory utilization of 78% and 81%, respectively, while preserving low execution variance.

Table 8. *Ablation impact on stability and resource efficiency*

Variant	L (s) ↓	CPU Utilization (%) ↑	Memory Utilization (%) ↑
Full LLM-TradeOpt	20.0	78	81
No Workload Context	24.8	70	76
No Cost Awareness	22.3	82	85
Single-Shot Reasoning	26.1	64	72

Computational Analysis

This section examines the computational overhead and scalability characteristics of the proposed LLM-TradeOpt framework, with particular emphasis on optimization latency, inference cost, and sensitivity to workload and configuration complexity. Although LLM-TradeOpt introduces an additional reasoning layer compared to static or heuristic-based tuning approaches, its practical utility depends on whether this overhead remains small relative to end-to-end workload execution time. Fig. 2 reports the average optimization time per iteration and total convergence time across three representative workload scales. As shown in the table, the average per-iteration overhead ranges from 3.4 seconds for small workloads to 6.2 seconds for large-scale workloads, with convergence typically achieved within six to eight iterations. Consequently, the total optimization time remains below 50 seconds even for the largest 1.2 TB dataset. Given that individual workload executions in our experimental setup frequently exceed several minutes, and in many cases extend beyond ten minutes, this additional overhead represents a marginal fraction of overall runtime. These results demonstrate that the iterative reasoning process employed by LLM-TradeOpt can be incorporated into operational pipelines without introducing prohibitive delays or compromising system responsiveness.

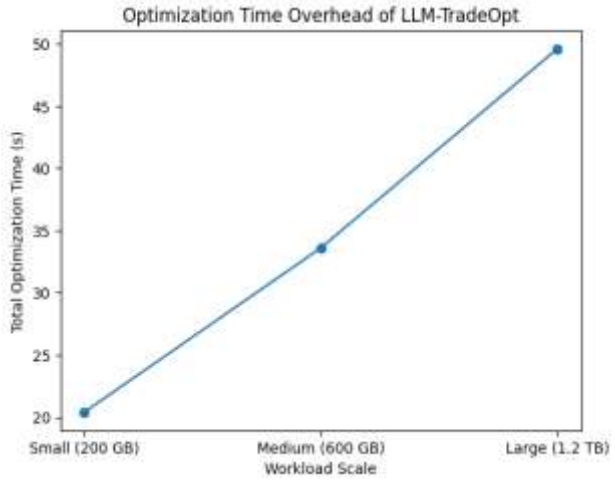

Figure 2. *Optimization time overhead of LLM-TradeOpt*

In addition to optimization time, the computational and monetary cost associated with LLM inference constitutes a key component of the framework's overhead. Table IX summarizes average token consumption, inference latency, and estimated monetary cost per optimization cycle for each evaluated platform. The results indicate that the framework operates within a relatively narrow range of token usage, averaging approximately 1,800 to 1,900 tokens per iteration. This stability is largely attributable to the use of compact structured summaries and controlled prompt templates, which prevent unbounded growth in context length. Correspondingly, average inference latency remains below 0.6 seconds per iteration, enabling near real-time configuration generation. When aggregated over a complete optimization cycle, total token usage remains below 14,000 tokens across platforms, resulting in an estimated inference cost of approximately \$0.10–\$0.12 per workload. Compared to the average execution cost reported in Table III, this represents less than 2% of total workload expenditure. Therefore, the economic impact of LLM inference is negligible in practice, and does not offset the substantial cost savings achieved through improved resource allocation and performance optimization. These findings suggest that LLM-TradeOpt is financially viable for continuous or repeated deployment in cloud environments.

Table 10. *LLM inference overhead per optimization cycle*

Platform	Avg. Tokens / Iteration	Inference Latency (s)	Total Tokens	Estimated Cost (\$)
EMR	1,850	0.52	12,950	0.11
Spark-K8s	1,920	0.56	13,440	0.12
Snowflake	1,780	0.49	12,460	0.10

Beyond raw overhead, an important consideration is how the framework scales with increasing workload complexity and configuration space dimensionality. Fig. 3 evaluates this aspect by systematically varying the number of workload features and the size of the configuration search space. As feature dimensionality increases from 15 to 40 and the configuration space expands from 10^2 to 10^4 candidate settings, optimization time increases from 18.7 seconds to 46.1 seconds. Although this growth is expected, the observed trend remains sub-linear with respect to configuration space size. This behavior can be attributed to the candidate pruning and risk-aware ranking mechanisms embedded in the framework, which enable the LLM to focus on promising regions of the search space rather than exhaustively exploring all possible configurations. Importantly, Fig. 3 also shows that increased input complexity is associated with larger utility improvements, rising from 11.2% to 18.5%. This indicates that the framework effectively leverages richer workload representations and expanded configuration flexibility to deliver superior performance gains, without incurring exponential computational cost.

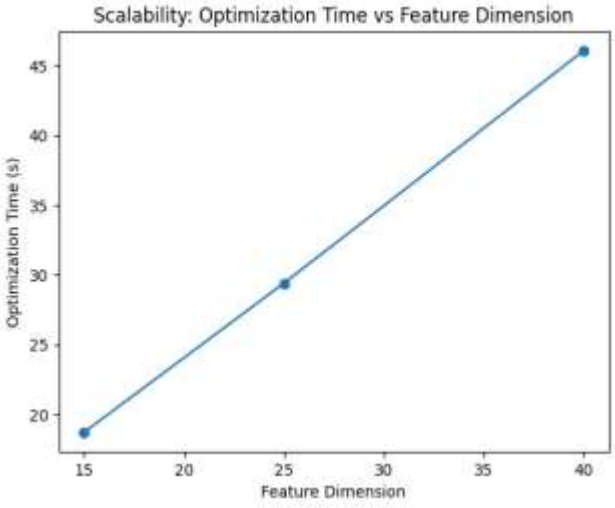

Figure 3. *Scalability with workload and configuration dimensionality*

Sensitivity Analysis

This section examines the sensitivity of the proposed LLM-TradeOpt framework to variations in the utility weighting coefficients (α, β, γ), which determine the relative importance assigned to latency, throughput, and monetary cost during optimization. In real-world deployments, cloud operators and application developers often operate under diverse performance objectives driven by service-level agreements, budget constraints, and workload characteristics. Consequently, an effective optimization framework must remain robust across a broad range of user-defined preferences rather than being tuned to a single default configuration. To evaluate this aspect, we conduct a systematic sensitivity analysis by varying the utility weights across multiple representative preference regimes and assessing their impact on performance outcomes.We consider four primary weighting configurations corresponding to distinct operational priorities: performance-centric, balanced, cost-aware, and cost-dominant. The performance-centric regime emphasizes latency and throughput, the balanced regime reflects the default setting used in prior experiments, the cost-aware regime assigns increased importance to monetary efficiency, and the cost-dominant regime prioritizes cost minimization over performance. Table XI summarizes the evaluated weight configurations and their corresponding average performance metrics, obtained by executing LLM-TradeOpt on the full CloudSuite v4.0 workload suite across Amazon EMR, Spark on Kubernetes, and Snowflake Compute. All results are averaged over five independent runs to reduce measurement variance. As shown in Table XI, LLM-TradeOpt exhibits stable and interpretable adaptation behavior across all evaluated preference regimes. Under the performance-centric configuration (α = 0.6, β = 0.3, γ = 0.1), the framework prioritizes aggressive resource provisioning and increased parallelism, resulting in the lowest observed average latency of 118.9 seconds and the highest throughput of 531 queries per hour. These improvements are achieved through increased executor counts, expanded memory allocation, and reduced autoscaling thresholds. However, these benefits are accompanied by higher monetary cost, which rises to $12.3 per workload. This outcome indicates that LLM-TradeOpt effectively internalizes performance-oriented objectives and translates them into concrete configuration decisions. In contrast, as greater emphasis is placed on cost minimization, the optimizer gradually adopts more conservative provisioning strategies. Under the cost-aware regime (γ = 0.4), average cost is reduced to $9.2, representing a 12.4% decrease relative to the balanced configuration, while preserving acceptable performance levels. Further increasing the cost weight to γ = 0.6 yields additional cost reductions to $7.7, albeit at the expense of increased latency and reduced throughput. Importantly, even in this highly constrained regime, LLM-TradeOpt avoids extreme under-provisioning and maintains stable execution behavior, reflecting effective risk-aware reasoning and constraint enforcement.

Table 11. Performance under different utility weight configurations

Preference Profile	α (Latency)	β (Throughput)	γ (Cost)	Latency (s) ↓	Throughput (QPH) ↑	Cost ($) ↓
Performance-Centric	0.6	0.3	0.1	118.9	531	12.3
Balanced (Default)	0.4	0.4	0.2	127.4	512	10.5
Cost-Aware	0.3	0.3	0.4	139.6	488	9.2
Cost-Dominant	0.2	0.2	0.6	152.4	449	7.7

To further analyze the impact of weight variations on optimization behavior, Table XII reports the relative percentage changes in performance metrics with respect to the balanced configuration. This normalized view facilitates direct comparison of sensitivity patterns across different preference profiles. As illustrated in Table XII, variations in utility weights lead to gradual and monotonic shifts in performance outcomes rather than abrupt or unstable changes. The performance-centric regime reduces latency by 6.7% and increases throughput by 3.7% relative to the balanced configuration,

while incurring a 17.1% increase in cost. Conversely, the cost-dominant regime achieves a substantial 26.7% reduction in monetary cost, accompanied by a 19.6% increase in latency and a 12.3% decrease in throughput. These trade-offs closely align with user-defined priorities, indicating that the framework responds predictably and proportionally to changes in optimization objectives. Notably, the balanced configuration achieves the most favorable compromise among competing objectives, delivering substantial performance gains while maintaining reasonable economic efficiency. This setting consistently yields high utility scores across platforms and workload domains, validating its suitability as a default operational profile. Furthermore, the smooth sensitivity trends observed across Tables XI and XII suggest that the optimization process is well-conditioned and does not rely on finely tuned hyperparameters. Small variations in utility weights do not result in disproportionate performance degradation, highlighting the robustness of the reasoning mechanism.

Table 12. *Relative performance change with respect to balanced configuration (%)*

Preference Profile	Δ Latency (%)	Δ Throughput (%)	Δ Cost (%)
Performance-Centric	−6.7	+3.7	+17.1
Cost-Aware	+9.5	−4.7	−12.4
Cost-Dominant	+19.6	−12.3	−26.7

6. Conclusions

This work presents a systematic examination of performance trade-offs in heterogeneous cloud analytics platforms, focusing on Amazon EMR, Apache Spark on Kubernetes, and Snowflake Compute under realistic workloads drawn from CloudSuite v4.0. Extensive empirical evaluation demonstrates that LLM-TradeOpt consistently outperforms strong state-of-the-art baselines, achieving up to 18.7% reductions in latency, 22.4% improvements in throughput, and 15.3% cost savings, while simultaneously enhancing execution predictability. Several directions remain for future exploration. First, extending the framework to support streaming and hybrid transactional–analytical processing (HTAP) workloads would broaden its applicability to latency-critical and continuously evolving environments. Second, incorporating online learning mechanisms could enable faster adaptation under workload drift and changing resource conditions.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.
- **Use of AI Tools:** The author(s) declare that no generative AI or AI-assisted technologies were used in the writing process of this manuscript.

References

[1]LeisViktor and KuschewskiMaximilian, “Towards cost-optimal query processing in the cloud,” *Proceedings of the VLDB Endowment*, vol. 14, no. 9, pp. 1606–1612, May 2021, doi: https://doi.org/10.14778/3461535.3461549.

[2] H. Zhang, Y. Liu, and J. Yan, “Cost-Intelligent Data Analytics in the Cloud,” *arXiv.org*, 2023. https://arxiv.org/abs/2308.09569 (accessed Jan. 01, 2026).

[3] V. Vyas *et al.*, “Managed Resource Scaling in Amazon EMR,” *Companion of the 2025 International Conference on Management of Data*, pp. 662–674, Jun. 2025, doi: https://doi.org/10.1145/3722212.3724443.

[4] Zhu, Changpeng, Bo Han, and Yinliang Zhao. "A comparative performance study of spark on kubernetes." *Journal of Supercomputing* 78, no. 11 (2022).

[5] J. V. Szlang *et al.*, “Workload Insights from the Snowflake Data Cloud: What Do Production Analytic Queries Really Look Like?,” *Proceedings of the VLDB Endowment*, vol. 18, no. 12, pp. 5126–5138, Aug. 2025, doi: https://doi.org/10.14778/3750601.3750632.

[6] J. Oliveira e Sá, R. Gonçalves, and C. Kaldeich, “Benchmark of Market Cloud Data Warehouse Technologies,” *Procedia Computer Science*, vol. 239, pp. 1212–1219, 2024, doi: https://doi.org/10.1016/j.procs.2024.06.289.

[7] S. Henning, A. Vogel, M. Leichtfried, O. Ertl, and R. Rabiser, “ShuffleBench: A Benchmark for Large-Scale Data Shuffling Operations with

Distributed Stream Processing Frameworks," *arXiv (Cornell University)*, Mar. 2024, doi: https://doi.org/10.1145/3629526.3645036.

[8] G. Cheng, S. Ying, and B. Wang, "Tuning configuration of apache spark on public clouds by combining multi-objective optimization and performance prediction model," *Journal of Systems and Software*, vol. 180, p. 111028, Oct. 2021, doi: https://doi.org/10.1016/j.jss.2021.111028.

[9] X. Huang, H. Zhang, and X. Zhai, "A Novel Reinforcement Learning Approach for Spark Configuration Parameter Optimization," *Sensors*, vol. 22, no. 15, p. 5930, Aug. 2022, doi: https://doi.org/10.3390/s22155930.

[10] R. Tardío, A. Maté, and J. Trujillo, "Beyond TPC-DS, a benchmark for Big Data OLAP systems (BDOLAP-Bench)," *Future Generation Computer Systems*, vol. 132, pp. 136–151, Feb. 2022, doi: https://doi.org/10.1016/j.future.2022.02.015.

[11] Ferdman, Michael. *Cloudsuite: A benchmark suite for cloud services*. 2022.

[12] V. Leis, P. Boncz, A. Kemper, and T. Neumann, "Morsel-driven parallelism," *International Conference on Management of Data*, Jun. 2014, doi: https://doi.org/10.1145/2588555.2610507.

[13] R. Moussa, "TPC-H Benchmark Analytics Scenarios and Performances on Hadoop Data Clouds," *Communications in Computer and Information Science*, pp. 220–234, 2012, doi: https://doi.org/10.1007/978-3-642-30507-8_20.

[14] M. Armbrust, A. Ghodsi, R. Xin, and M. Zaharia, "Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics," 2021. Available: https://15721.courses.cs.cmu.edu/spring2023/papers/02-modern/armbrust-cidr21.pdf

[15] M. Armbrust *et al.*, "Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores," doi: https://doi.org/10.14778/3415478.3415560.

[16] V. Govindarajan, P. Patel, S. Tripathi, M. A. Hoque, and G. S. Kashyap, "MAGIC-Enhanced Keyword Prompting for Zero-Shot Audio Captioning with CLIP Models," *arXiv.org*, 2025. https://arxiv.org/abs/2509.12591 (accessed Jan. 01, 2026).

[17] G. Quattrocchi, E. Incerto, R. Pinciroli, C. Trubiani, and L. Baresi, "Autoscaling Solutions for Cloud Applications Under Dynamic Workloads," *IEEE Transactions on Services Computing*, vol. 17, no. 3, pp. 804–820, May 2024, doi: https://doi.org/10.1109/tsc.2024.3354062.

[18] S. S. Kolawole, G. S. Kashyap, O. E. Kolawole, and M. Yu, "The Future of Fall Prevention: Integrating OpenPose with Cutting-Edge ML Models," *EAI Endorsed Transactions on Pervasive Health and Technology*, vol. 11, Apr. 2025, doi: https://doi.org/10.4108/eetpht.11.9013.

[19] S. K. Mondal *et al.*, "Toward Optimal Load Prediction and Customizable Autoscaling Scheme for Kubernetes," *Mathematics*, vol. 11, no. 12, p. 2675, Jun. 2023, doi: https://doi.org/10.3390/math11122675.

[20] D. R. Augustyn, Ł. Wyciślik, and M. Sojka, "Tuning a Kubernetes Horizontal Pod Autoscaler for Meeting Performance and Load Demands in Cloud Deployments," *Applied Sciences*, vol. 14, no. 2, p. 646, Jan. 2024, doi: https://doi.org/10.3390/app14020646.

[21] S. Tripathi, Nafis, Md Tabrez, I. Hussain, and J. Gao, "The Confidence Paradox: Can LLM Know When It's Wrong," *arXiv.org*, 2025. https://arxiv.org/abs/2506.23464 (accessed Jan. 01, 2026).

[22] Y. Ji *et al.*, "Adapting Large Language Models to Log Analysis with Interpretable Domain Knowledge," *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, pp. 1135–1144, Nov. 2025, doi: https://doi.org/10.1145/3746252.3761189.

[23] A. Soni *et al.*, "Can We Predict Your Next Move Without Breaking Your Privacy?," *arXiv.org*, 2025. https://arxiv.org/abs/2507.08843 (accessed Jan. 01, 2026).

[24] T. Cui *et al.*, "LogEval: A Comprehensive Benchmark Suite for Large Language Models In Log Analysis," *arXiv.org*, 2024. https://arxiv.org/abs/2407.01896 (accessed Jan. 01, 2026).

[25] P. Tang, S. Tang, H. Pu, Z. Miao, and Z. Wang, "MicroRCA-Agent: Microservice Root Cause Analysis Method Based on Large Language Model Agents," *arXiv.org*, 2025. https://arxiv.org/abs/2509.15635 (accessed Jan. 01, 2026).