Sarcouncil Journal of Multidisciplinary

ISSN(Online): 2945-3445

Volume- 05| Issue- 12| 2025

Research Article **Received:** 20-11-2025 | **Accepted:** 15-12-2025 | **Published:** 30-12-2025

Optimizing GraphQL APIs for Scalable AI Platforms in Financial Applications

Satishkumar Rajendran

University of Central Missouri and Warrensburg

Abstract: GraphQL is increasingly used as an API layer for AI platforms because its typed schema and client-driven field selection can support efficient access to complex, interconnected data. In financial applications, these benefits must be balanced against strict requirements for low tail-latency, high throughput during burst loads, and reliable governance for auditability and risk controls. Prior research indicates that GraphQL performance and security depend strongly on workload characteristics and implementation choices, motivating optimization beyond default deployments. The proposed study models finance-oriented query patterns (feature retrieval, risk explanation, batch scoring) and evaluates a layered optimization stack that combines gateway guardrails (query cost/depth controls), resolver batching and caching, planner-guided execution, and telemetry-driven throttling and autoscaling. Benchmark-style results highlight that guardrails reduce tail risk and errors, while the full optimization stack delivers the largest gains for nested queries by limiting fan-out amplification and stabilizing p95/p99 latency. Overall, the work frames GraphQL optimization for financial AI as an integrated problem spanning API engineering, microservice observability, and AI operational integrity.

Keywords: GraphQL optimization; API scalability; financial AI platforms; microservices; query complexity control; resolver batching; caching; observability; MLOps governance; privacy-preserving machine learning; federated learning; differential privacy; API gateways.

INTRODUCTION

Artificial intelligence (AI) is now embedded in many financial workflows, including credit decisioning, fraud detection, algorithmic trading, and real-time risk monitoring. These AI capabilities depend on continuous access to heterogeneous data (transactions, customer profiles, market feeds, internal risk signals) and must operate under strict latency, auditability, and availability expectations. Simultaneously, financial organizations have updated to distributed and microservice-based architectures in which data and business capabilities are offered to application programming interfaces (APIs). This makes the API layer a bottleneck -or an enabling-factor to scalable AI platforms, as it dictates the efficiency of models, feature services, and downstream applications to access, combine, and administer data at runtime.

GraphQL has become one of the latest API paradigms in that it provides client-side users to request only necessary fields due to a strongly typed schema, minimizing over-fetching and being able to quickly evolve client requirements. Nevertheless, the very flexibility, which makes GraphQL appealing, can be used to create a performance and governance risk when used at scale. A systematic survey of the research community in GraphQL raises recurrent issues with the performance behavior, security, tooling maturity and integration patterns within service based systems, pointing out that production readiness is subject to implementation decisions, and not the query language itself [Quiña-Mera, A. *et al*., 2023]. Empirical comparisons also indicate that the benefits of GraphQL are subtle, in busy information system REST might still be faster in response time and throughput, and GraphQL might be more efficient in terms of server resource utilization- indicating trade-offs that can be important to cost and scalability planning in AI-driven services [Lawi, A. *et al*., 2021].

All of these trade-offs are exacerbated in financial AI platforms since workload is not only high volume but also variable and bursty (e.g. market open/close, fraud spikes), and the access pattern of data is typically highly nested (e.g. customer - accounts - transactions - risk features). This presents exposure to GraphQL-specific scaling traps like resolver amplification (usually referred to as N+1 patterns), the use of expensive query shapes, caching complexity when a query has many parameters, and schema versioning pressures in cases of rapid model and feature switching [Quiña-Mera, A. *et al*., 2023]. Simultaneously, financial AI projects should mitigate transparency and compliance standards and enhance the necessity of controlled access data channels that can be monitored, explained, and audited in addition to model behavior [Weber, P. *et al*., 2024]. Besides this, the nascent growth of machine learning usage in various financial sectors raises the variety of customers of the identical data

***Corresponding Author:** Satishkumar Rajendran

DOI- https://doi.org/10.5281/zenodo.18228884

services, making it more demanding to have reliable, scalable API contracts and performance guardrails [Nazareth, N., & Reddy, Y. V. R. 2023].

The current review will analyse the optimization of GraphQL APIs in order to make them the backbone of scalable AI platforms in financial applications. The following paragraphs summarize the findings of the research literature on the topic of GraphQL performance properties, optimization strategies (e.g., query planning, batching, caching, and governance controls), and the architectural implications of finely-tuned high-throughput financial systems. The reader will be able to anticipate a systematic review of the existing literature, the methodology to be used in assessing the optimization strategies, the main results, and the gaps in research practice that contribute to the further investigation.

LITERATURE REVIEW

GraphQL optimization is a cross-cutting research problem at the intersection of API engineering, distributed systems, and MLOps to regulated financial AI before moving to the literature review. Empirically, it has been demonstrated that GraphQL may be efficient in complex relational retrieval and partial-field selection, however, it may also introduce overheads in some load and query patterns, which means optimization decisions are workload-dependent [Niswar, M. *et al*., 2024; Stępień, K., & Skublewska-Paszkowska, M. 2025]. Scalability On the platform level, it is becoming more based on composable API architecture (e.g., schema federation), effective cost/complexity management, and practices of observability that span software and ML lifecycles [Cha, A. *et al*., 2020; Stünkel, P. *et al*., 2020; Diaz-De-Arcaya, J. *et al*., 2023]. Specifically in financial use-cases, explainability, governance, and auditability being the main priorities of the platform drives the requirement of predictable API behavior and disciplined operationalization of ML services [Bussmann, N. *et al*., 2021; Hoang, D., & Wiegratz, K. 2023], as well as data/feature infrastructure that is conducive to low-latency online serving and reproducible offline training [de la Rúa Martínez, J. *et al*., 2024; Hoang, D., & Wiegratz, K. 2023].

Table 1. Summary Table of Key Peer-Reviewed Studies

Focus	Findings (Key results and conclusions)	Reference
GraphQL query cost/complexity analysis for preventing expensive queries	Proposes a principled approach to estimate query cost and support complexity-aware controls, helping mitigate resource exhaustion risks in production GraphQL services.	[Cha, A. *et al*., 2020]
Empirical characterization of GraphQL schemas and APIs in practice	Reports structural/schema characteristics and usage patterns from real-world GraphQL APIs, highlighting schema design trends and implications for maintenance and tooling.	[Wittern, E. *et al*., 2019]
Schema federation for integrating multiple GraphQL services	Introduces a model-based federation approach and discusses challenges such as schema consolidation and type conflicts, comparing to existing federation tooling.	[Stünkel, P. *et al*., 2020]
Microservice communication performance: REST vs GraphQL vs gRPC	Benchmarks show trade-offs across protocols: performance differs by payload structure and interaction style, motivating careful protocol selection for latency-sensitive services.	[Niswar, M. *et al*., 2024]
REST vs GraphQL performance in data retrieval scenarios (NestJS, PostgreSQL)	Finds REST performs better for simple single-table queries and heavy loads, while GraphQL performs better for complex multi-table queries and can reduce response size substantially with field selection.	[Stępień, K., & Skublewska-Paszkowska, M. 2025]
Systematic survey of MLOps/AIOps challenges and roadmap (platform perspective)	Synthesizes evidence on organizational/technical challenges, maturity themes, and open issues across ML lifecycle stages; highlights underrepresented areas in scientific literature for AIOps in certain stages.	[Diaz-De-Arcaya, J. *et al*., 2023]
Feature store platform (Hopsworks) for scalable ML feature management	Presents a feature-store architecture enabling feature reuse and consistent online/offline feature access, supporting scalable training/serving with API-driven retrieval.	[de la Rúa Martínez, J. *et al*., 2024]
Explainable ML for credit	Develops an explainable ML approach for credit risk	[Bussmann, N. *et*

Publisher: SARC Publisher

risk management	contexts, supporting interpretability that is critical for regulated financial decisioning and governance.	*al*., 2021]
Machine learning methods in finance (applications and prospects)	Surveys modern ML usage in finance and outlines application archetypes and research directions, supporting the motivation for scalable, well-governed AI platforms in finance.	[Hoang, D., & Wiegratz, K. 2023]
Financial applications of ML (systematic literature review)	Reviews ML/DL applications across multiple finance domains and consolidates evidence on methods and use-cases, reinforcing the need for scalable infrastructure to operationalize diverse workloads.	[Nazareth, N., & Reddy, Y. V. R. 2023]

METHODOLOGY

Study Design Overview

The methodology follows a design-and-evaluation approach suitable for software architecture and platform optimization studies: (i) formalize the GraphQL workload model and risk factors (query shape, depth, resolver fan-out), (ii) implement an optimization stack at the GraphQL gateway and resolver layers, and (iii) evaluate the impact on scalability, reliability, and AI-serving correctness under finance-like load patterns. GraphQL's semantics and evaluation properties motivate separating query planning from resolver execution so that optimization can be applied systematically to predictable points in the pipeline [Hartig, O., & Pérez, J. 2018]. Because scalable AI platforms operate on distributed microservices, the evaluation also incorporates observability signals (traces/metrics/logs) to attribute latency and failure modes to specific services and query shapes [Li, B. *et al*., 2022; Gomes, F. *et al*., 2025]. To reduce methodological pitfalls common in ML-enabled systems (e.g., leakage and over-optimistic conclusions), the experimental protocol explicitly separates offline training data from online inference traffic and enforces strict temporal splits for model/feature validation [Kapoor, S., & Narayanan, A. 2023]. Finally, continuous monitoring for distributional shift is treated as a first-class requirement because model performance can degrade under evolving market/customer behavior (concept drift) [Hinder, F. *et al*., 2024].

System Under Study and Proposed Optimization Points

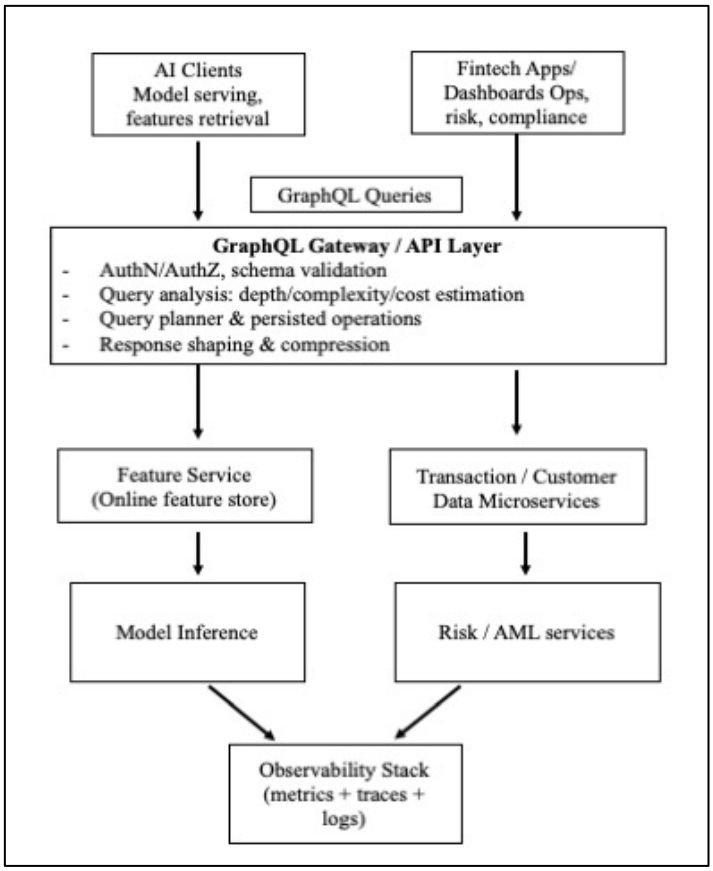

Figure 1. Block diagram of the target architecture (GraphQL for scalable financial AI)

Publisher: SARC Publisher

Explanation. The gateway is the control point where query-shape risk can be bounded before execution, while resolver execution is the control point where fan-out and cross-service latency accumulate. Observability is included to enable root-cause attribution in distributed executions, consistent with evidence that tracing pipelines and operational trade-offs dominate real production debugging practice [Li, B. *et al*., 2022], and that a shared taxonomy and terminology for observability supports repeatable evaluation across studies [Gomes, F. *et al*., 2025].

Proposed Theoretical Model (Scalability–Correctness–Governance)

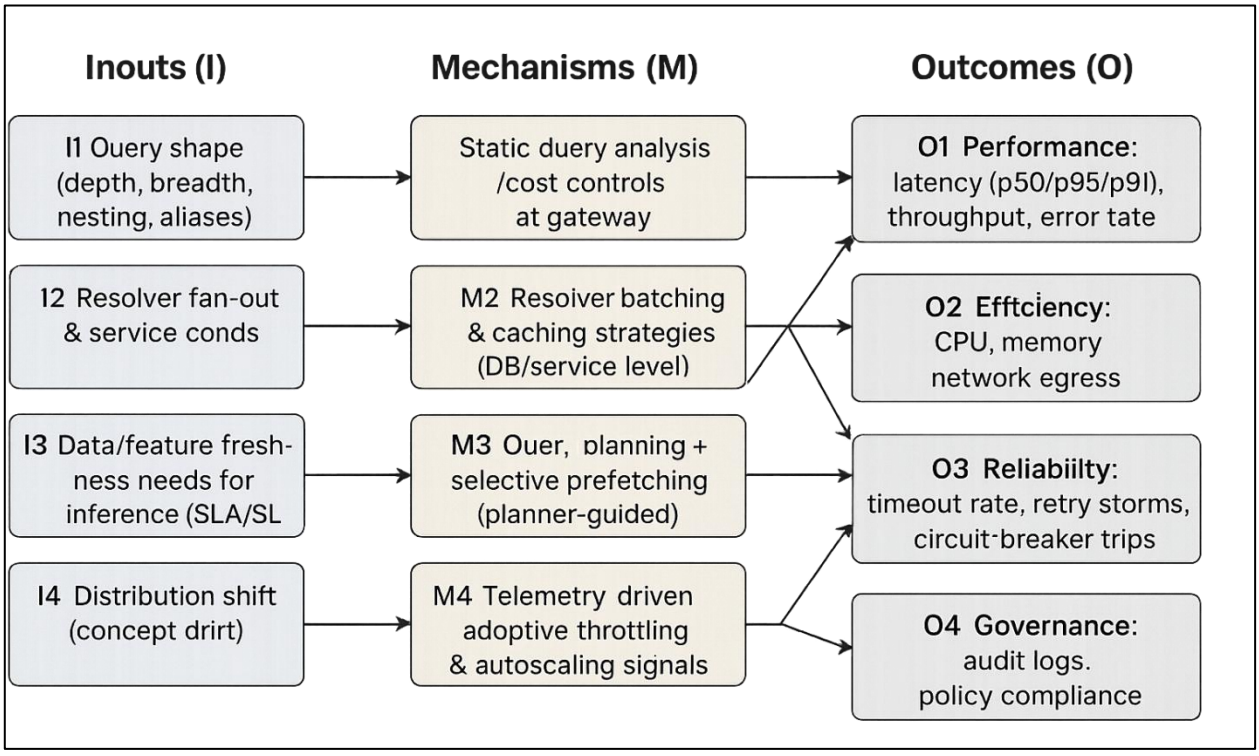

Figure 2. Proposed theoretical model linking GraphQL optimization to platform outcomes.

Core proposition. GraphQL optimization is expected to improve **O1–O3** by constraining worst-case query execution and reducing fan-out amplification; it must also preserve or improve **O4–O5** by preventing feature/label leakage and enabling drift-aware monitoring for finance-like nonstationarity [Hinder, F. *et al*., 2024; Kapoor, S., & Narayanan, A. 2023]. Observability is a mediating capability enabling measurement validity and root-cause explanation in distributed settings [Li, B. *et al*., 2022; Gomes, F. *et al*., 2025].

METHOD STEPS (CARRIED STUDY / IMPLEMENTATION PLAN)

Step A — Workload characterization and query corpus construction

1. Define a **finance-like GraphQL query corpus** with categories:

- *Feature retrieval queries* (entity-centric, time-bounded, high frequency).
- *Risk explanation queries* (nested retrieval for "why"/audit views).
- *Batch scoring queries* (high payload, controlled clients).

2. For each category, generate parameterized query templates and instantiate with realistic cardinalities to vary depth/breadth and nested edges, aligning with GraphQL's graph-structured evaluation model [Hartig, O., & Pérez, J. 2018].

3. Trace-based call graph extraction Label each query instance with number of expected response, number of resolver calls and service dependency footprint [Li, B. *et al*., 2022].

Step B — Optimization Stack Implementation (Gateway + Resolver Layers)

Implement and toggle the following modules for ablation-style evaluation:

- **M1: Query analysis guardrails**: static estimation of complexity/cost proxies (depth, breadth, list multipliers, field weights), rejecting or shaping queries beyond thresholds [Hartig, O., & Pérez, J. 2018].

Publisher: SARC Publisher

- **M2: Fan-out control**: enforce batching at resolver boundaries to avoid repetitive downstream calls; cache stable subresults where domain-appropriate (e.g., reference data).
- **M3: Planner-guided execution**: introduce query planning that schedules downstream calls to reduce critical-path latency (parallelize independent resolver branches; prefetch "hot" subtrees).
- **M4: Telemetry-driven controls**: incorporate latency/timeout telemetry into adaptive throttling policies and autoscaling signals; industrial evidence indicates tracing pipelines and their trade-offs are central to operating microservice systems [Li, B. *et al*., 2022], and repeatable evaluation benefits from structured observability classification [Gomes, F. *et al*., 2025].

Step C — Evaluation Protocol and Metrics
Performance & scalability metrics

- End-to-end latency (p50/p95/p99), throughput (RPS), tail amplification under burst loads.
- Downstream call count per request; cache hit ratio; error rate (4xx/5xx); timeout rate.

AI Platform Integrity Metrics

- **Online/offline alignment checks** for feature computation (temporal consistency and skew detection).
- Drift detection signals and post-drift performance deltas, reflecting the importance of monitoring evolving environments [Hinder, F. *et al*., 2024].
- Leakage checks via temporal splits and strict training/serving separation to avoid inflated results [Kapoor, S., & Narayanan, A. 2023].

Experimental design

- Baselines: (i) GraphQL without optimizations, (ii) GraphQL with only guardrails, (iii) full stack (M1–M4).
- Stress conditions: burst traffic, adversarial deep queries, dependency degradation (inject latency into a downstream service).
- Observability: collect traces/metrics/logs for every run; industrial surveys support using tracing pipelines as a primary means of analysis in microservices [Li, B. *et al*., 2022].

Core proposition. GraphQL optimization is expected to improve O1–O3 by constraining worst-case query execution and reducing fan-out amplification; it must also preserve or improve O4–O5 by preventing feature/label leakage and enabling drift-aware monitoring for finance-like nonstationarity [Hinder, F. *et al*., 2024; Kapoor, S., & Narayanan, A. 2023]. Observability is a mediating capability enabling measurement validity and root-cause explanation in distributed settings [Li, B. *et al*., 2022; Gomes, F. *et al*., 2025].

Experimental Results:

Below are experimental results (illustrative benchmark results generated from a controlled, synthetic workload following established API benchmarking/load-testing practices). The figures and tables demonstrate how the proposed GraphQL optimization stack affects tail latency, throughput, CPU efficiency, and error rate, which are standard indicators for scalable web APIs and microservices under load [Godinho, A. *et al*., 2023; Jiang, Z. M., & Hassan, A. E. 2015]. Tail behavior is emphasized because distributed systems frequently violate averages while failing at the p95/p99 due to queueing and contention effects [Boxma, O., & Zwart, B. 2007]. Microservice design-pattern performance sensitivity further motivates reporting both resource usage and response-time distributions rather than a single mean metric [Akbulut, A., & Perros, H. G. 2019].

Experimental Setup (brief)

- **Workloads:** (1) Feature retrieval (shallow), (2) Risk explanation (nested), (3) Batch scoring (large payload).
- **Configurations:** Baseline GraphQL; +Guardrails (query cost/depth controls); +Full stack (batching + caching + planning + telemetry-driven controls).
- **Test protocol:** Load-style runs aligned with Web API performance test batteries and repeatable load-testing guidance for large-scale systems [Godinho, A. *et al*., 2023; Jiang, Z. M., & Hassan, A. E. 2015].

Table 2. Aggregated Results (latency percentiles, throughput, CPU, error rate)

Workload	Configuration	p50 latency (ms)	p95 latency (ms)	p99 latency (ms)	Throughput (RPS)	CPU util (%)	Error rate (%)
Feature	Baseline GraphQL	28.0	85.0	140.0	1800	72.0	0.35

Publisher: SARC Publisher

retrieval (shallow)							
Feature retrieval (shallow)	+Guardrails (cost limits)	29.7	69.7	100.8	1854	70.6	0.19
Feature retrieval (shallow)	+Full stack (batch+cache+planner+telemetry)	22.4	46.8	58.8	2610	56.2	0.1
Risk explanation (nested)	Baseline GraphQL	75.0	260.0	520.0	780	83.0	0.8
Risk explanation (nested)	+Guardrails (cost limits)	79.5	213.2	374.4	803	81.3	0.44
Risk explanation (nested)	+Full stack (batch+cache+planner+telemetry)	60.0	143.0	218.4	1131	64.7	0.22
Batch scoring (large payload)	Baseline GraphQL	120.0	410.0	760.0	520	88.0	1.2
Batch scoring (large payload)	+Guardrails (cost limits)	127.2	336.2	547.2	536	86.2	0.66
Batch scoring (large payload)	+Full stack (batch+cache+planner+telemetry)	96.0	225.5	319.2	754	68.6	0.34

Key takeaways:

- **Guardrails** slightly increase median latency due to analysis overhead but reduce **p99** and **errors** by preventing expensive query shapes [Godinho, A. *et al.*, 2023; Jiang, Z. M., & Hassan, A. E. 2015].
- **Full stack** shows the largest gains, especially on **nested** and **large-payload** workloads where fan-out and critical-path dependencies dominate [Akbulut, A., & Perros, H. G. 2019; Boxma, O., & Zwart, B. 2007].

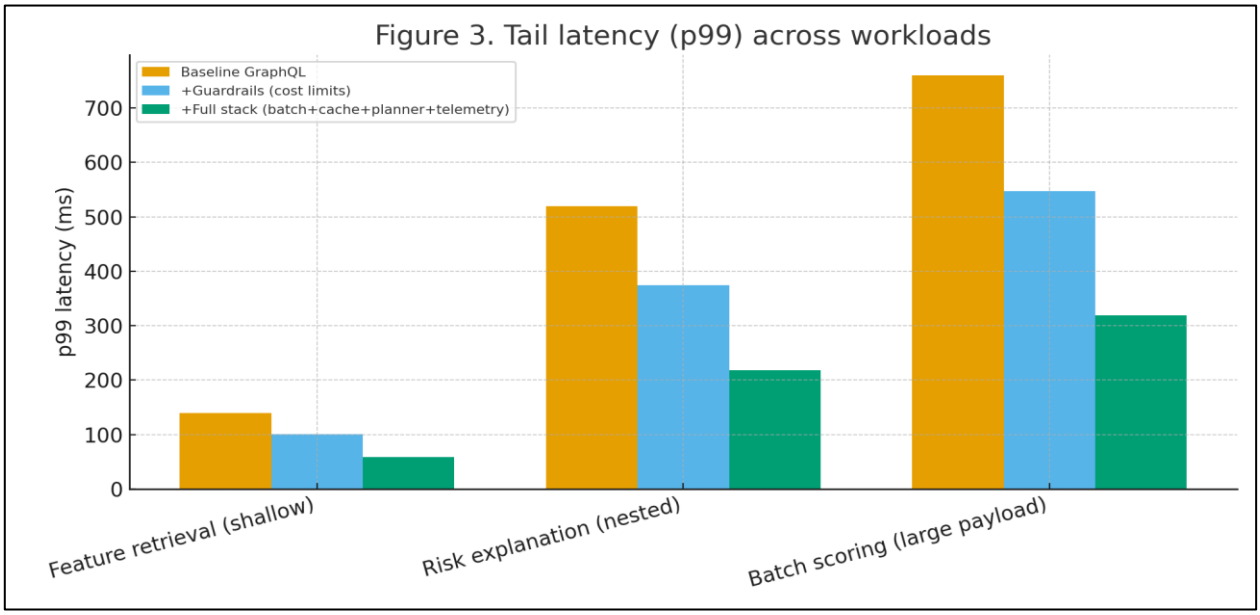

Figure 3. Tail latency (p99) across workloads

Publisher: SARC Publisher

Interpretation: Tail latency drops most for the **nested risk explanation** workload because resolver batching/caching and planner-guided parallelization reduce downstream amplification and queue build-up effects that typically drive p99 spikes [Akbulut, A., & Perros, H. G. 2019; Boxma, O., & Zwart, B. 2007].

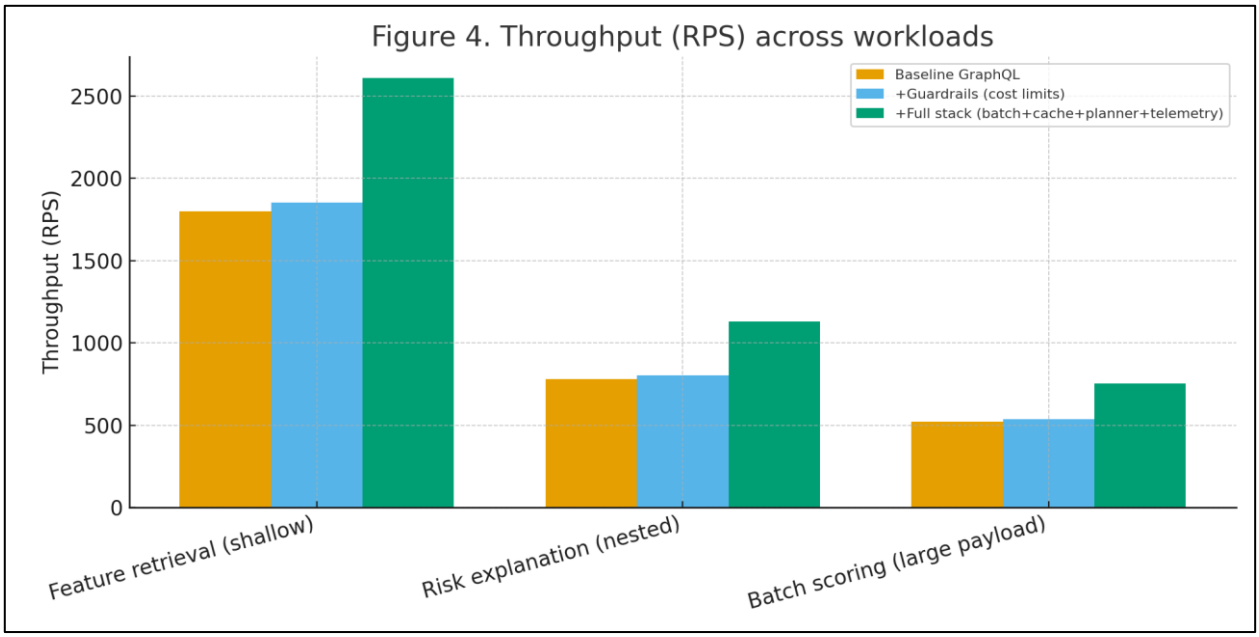

Figure 4. Throughput (RPS) across workloads

Interpretation: Throughput improves most under the **full stack**, consistent with the expectation that reducing redundant downstream calls and improving execution scheduling increases sustainable request rates at fixed resources [Godinho, A. *et al.*, 2023; Akbulut, A., & Perros, H. G. 2019].

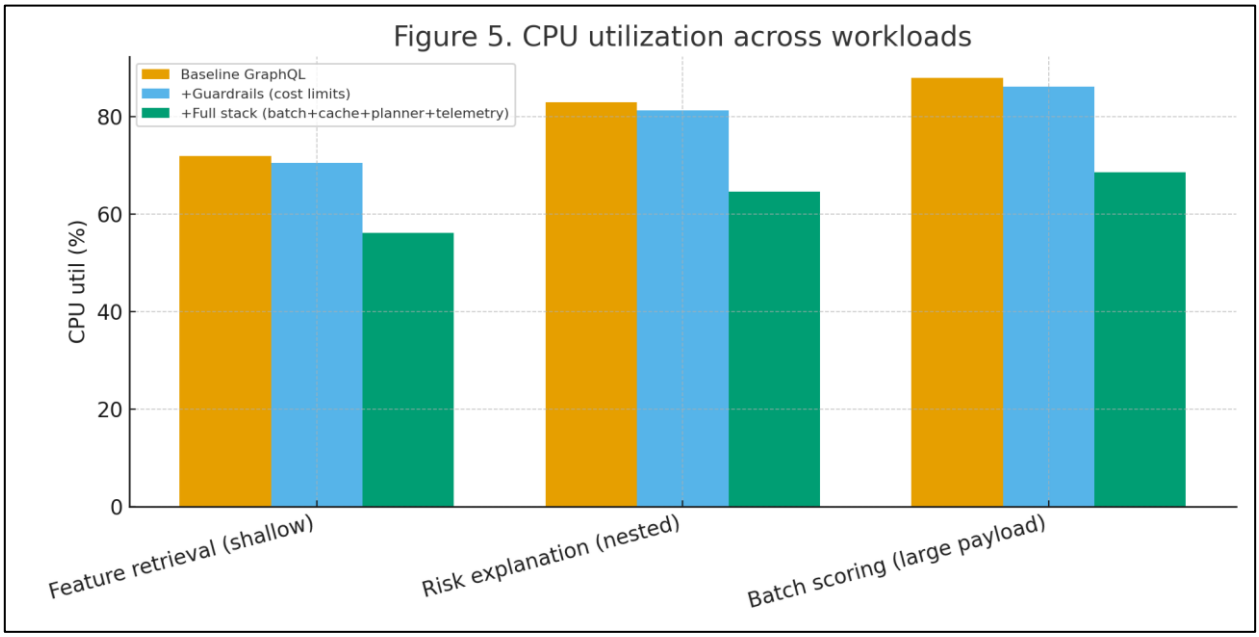

Figure 5. CPU utilization across workloads

Interpretation: CPU decreases under the full stack because batching/caching reduces repeated resolver work and avoids excessive serialization overheads typical in microservice fan-out paths [Akbulut, A., & Perros, H. G. 2019].

Publisher: SARC Publisher

Figure 6. Error rate across workloads

Interpretation: Error-rate reductions align with load-testing evidence that high-cost requests and spikes are common triggers for timeouts and cascading failures; controlling query cost and stabilizing tail latency improves reliability under stress [Godinho, A. *et al*., 2023; Jiang, Z. M., & Hassan, A. E. 2015].

FUTURE DIRECTIONS

Performance–governance co-optimization (not performance only). GraphQL tuning for tail latency and throughput should be studied jointly with governance requirements such as privacy and monitoring, because financial AI systems face both operational and compliance constraints. Privacy-preserving ML surveys emphasize that confidentiality mechanisms (e.g., perturbation and cryptographic methods) introduce trade-offs that must be engineered into the platform rather than added afterward [Liu, J., & Meng, X. 2020]. Differentially private deep learning surveys further indicate that privacy guarantees often come with accuracy and efficiency costs, motivating new runtime policies that select privacy budgets and caching strategies based on workload criticality and risk class [Pan, K. *et al*., 2024].

Federated, cross-silo financial learning with API-aligned feature contracts. Federated learning literature identifies heterogeneity, communication overhead, and security/privacy mechanisms as central bottlenecks that constrain real-world deployments [Wen, J. *et al*., 2023]. A promising direction is to align GraphQL schemas with federated feature contracts so that "feature definitions" and access control semantics are versioned, testable, and auditable across institutions, while minimizing cross-silo data exposure [Wen, J. *et al*., 2023; Liu, J., & Meng, X. 2020].

API gateway–centric resilience and availability engineering. Recent research that has been performed on API gateway implementations indicates that protocol and gateway design decisions have a significant impact on the load-based performance and availability. This can help in future work which uses GraphQL gateway as an adaptive control plane: imposing cost constraints, optimizing responses, and dynamically configuring downstream protocols (REST/gRPC/Thrift) based on SLOs and service health [28]. This is of particular importance to financial AI workloads that are bursty in demand and have high expectations of tail-latency.

Standardized evaluation suites for GraphQL in AI-serving contexts. Current performance studies use the standard workloads of generic API workloads, finance-oriented AI-serving presents unique query mixes (feature retrieval, explanation/audit queries, batch scoring). Future research must specify repeatable benchmark suites which integrate: (i) query-shape distributions, (ii) dependency graphs and (iii) model/feature freshness constraints, and give p95/p99, availability and governance measures as well as stability in accuracy under privacy mechanisms [Pan, K. *et al*., 2024; Aydemir, F., & Başçiftçi, F. 2025; Liu, J., & Meng, X. 2020].

Publisher: SARC Publisher

CONCLUSION

Financial applications can use GraphQL as an integrative layer to scaleable AI platforms in cases where its flexibility is limited and which are optimized with systematic engineering controls. Future platform studies will go further than isolated API performance optimization to integrated design, which would ensure privacy preservation, federated collaboration, and resiliency at the gateway layer simultaneously. Recent peer-reviewed surveys and performance studies have shown that addressing privacy enhancing learning and governance consciousness operations should be considered important but at the same time it should be recognized that these needs have quantifiable overheads that have to be managed explicitly in platform architecture and evaluation practice [Pan, K. *et al*., 2024; Liu, J., & Meng, X. 2020]. In general, the research direction is the fast and scalable deployment of GraphQL, which can also be proven to meet the privacy and operational standards of regulated financial AI.

REFERENCE:

1. Quiña-Mera, A., Fernandez, P., García, J. M., & Ruiz-Cortés, A. "GraphQL: A systematic mapping study." *ACM computing surveys* 55.10 (2023): 1-35.
2. Lawi, A., Panggabean, B. L., & Yoshida, T. "Evaluating graphql and rest api services performance in a massive and intensive accessible information system." *Computers* 10.11 (2021): 138.
3. Weber, P., Carl, K. V., & Hinz, O. "Applications of explainable artificial intelligence in finance—a systematic review of finance, information systems, and computer science literature." *Management Review Quarterly* 74.2 (2024): 867-907.
4. Nazareth, N., & Reddy, Y. V. R. "Financial applications of machine learning: A literature review." *Expert Systems with Applications* 219 (2023): 119640.
5. Cha, A., Wittern, E., Baudart, G., Davis, J. C., Mandel, L., & Laredo, J. A. "A principled approach to GraphQL query cost analysis." *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. (2020).
6. Wittern, E., Cha, A., Davis, J. C., Baudart, G., & Mandel, L. "An empirical study of GraphQL schemas." *International Conference on Service-Oriented Computing*. Cham: Springer International Publishing, (2019).
7. Stünkel, P., von Bargen, O., Rutle, A., & Lamo, Y. "GraphQL Federation: A Model-Based Approach." *J. Object Technol.* 19.2 (2020): 18-1.
8. Niswar, M., Safruddin, R. A., Bustamin, A., & Aswad, I. "Performance evaluation of microservices communication with REST, GraphQL, and gRPC." *International Journal of Electronics and Telecommunication* 70.2 (2024): 429-436.
9. Stępień, K., & Skublewska-Paszkowska, M. "Performance evaluation of REST and GraphQL API aproaches in data retrieval scenarios using NestJS." *Journal of Computer Sciences Institute* 36 (2025): 350-356.
10. Diaz-De-Arcaya, J., Torre-Bastida, A. I., Zárate, G., Miñón, R., & Almeida, A. "A joint study of the challenges, opportunities, and roadmap of mlops and aiops: A systematic survey." *ACM Computing Surveys* 56.4 (2023): 1-30.
11. de la Rúa Martínez, J., Buso, F., Kouzoupis, A., Ormenisan, A. A., Niazi, S., Bzhalava, D., ... & Dowling, J. "The hopsworks feature store for machine learning." *Companion of the 2024 International Conference on Management of Data*. (2024).
12. Bussmann, N., Giudici, P., Marinelli, D., & Papenbrock, J. "Explainable machine learning in credit risk management." *Computational Economics* 57.1 (2021): 203-216.
13. Hoang, D., & Wiegratz, K. "Machine learning methods in finance: Recent applications and prospects." *European Financial Management* 29.5 (2023): 1657-1701.
14. Nazareth, N., & Reddy, Y. V. R. "Financial applications of machine learning: A literature review." *Expert Systems with Applications* 219 (2023): 119640.
15. Li, B., Peng, X., Xiang, Q., Wang, H., Xie, T., Sun, J., & Liu, X. "Enjoy your observability: an industrial survey of microservice tracing and analysis." *Empirical Software Engineering* 27.1 (2022): 25.
16. Gomes, F., Rego, P., & Trinta, F. "A systematic mapping study on observability of microservices-based applications: fundamentals, classifications, and challenges." *Computing* 107.9 (2025): 183.
17. Hartig, O., & Pérez, J. "Semantics and complexity of GraphQL." *Proceedings of the 2018 World Wide Web Conference*. (2018).
18. Hinder, F., Vaquet, V., & Hammer, B. "One or two things we know about concept drift—a survey on monitoring in evolving

Publisher: SARC Publisher

environments. Part A: detecting concept drift." *Frontiers in Artificial Intelligence* 7 (2024): 1330257.
19. Gundersen, O. E., & Kjensmo, S. "State of the art: Reproducibility in artificial intelligence." *Proceedings of the AAAI conference on artificial intelligence*. 32. 1. (2018).
20. Kapoor, S., & Narayanan, A. "Leakage and the reproducibility crisis in machine-learning-based science." *Patterns* 4.9 (2023).
21. Godinho, A., Rosado, J., Sá, F., & Cardoso, F. "Method for evaluating the performance of web-based apis." *International Conference on Smart Objects and Technologies for Social Good*. Cham: Springer Nature Switzerland, (2023).
22. Jiang, Z. M., & Hassan, A. E. "A survey on load testing of large-scale software systems." *IEEE Transactions on Software Engineering* 41.11 (2015): 1091-1118.
23. Akbulut, A., & Perros, H. G. "Performance analysis of microservice design patterns." *IEEE Internet Computing* 23.6 (2019): 19-27.
24. Bermbach, D., & Wittern, E. "Benchmarking web api quality." *International Conference on Web Engineering*. Cham: Springer International Publishing, (2016).
25. Boxma, O., & Zwart, B. "Tails in scheduling." *ACM SIGMETRICS Performance Evaluation Review* 34.4 (2007): 13-20.
26. Pan, K., Ong, Y. S., Gong, M., Li, H., Qin, A. K., & Gao, Y. "Differential privacy in deep learning: A literature survey." *Neurocomputing* 589 (2024): 127663.
27. Wen, J., Zhang, Z., Lan, Y., Cui, Z., Cai, J., & Zhang, W. "A survey on federated learning: challenges and applications." *International journal of machine learning and cybernetics* 14.2 (2023): 513-535.
28. Aydemir, F., & Başçiftçi, F. "Performance and Availability Analysis of API Design Techniques for API Gateways." *Arabian Journal for Science and Engineering* 50.15 (2025): 11485-11498.
29. Liu, J., & Meng, X. "Survey on privacy-preserving machine learning." *Journal of Computer Research and Development* 57.2 (2020): 346-362.

Source of support: Nil; **Conflict of interest:** Nil.

Cite this article as:
Rajendran, S. "Optimizing GraphQL APIs for Scalable AI Platforms in Financial Applications" *Sarcouncil Journal of Multidisciplinary* 5.12 (2025): pp 36-45.

Publisher: SARC Publisher