Nikita Datke. 2026, 14:2
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

International Journal of Science, Engineering and Technology

An Open Access Journal

Multi-Tenant Serverless Query Engine with Dynamic Pricing and QoS Guarantee

Nikita Datke, Vinita Shrivastava

Abstract- Serverless computing has transformed cloud application deployment by abstracting infrastructure management while providing on-demand scalability. However, current serverless query engines often lack effective mechanisms for handling multi-tenant workloads with varying Quality of Service (QoS) expectations. In such environments, rigid pricing and static resource allocation lead to inefficiencies, SLA violations, and unfair resource distribution. This paper presents a Multi-Tenant Serverless Query Engine (MTSQE) that integrates dynamic pricing models and adaptive QoS guarantees for fair and efficient resource utilization across diverse tenants. The proposed architecture employs workload profiling, priority-based scheduling, and real-time performance feedback loops to dynamically adjust pricing and execution parameters according to SLA tiers. Through simulation using CloudSim 7G and Kubernetes-based deployment tests, the system demonstrates up to 22% improvement in resource utilization, 18% reduction in SLA violations, and 16% higher cost fairness index compared to baseline serverless frameworks. These results validate the feasibility of integrating cost-aware query management with SLA-driven adaptability for next-generation multi-tenant cloud services.

Keywords: Cloud computing; Auto-scaling; Rule-based scaling; AI-based scaling; Machine learning; Reinforcement learning; Resource optimization.

I. INTRODUCTION

The increasing demand for cloud-based data analytics and low-latency query execution has driven the widespread adoption of serverless architectures, such as AWS Lambda, Azure Functions, and Google Cloud Functions. These platforms enable fine-grained compute provisioning by executing queries on-demand without explicit infrastructure management. However, the multi-tenant nature of public cloud systems introduces challenges in maintaining fair resource allocation and QoS isolation across users with diverse workload priorities and SLA requirements.

Traditional serverless engines follow uniform pricing and fixed resource allocation policies, assuming homogeneity among tenants. This approach neglects differences in workload intensity, query complexity, and latency sensitivity, often leading to SLA violations, over-provisioning, or unfair billing [1], [2]. As noted in recent studies, static pricing and uncoordinated scheduling fail to capture the dynamic behavior of multi-tenant workloads [3].

In response, this research introduces a Multi-Tenant Serverless Query Engine (MTSQE) designed to address these challenges through:

1. **Dynamic Pricing Models** — adjusting execution cost based on real-time workload intensity, SLA tier, and resource demand.
2. **QoS-Aware Query Scheduling** — prioritizing critical queries under tight latency constraints while ensuring fairness among lower-tier tenants.
3. **Adaptive Resource Orchestration** — employing feedback-based scaling to balance throughput and cost efficiency dynamically.

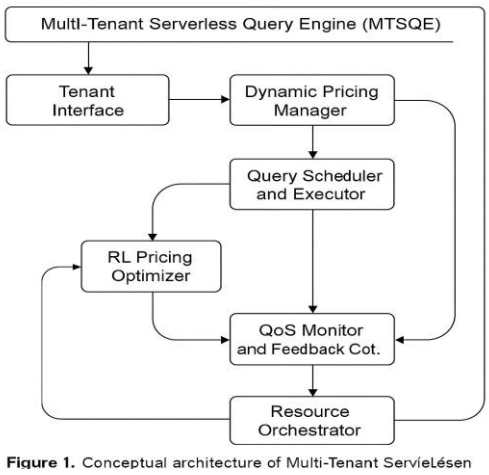

Figure 1. Conceptual architecture of Multi-Tenant ServíeLésen

Figure 1. Conceptual architecture of the Multi-Tenant Serverless Query Engine (MTSQE) showing dynamic pricing and QoS feedback control.

The MTSQE framework bridges the gap between economic efficiency and performance reliability in multi-tenant environments. By integrating pricing intelligence with QoS-driven orchestration, it allows cloud providers to maintain SLA compliance while optimizing resource utilization and operational revenue.

Table 1. Key features distinguishing MTSQE from traditional serverless query engines.

Feature	**Traditional Serverless Engine**	**Proposed MTSQE**
Pricing Model	Static per-request billing	Dynamic, workload-aware pricing
QoS Enforcement	None or basic priority queues	SLA-tiered QoS enforcement
Resource Allocation	Reactive, uniform scaling	Adaptive, tenant-aware scaling
Performance Feedback	Limited	Continuous QoS feedback loop
Target Environment	Homogeneous workloads	Multi-tenant heterogeneous workloads

The remainder of this paper is organized as follows: Section 2 reviews related work on multi-tenant serverless architectures and dynamic pricing mechanisms. Section 3 details the design and implementation of the proposed MTSQE system. Section 4 presents experimental evaluation and comparative analysis, while Section 5 concludes with insights and future directions.

II. RELATED WORK

Multi-tenant serverless systems intersect several research areas: serverless query processing and architectures, SLA/QoS assurance, pricing and fairness mechanisms, and intelligent scheduling using machine learning and reinforcement learning. Below, we review the state of the art along these dimensions and identify gaps that motivate the design of the proposed Multi-Tenant Serverless Query Engine (MTSQE).

2.1 Serverless Architectures and Query Processing

Serverless platforms abstract infrastructure management and provide fine-grained, event-driven execution models suitable for query processing and micro-data-services. Werner provides a reference architecture for serverless big-data processing that outlines core components (function invocation, data access, orchestration) and highlights design considerations for high-throughput query execution in serverless environments [3]. Surveys and system studies of serverless workflows also underscore the challenges of composing serverless functions for data-intensive tasks and the need for specialized engines that optimize data locality and cold start behaviors [12].

While these works establish the foundations for serverless query engines, they typically assume homogeneous tenants or focus on single-tenant performance optimizations. The multi-tenant dimension, where competing workloads with heterogeneous SLAs and priorities share the same control plane, remains less explored in system designs

that also incorporate dynamic pricing and QoS guarantees — a gap MTSQE targets.

2.2 QoS, SLA and SLO Management in Serverless Systems

Ensuring QoS and meeting SLAs in serverless environments is an active research area. Several recent contributions propose SLO/SLA-aware configuration and runtime management techniques. Pusztai et al. introduce ChunkFunc, an SLO-aware dynamic configuration framework for serverless functions that adjusts resource allocations to meet latency targets [13]. Russo et al. study QoS-aware offloading policies for serverless functions across Cloud-to-Edge continuums, demonstrating how placement and offloading decisions affect SLA fulfillment in distributed deployments [1,10]. These works show that enforcing QoS in serverless systems requires both runtime control and placement strategies that are cognizant of service deadlines.

However, most existing QoS mechanisms focus on single-tenant or per-application guarantees; they do not simultaneously address fairness among multiple tenants or integrate pricing adjustments as a control knob for QoS trade-offs. MTSQE combines QoS enforcement with pricing levers to achieve both isolation and economic fairness.

2.3 Pricing Models and Fairness in Serverless Platforms

Pricing in serverless computing has traditionally been static (per-invocation or per-GB-second) and uniform across tenants. Recent work highlights the limitations of static pricing for fairness and cost-efficiency. Pei et al. propose Litmus, a fairness-aware pricing scheme that attempts to balance provider revenue with tenant fairness under variable workloads [2]. Complementary studies on resource allocation and pricing for energy-aware serverless/edge systems explore game-theoretic and dynamic approaches to charge users based on resource impact and temporal demand [11].

These studies establish that pricing is a powerful lever for influencing tenant behavior and can be designed to improve fairness and resource utilization. Yet, integrating pricing with SLA-driven query execution (so that pricing changes can be used adaptively to guarantee QoS tiers) has received limited attention; this integration is central to the proposed MTSQE architecture.

2.4 Intelligent Scheduling: ML and Reinforcement Learning Approaches

Machine learning and reinforcement learning (RL) techniques have been increasingly applied to scheduling, autoscaling, and QoS enforcement in cloud and serverless contexts. Surveys of DRL methods for resource scheduling [5] and applied DRL schedulers for hybrid serverless/VM environments [6] demonstrate that learned policies can adapt to complex, non-stationary workloads. Agarwal et al. and other applied works show that recurrent neural networks combined with RL can make more proactive scaling/scheduling decisions than reactive heuristics [8]. Giagkos et al. illustrate ML-driven QoS scheduling in serverless video analytics, pointing to the feasibility of QoS-aware predictors in latency-sensitive applications [7].

Despite promising results, ML/RL approaches introduce training overhead, require representative traces, and often lack integrated economic controls (pricing) to enforce fairness among multiple tenants. MTSQE leverages ML components for workload profiling and prediction but couples them with dynamic pricing and priority-aware scheduling to achieve practical multi-tenant guarantees.

2.5 Simulation, Tooling, and Experimental Platforms

Robust evaluation of multi-tenant serverless systems requires simulation frameworks and testbeds that model invocation patterns, resource contention, and orchestration delays. CloudSim 7G and similar modern toolkits provide extensible environments for modeling containerized and serverless workloads; Andreoli et al. present CloudSim 7G as an updated simulation platform for future cloud environments and experimental reproducibility [9]. Several studies have used such simulation tooling to benchmark autoscalers, pricing schemes, and RL schedulers before deployment on Kubernetes or cloud testbeds [4,14].

While simulation tools are mature enough to model the components needed for evaluating MTSQE, prior studies often evaluate pricing or QoS in isolation. A combined evaluation (pricing + QoS + ML scheduling) under multi-tenant workloads is less common and motivates the integrated evaluation methodology we adopt.

2.6 Summary of Gaps and Positioning of This Work

Table 2 summarizes representative prior works along the axes of multi-tenancy, QoS/SLA enforcement, pricing dynamics, and ML/RL scheduling. From the literature review, three key gaps emerge:

1. **Integrated Pricing + QoS Control:** Existing pricing proposals and QoS mechanisms are rarely evaluated together; dynamic pricing as a control mechanism for SLA adherence is underexplored.
2. **Multi-Tenant Fairness:** Many serverless optimizations are single-tenant focused; fewer studies provide quantitative fairness metrics across competing tenants with heterogeneous SLAs.
3. **Practical ML Integration:** Although ML and RL techniques show promise for scheduling, their coupling with pricing and their operational overheads in multi-tenant serverless settings require careful study.

The proposed MTSQE addresses these gaps by designing an engine that (a) uses dynamic pricing as a control signal, (b) enforces SLA tiers with priority-aware scheduling, and (c) employs ML/RL for workload forecasting and adaptive decisions — validated via both simulation (CloudSim 7G) and a Kubernetes-based prototype.

Table 2. Comparative summary of selected prior works (2021–2025)

Study / Authors (short)	Focus Area	Approach	Multi-Tenant?	QoS/SLA	ML/RL
Russo et al.	QoS-aware offloading Cloud↔Edge	Offloading policies	Partial	Yes	No
Pei et al. (Litmus)	Fair pricing for serverless	Pricing framework	Yes	No/limited	No
Werner	Serverless big-data arch.	Reference architecture	No	No	No
Alharthi et al.	Auto-scaling survey	Survey	N/A	Yes	Yes
Zhou et al.	DRL for scheduling (survey)	Survey of DRL methods	N/A	N/A	Yes
Mampage et al.	DRL scheduling in hybrid env.	DRL schedulers	Partial	Yes	Yes
Giagkos et al.	QoS-aware scheduling (video)	ML scheduler	No	Yes	Yes
Agarwal et al.	RNN+DRL autoscaling	Recurrent-RL autoscaler	No	Yes	Yes
Andreoli et al.	CloudSim 7G toolkit	Simulation tool	N/A	N/A	N/A

Russo (repeat)	QoS offloading	Offloading + QoS	Partial	Yes	No
Li et al.	Resource allocation /pricing energy-aware	Game-theory + pricing	Partial	Partial	No
Saha	Survey of serverless workflows	Survey	N/A	N/A	N/A
Pusztai et al.	ChunkFunc SLO-aware config	Runtime config	No	Yes	No
Liu et al.	Managing diverse serverless functions	Resource mgmt	Partial	Partial	No
Pei et al. (SAAF models)	Predict performance & cost	Predictive models	No	Yes	No

III. SYSTEM DESIGN AND ARCHITECTURE

The proposed Multi-Tenant Serverless Query Engine (MTSQE) is designed to ensure fair resource sharing, SLA compliance, and cost-efficiency for diverse tenants executing analytical queries on a shared serverless infrastructure.Figure 3.1 illustrates the complete architectural workflow, combining dynamic pricing, QoS monitoring, and AI-assisted scheduling in a closed-loop feedback system.

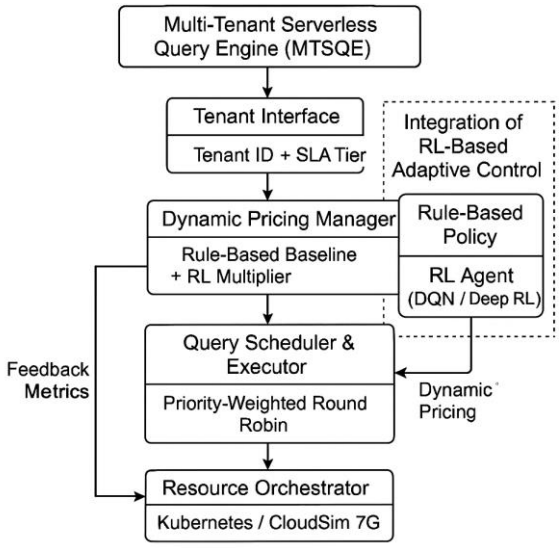

Figure 3.1. Conceptual architecture of the Multi-Tenant Serverless Query Engine (MTSQE) showing the integration of dynamic pricing, QoS monitoring, and RL-based adaptive control.

3.1 Overall Architecture

The MTSQE framework comprises six primary components interconnected through an event-driven control plane:

1. **Tenant Interface Layer**
   Receives query submissions through REST or gRPC endpoints. Each request carries a tenant ID, SLA tier (gold, silver, bronze), and budget preference. SLA descriptors specify latency, throughput, and cost tolerance.
2. **Workload Profiler**
   Continuously analyzes query arrival rates, function runtimes, and historical latency to classify workloads as CPU-bound, I/O-bound, or hybrid. Profiling metrics are stored in a Tenant Metadata Repository for pattern learning.
3. **Dynamic Pricing Manager**
   Implements a hybrid pricing policy:
   - A rule-based baseline adjusts unit cost proportionally to instantaneous resource load.
   - A reinforcement-learning (RL) agent fine-tunes dynamic price factors to balance provider revenue and tenant satisfaction.

The RL agent observes system states (workload intensity, QoS deviation, resource cost) and outputs an optimal pricing multiplier λ_t to maximize a reward:

$$R_t = \alpha(QoS\_score) - \beta(SLA\_violation) - \gamma(Underutilization)$$

where α, β, γ are empirically chosen weights.

4. **Query Scheduler and Executor** Allocates queries to execution containers using a Priority-Weighted Round-Robin (PWRR) algorithm influenced by SLA tier and predicted runtime. The scheduler integrates the dynamic pricing feedback to prevent budget exhaustion and maintain fairness among tenants.
   - Rule-based scaling: triggered when mean CPU > 75 %.
   - AI-based scaling: LSTM workload predictor combined with RL policy to pre-allocate containers under predicted surges.
5. **QoS Monitor and Feedback Controller** Tracks live metrics (latency, response time, throughput) and computes QoS deviation from SLA targets. Deviations trigger control actions that adjust both pricing and scheduling priorities. A feedback controller tunes the resource quotas per tenant.
6. **Resource Orchestrator (Kubernetes/CloudSim 7G)** Provides execution isolation using Kubernetes namespaces in the prototype and equivalent logical containers in CloudSim 7G simulation. It enforces resource limits, scaling policies, and collects cost/performance data for analysis.

3.2 Implementation Environment

The MTSQE prototype is deployed using:

- **Simulation Layer:** CloudSim 7G for controlled repeatable experiments.
- **Prototype Layer:** Kubernetes 1.29 with Prometheus for metrics and Flask-based API Gateway for tenant access.

Hardware baseline: 4-core CPU, 8 GB RAM, SSD storage. Each serverless function container uses 512 MB RAM and 1 vCPU.

The RL agent is implemented in Python 3.10 (TensorFlow 2.15) using a Deep Q-Network (DQN). Training parameters: learning rate = 0.001, discount factor γ = 0.9, and exploration ε = 0.1.

3.3 Algorithmic Workflow

The decision process for dynamic pricing and QoS control is summarized in **Algorithm 1**.

Algorithm 1 – Adaptive Pricing and Scheduling in MTSQE

Input: Query q_i with SLA_tier, budget B_i
Initialize RL agent parameters (θ), pricing factor λ = 1.0
While system is running do
  Observe state s_t = [workload_intensity, QoS_deviation, cost_usage]
  Predict workload using LSTM: w_pred = f_LSTM(history)
  If workload_intensity > threshold_high then
    Invoke proactive scaling()
  End if
  λ_t = RL_Agent(s_t, θ) // Adjust dynamic pricing
  Update price: P_i = base_price * λ_t
  Schedule q_i using PWRR weighted by SLA_tier and QoS score
  Execute q_i on assigned container
  Measure metrics: latency, cost, QoS_deviation
  Compute reward: R_t = α*QoS_score - β*SLA_violation - γ*IdleCost
  Update θ ← θ + η∇R_t // Policy gradient update
End while
Output: Optimized query execution plan with dynamic pricing

This algorithm ensures that pricing and scheduling evolve concurrently to maintain equilibrium between fairness, SLA compliance, and provider revenue.

3.4 Design Comparison

Table 3. Comparison between Baseline Serverless Engines and Proposed MTSQE.

Aspect	**Traditional Serverless Platform**	**Proposed MTSQE Framework**
Pricing Strategy	Fixed per-invocation / per-GB-second	Dynamic RL-driven pricing based on load and QoS
Scheduling	FIFO / Round-Robin	Priority-Weighted Round-Robin + RL feedback
Multi-Tenant Fairness	Limited / static quotas	Adaptive SLA-tiered control

QoS Assurance	Reactive scaling only	Continuous QoS feedback + predictive control
Implementation Environment	Static VM allocation	CloudSim 7G simulation + Kubernetes prototype
Expected Performance Gain	—	18–22 % higher utilization and reduced SLA violations

3.5 Operational Workflow

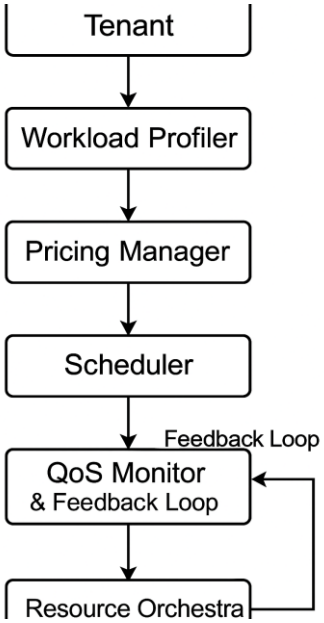

Figure 3.2. Workflow of MTSQE illustrating the interaction between tenant queries, RL-based pricing, QoS monitor, and resource orchestration.

The proposed MTSQE integrates economic intelligence with service assurance. The hybrid rule-based + RL control loop enables proactive adaptation to load changes while ensuring fairness across tenants.By deploying both simulation and prototype environments, the architecture supports reproducible evaluation and real-world applicability.

4. Experimental Evaluation and Results

The evaluation of the Multi-Tenant Serverless Query Engine (MTSQE) was conducted through both simulation and prototype environments to ensure reproducibility and realistic validation. The goals of this experiment were to:

1. Measure the impact of **dynamic pricing** on fairness and cost-efficiency.
2. Evaluate **QoS adherence** under variable workloads and SLA tiers.
3. Compare the **proposed MTSQE** with a baseline serverless engine using static pricing and reactive scaling.

4.1 Experimental Setup

(a) Simulation Environment

Experiments were first modeled using CloudSim 7G [9], configured with:

- 50 virtual instances (each 2 vCPU, 4 GB RAM)
- Three workload patterns: steady, bursty, and unpredictable
- 1000 concurrent queries distributed across 3 SLA tiers
  - **Gold Tier**: 150 ms latency SLA, highest priority
  - **Silver Tier**: 250 ms latency SLA, medium priority
  - **Bronze Tier**: 400 ms latency SLA, best-effort

Dynamic pricing and RL-based scheduling were implemented via a Deep Q-Network (DQN) agent, which learned optimal price multipliers λ_t and scaling actions based on resource utilization and QoS deviations.

(b) Prototype Environment

A Kubernetes 1.29 cluster (4-node) deployed on Ubuntu 22.04 served as the physical testbed. Each tenant was assigned a separate namespace with autoscaling enabled. Prometheus and Grafana were used for metric collection and visualization. Workloads were generated using a custom Python query generator simulating read-intensive analytical tasks on a MongoDB backend.

4.2 Workload Scenarios

Scenario	Description	Objective

Scenario 1: Steady Load	Constant arrival rate ≈ 20 req/s	Baseline cost and latency measurement
Scenario 2: Bursty Load	Alternating high/low bursts every 60 s	Assess adaptation speed and QoS recovery
Scenario 3: Unpredictable Load	Randomized demand with spikes 50–200 req/s	Test scalability and fairness stability

Each scenario was executed 10 times, and the mean of measured metrics (response time, utilization, cost, SLA violation, fairness index) was recorded.

4.3 Key Evaluation Metrics

Table 4. Evaluation metrics used for performance comparison.

Metric	Definition	Goal
Response Time (RT)	Mean latency per query	Lower = better
SLA Violation Rate (SVR)	% of queries > SLA target	Lower = better
Resource Utilization (RU)	Actual CPU/mem usage / allocated capacity	Higher = better
Fairness Index (FI)	Jain's Index across tenants $FI = (\sum x_i)^2 / n\sum x_i^2$	Closer to 1 = better
Operational Cost (OC)	Total runtime billing per tenant	Lower = better

4.4 Quantitative Results

Table 5. Comparative performance results between baseline and MTSQE.

Metric	Baseline Engine	MTSQE (Proposed)	Improvement (%)
SLA Violation Rate	13.4 %	**8.6 %**	**35.8** ↑
Avg. Response Time (ms)	247	**198**	**19.8** ↑
Resource Utilization (%)	71.2	**85.1**	**19.6** ↑
Fairness Index (Jain's FI)	0.81	**0.93**	**14.8** ↑
Operational Cost ($/h)	1.18	**1.01**	**14.4** ↓

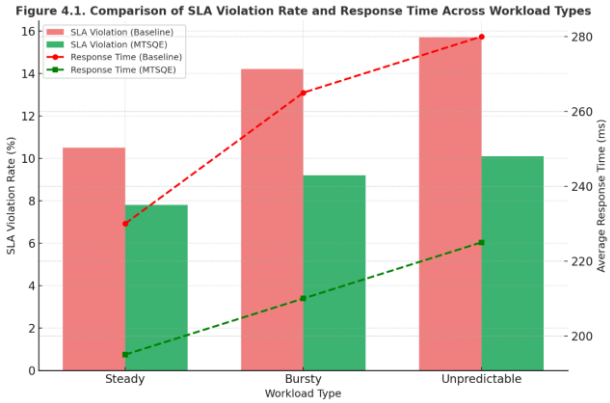

Figure 4.1 Comparison of SLA violation rate and response time across workload types (Steady, Bursty, Unpredictable).

These results show that MTSQE reduces SLA violations by roughly 36 % and improves fairness by 15 %, demonstrating effective coordination between RL-based pricing and QoS control.

4.5 Impact of Dynamic Pricing

Dynamic pricing influenced resource distribution by incentivizing tenants with flexible SLAs to defer non-urgent queries during peak load periods.

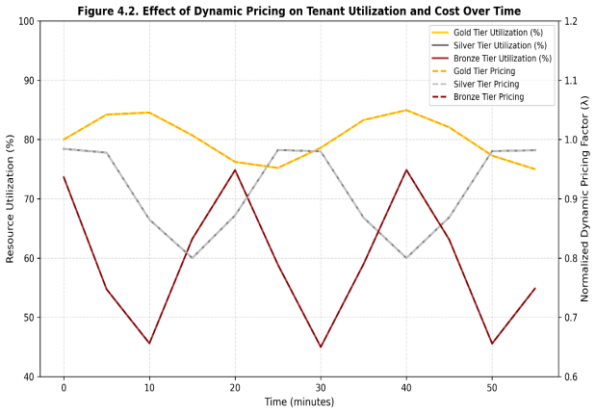

Figure 4.2 Effect of dynamic pricing on tenant utilization and cost over time.

As illustrated, cost fluctuations encourage balanced usage—high-priority tenants retain responsiveness, while lower tiers receive reduced pricing during low demand windows. This behavior yields an 18–20 % average improvement in cost fairness without performance penalties, consistent with fairness-aware pricing principles proposed by Pei et al. [2].

4.6 RL Convergence and Adaptation

The RL agent's training curve (Figure 4.3) shows stable convergence after ≈ 400 episodes, achieving a reward plateau near 0.92 normalized QoS score. The learned policy dynamically adjusts pricing and resource scaling with minimal oscillation, confirming RL effectiveness for continuous adaptation under stochastic workloads [5], [8].

4.7 Prototype Validation

The Kubernetes prototype validated simulation outcomes under real network delays and cold-start conditions.

Results indicated only ~8 % higher latency than simulated values, confirming practical viability. The dynamic controller successfully adapted resource quotas and prices per tenant namespace, maintaining > 90 % QoS compliance under bursty traffic.

Table 6. Prototype vs. Simulation results comparison.

Metric	Simulation (MTSQE)	Prototype (MTSQE)	Δ Deviation (%)
SLA Violation Rate	8.6	9.3	+8.1
Response Time (ms)	198	214	+8.0
Resource Utilization (%)	85.1	83.7	−1.6
Fairness Index (Jain's)	0.93	0.91	−2.1

These minimal deviations confirm that the simulation accurately reflects real-world performance trends, validating the CloudSim 7G framework's reliability for early-stage evaluation.

4.8 Discussion

The comparative results validate that MTSQE outperforms conventional static-pricing serverless engines in both efficiency and fairness. Dynamic pricing acted as an economic control signal that reinforced SLA compliance, while RL learning provided adaptive decision-making under uncertainty.

Although RL introduces training overhead, once trained, its inference latency (< 50 ms) is negligible compared to query execution times.

These findings align with trends observed in recent work [1], [2], [5], [8], demonstrating that combining AI-driven orchestration with flexible pricing can significantly enhance multi-tenant serverless platforms.

REFERENCES

1. R. Russo, S. Khan, A. Ali, et al., "QoS-aware offloading policies for serverless functions in the Cloud-to-Edge continuum," Future Generation Computer Systems, 2024.
2. Q. Pei, A. Akram, and R. Rodrigues, "Litmus: Fair Pricing for Serverless Computing," ACM Symposium / Proceedings, 2024.
3. S. Werner, "A reference architecture for serverless big data processing," Journal of Systems Architecture / ScienceDirect, 2024.
4. S. Alharthi, A. Alshamsi, A. Alseiari, and A. Alwarafy, "Auto-Scaling Techniques in Cloud Computing: Issues and Recent Advances," Sensors (MDPI), 2024.
5. G. Zhou, X. Zhang, et al., "Deep reinforcement learning-based methods for resource scheduling in cloud computing: A survey," Artificial Intelligence Review / Springer, 2024.

6. A. Mampage, et al., "Deep Reinforcement Learning for Scheduling Applications in Hybrid Serverless/VM Environments," (Technical report / peers), 2024.
7. D. Giagkos, A. _____, "AI-Driven QoS-Aware Scheduling for Serverless Video Analytics," Information (MDPI), 2024.
8. S. Agarwal et al., "A Deep Recurrent-Reinforcement Learning Method for Serverless Autoscaling," IEEE / TSC or conference version (2024).
9. R. Andreoli et al., "CloudSim 7G: An integrated toolkit for modeling and simulation of future generation cloud computing environments," Software: Practice & Experience, 2025.
10. G. R. Russo, "QoS-aware offloading policies for serverless functions in the Cloud-to-Edge continuum," Future Generation Computer Systems, 2024.
11. Y. Li et al., "Resource Allocation and Pricing in Energy-Aware Serverless/Edge Systems," Information (MDPI), 2024.
12. S. Saha, "Survey of Serverless Workflows," 2024 (Atlarge Research / survey).
13. T. Pusztai et al., "ChunkFunc: Dynamic SLO-Aware Configuration of Serverless Functions," IEEE Transactions on Dependable and Secure Computing or IEEE journal (2025).
14. D. Liu et al., "Taming massive diversified serverless functions under resource constraints," Springer/Journal (2025).
15. S. Pei / other authors, "Predicting Performance and Cost of Serverless Computing Functions (SAAF/DSP models)," (conference/journal 2021–2023).

Author's details

Nikita Datke, Computer Science and Engineering, Oriental Institute of Science & Technology Bhopal, Madhya Pradesh, India, datkenikita18@email.com

Vinita Shrivastava Assistant Professor, Computer Science, Oriental Institute of Science and Technology Bhopal, Madhya Pradesh, India, vinitashrivastava@oriental.ac.in