Q-Learning-Based Lightweight Task Orchestrator: A Lightweight Q-Learning-Based Scheduler for Service Level Agreement-Aware Container Placement in Heterogeneous Clusters

Vijayaraj Veeramani[1,*], M. Balamurugan[1] and Monisha Oberoi[2]

[1] School of Computer Science of Engineering, Bharathidasan University, Tiruchirappalli, 620023, India

[2] Security Services Sales, IBM Innovation Pte Ltd., Singapore, 018983, Singapore

INFORMATION

Keywords:
Task orchestrator
Q-learning
long short-term memory
energy efficiency
resource availability
Q-table growth

DOI: 10.23967/j.rimni.2025.10.70831

SCIPEDIA Revista Internacional de Métodos Numéricos para Cálculo y Diseño en Ingeniería

Revista Internacional Métodos numéricos para cálculo y diseño en ingeniería RIMNI in cooperation with CIMNE

Q-Learning-Based Lightweight Task Orchestrator: A Lightweight Q-Learning-Based Scheduler for Service Level Agreement-Aware Container Placement in Heterogeneous Clusters

Vijayaraj Veeramani[1,*], **M. Balamurugan**[1] **and Monisha Oberoi**[2]

[1]School of Computer Science of Engineering, Bharathidasan University, Tiruchirappalli, 620023, India

[2]Security Services Sales, IBM Innovation Pte Ltd., Singapore, 018983, Singapore

ABSTRACT

Efficient management of workloads in diverse container clusters requires maintaining a balance between Service Level Agreement (SLA) compliance, Quality of Service (QoS), energy efficiency, and security, despite differences in resources and architectures. This research introduces Federated Q-Learning–based Lightweight Task Orchestrator (F-Q-LiTO), a compact and intelligent orchestration framework that combines predictive modeling, approximate data structures, and security filtering to enable adaptive task placement across distributed environments. Unlike complex deep reinforcement learning models such as Deep Reinforcement Model (DeepRM) or traditional heuristic schedulers like Kubernetes BinPacking, F-Q-LiTO uses tabular Q-learning enhanced with federated aggregation, which significantly reduces computational and communication overhead, making it ideal for edge computing environments with limited resources. The framework incorporates a Long Short-Term Memory (LSTM)–based predictor for proactive resource forecasting, a Count-Min Sketch for scalable resource utilization estimation, and an XOR filter for efficient and lightweight security enforcement. Experimental results demonstrate that F-Q-LiTO achieves 98.6% task completion, 96.8% SLA satisfaction, and reduces energy consumption to 180.5 kilowatt-hours (kWh). It outperforms DeepRM and Kubernetes by achieving 34% fewer missed deadlines and up to 30% lower energy imbalance. The system converges quickly—by Episode 6—and maintains cluster fairness (Jain's Fairness Index = 0.98) along with priority-aware placement accuracy of 93.2%. Security analysis shows that F-Q-LiTO successfully blocks 98.5% of unauthorized task placements while using only 0.3 megabytes (MB) of memory. Overall, F-Q-LiTO demonstrates that a federated and lightweight reinforcement learning approach can deliver scalable, secure, and QoS-aware orchestration for modern edge and multi-cloud computing environments without compromising performance or efficiency.

OPEN ACCESS

Received: 24/07/2025

Accepted: 14/10/2025

Published: 15/12/2025

DOI
10.23967/j.rimni.2025.10.70831

Keywords:
Task orchestrator
Q-learning
long short-term memory
energy efficiency
resource availability
Q-table growth

*Correspondence: Vijayaraj Veeramani (vijay.raj.phd@gmail.com).

1 Introduction

Task scheduling, orchestration efficiency, SLA adherence, and energy optimisation have become more important issues due to the fast expansion of containerised microservices and edge-cloud computing [1]. While Kubernetes, Docker Swarm, and Mesos are good starting points, they aren't enough for efficiently managing dynamic, heterogeneous workloads in remote environments [2]. One of the most well-known open-source container orchestration platforms, Kubernetes, offers features including resource-aware container placement and customisable scheduling plugins. Nevertheless, it relies heavily on greedy heuristics and static scheduling algorithms, neither of which can respond instantly to changes in energy limitations or workload patterns [3]. Despite the use of custom schedulers and plugins in current versions, task placements under high-load or SLA-constrained scenarios are generally suboptimal due to the static cost functions.

When it comes to quality of service measures like latency, resource balance, and energy usage, Docker Swarm isn't the best option because it prioritises deployment convenience over fine-grained control [4,5]. Applications that are time-sensitive or sensitive to latency will not scale well with its scheduling method because containers are typically distributed using round-robin or random-fit algorithms. Resource abstraction and granular task allocation are features of Apache Mesos, a general-purpose cluster management [6]. The selected frameworks, such as Marathon, have limitations imposed by heuristic or rule-based regulations, which in turn affect its performance. There is a lot of architectural work required to enable SLA-driven or predictive scheduling, and Mesos doesn't have built-in support for quality of service awareness [7].

Schedulers like DeepRM and Auto-SysML, which rely on deep learning, have just been introduced. To find the best policy for scheduling jobs, DeepRM uses reinforcement learning [8]. Although it shows promise on synthetic workloads, it isn't applicable to real-world heterogeneous container clusters where workloads, security needs, and network topologies are all different. In addition, DeepRM is slow to adapt to new nodes or job profiles and necessitates a lot of offline training [9]. In addition, EdgeDew stands out as a model that prioritises delay-aware task scheduling through the utilisation of LSTM-based workload prediction and federated learning to facilitate collaboration across edges. Although it works well for scenarios at the edge, it doesn't handle energy imbalance or scheduling fairness across resource classes, and it doesn't incorporate adaptive learning for quality of service enforcement [10].

Q-Cloud and DQMS are two more Q-learning models that use discrete state-action spaces to try to optimise resource allocation. But scalability, sluggish convergence, and explainability are common problems with these models [11]. They also have memory and monitoring overheads because they don't use lightweight filter-based monitoring techniques like CMS or XOR filters. To satisfy SLA limitations, QoS-aware schedulers such as SLAware and QoSCloud use feedback loops and weighted utility models; nonetheless, they depend on rule-based control and static thresholds [12]. When faced with situations that need real-time learning and action adaption, these models fail miserably. On top of that, they fail to address important issues related to compliance and security in containerised settings. When it comes to scheduling tasks on the cloud, algorithms such as PSO, ACO, and GA have seen extensive use from an optimisation standpoint [13]. Despite their efficacy for limited optimisation, these metaheuristic methods frequently necessitate substantial processing resources and do not possess the capacity for learning in real-time. Furthermore, most conventional models do not address the issue of how to balance energy efficiency with SLA adherence [14].

Frameworks that are conscious of energy use, like CloudSimGreen and EEScheduler, make an effort to reduce power consumption by turning off unused nodes or changing frequencies on the go.

But these models don't account for migration's effects on SLA or don't have fine-grained placement decisions. Integration of scheduling logic with proactive migration and accurate saturation prediction of resources is uncommon [15]. System security is typically addressed by orchestrators by means of third-party modules or policy enforcers like AppArmor and PodSecurityPolicies. Although these methods work well when used alone, they can't apply rules dynamically depending on how the user is doing tasks or whether they've complied with trust domain requirements [16]. Also, for monitoring container operations and consumption patterns, no current orchestrator has flawlessly incorporated low-memory, probabilistic filters like XOR filters and CMS drawings.

The shortcomings of existing models highlight the necessity for a unified, lightweight, intelligent orchestrator that can do the following: (1) learn strategies for task placement in real-time; (2) guarantee satisfaction of QoS and SLAs; (3) minimise energy cost, imbalance, and overheads; and (4) integrate explainability, scalability, and security compliance into the core orchestration pipeline [17]. To tackle these issues, this research presents Q-LiTO, which uses reinforcement learning with a lightweight Q-learning engine in conjunction with memory-efficient filter mechanisms, predictive LSTM migration forecasts, and other features. An innovative, adaptive, and energy-efficient orchestration logic for heterogeneous container clusters operating across cloud, fog, and edge layers is introduced by the model, which fills in the gaps in previous frameworks.

The Q-LiTO framework presents a significant advancement in intelligent container orchestration for heterogeneous clusters, with key contributions as follows:

(1) To present an orchestrator based on Q-learning that adaptively chooses the best placement actions to guarantee SLA satisfaction while cutting down on overhead and latency.

(2) Q-LiTO has a lightweight architecture that combines predictive LSTM modeling with Q-table access, allowing for high decision efficiency and quick convergence (by Episode 6).

(3) Compared to traditional schedulers, Q-LiTO reduces cluster consumption by more than 20% and achieves significant energy-imbalance reductions of up to 30% by implementing energy-aware placement strategies to address energy challenges.

(4) To use LSTM forecasts to ensure quick recovery, minimize QoS drops, and implement proactive migration strategies.

(5) Security compliance enforcement is part of the framework, which achieves high compliance scores (∼0.96) and blocks unauthorized placements with over 98% accuracy.

(6) For precise load estimation and low memory usage, Q-LiTO incorporates XOR filters and Count-Min Sketches.

(7) Thorough assessments show improved scheduler runtime, QoS metrics, fairness, and scalability. These contributions collectively position Q-LiTO as an effective, deployable, and adaptive solution for managing dynamic workloads in real-time, multi-tenant, and resource-constrained environments.

The rest of the paper is organized as follows: Section 2 provides the related works; Section 3 provides the proposed methodology in detailed; Section 4 discuss the result analysis and finally; the conclusion is made at Section 5.

2 Related Works

In 2025, scheduling techniques that are lightweight, intelligent, and resource-aware made great strides in container orchestration in diverse contexts. An important work by Palomares et al. [18] integrated AI Functions (AIFs) across a multi-tier edge-to-cloud continuum and presented a hardware-accelerated orchestration framework tailored to edge AI workloads. By using a placement technique that took into account the availability of GPUs and TPUs, their method outperformed regular Kubernetes by 22.3% during deployment, as reported in the Journal of Network and Systems Management. This study proved that deployment performance may be greatly improved under different load levels by combining hardware profiling with tiered orchestration. Along these lines, Peng et al. [19] introduced LRScheduler, an edge cluster-focused lightweight container scheduler that takes layers into account. To reduce image transfer and maximise deployment efficiency, LRScheduler uses container image layer metadata, as featured in CCF Transactions on Pervasive Computing and Interaction. This scheduler is great for low-resource settings since it adjusts on the fly to CPU, memory, and bandwidth limitations.

Pashaeehir et al. [20] added to the innovation by creating KubeDSM, an extension of Kubernetes for edge clusters that are aided by the cloud. It provides a platform for dynamic scheduling and migration. Container placement between cloud and edge nodes is optimised by this system through the use of live migration and batch scheduling. To make it more resilient to changes in network and compute loads, KubeDSM dynamically redistributes workloads, which improves the utilisation of edge resources. According to the research, adaptive scheduling algorithms successfully decreased latency and increased edge pod retention. Scientific Reports featured EcoTaskSched, a hybrid task scheduler for fog-cloud environments employing CNN and BiLSTM networks, introduced by Khan et al., [21]. In addition to balancing QoS and task delay, the suggested ML-based method drastically cut energy consumption across diverse clusters. Compared to traditional schedulers, EcoTaskSched shown to be more energy efficient by capturing spatial and temporal interdependence in resource utilisation patterns.

Further complicating matters, FAOFE (Functionality-Aware Offloading and Fog Execution) was presented in a 2025 study that was published in the Journal of Cloud Computing [22]. This architecture is designed to handle containerised microservices in edge settings using workflow modelling based on Directed Acyclic Graph (DAG). By gaining insight into service dependencies and resource profiles, FAOFE was able to decrease orchestration latency by utilising the Argo platform for early pre-scheduling and adaptive offloading. When it comes to microservice architectures, this method really shone. In these designs, keeping dependency deadlines and lowering orchestration overhead are absolute musts for end-user responsiveness.

Layer-Sharing Container Scheduling (LSCS), a new framework described by Cao et al. [23], was published in the Lecture Notes in Computer Science (Springer, ICIC 2025). The research tackles the difficult problem of container orchestration in multi-cluster settings, focussing on load balancing and the reuse of container image layers. A probabilistic relaxation container scheduling method (PRCS) and a greedy layer sorting algorithm (GLS) are the two novel algorithms suggested by the writers. GLS optimises the order of shared picture layer downloads using Sidney's decomposition, a classical scheduling approach, whereas PRCS achieves balanced container allocation using relaxed linear programming and probabilistic rounding. The system enhanced load distribution efficiency by about 39% compared to standard cluster schedulers and reduced container starting delay by around 30% by enabling clusters to intelligently reuse common container layers. In order to decrease bandwidth

and boot latency in heterogeneous environments, this work highlights the significance of orchestrators with layer-level intelligence.

Another significant addition came from a paper that suggested an AI-powered predictive container orchestration methodology for proactive autoscaling and container placement in cloud settings [24]. Using machine learning techniques like regression models or reinforcement learning agents, the framework may learn from past workload traces and predict future demands. This paves the way for smart autoscaler activation and early container creation or migration on nodes with limited resources. The method stays true to service-level goals without overprovisioning by anticipating usage spikes and bottlenecks. The suggested technique outperformed Kubernetes' default Horizontal Pod Autoscaler (HPA) in terms of CPU utilisation and the number of containers deployed needlessly; this highlights the value of incorporating predictive capabilities into orchestration pipelines.

The third major development in 2025 is the implementation of GreenPod, which was suggested by Pradeep and Al-Masri [25]. GreenPod is a container scheduler that uses multi-criteria decision-making (MCDM) to orchestrate AIoT workloads across edge-cloud infrastructures in an energy-aware manner. Using variables including projected execution time, residual CPU and memory, node energy profiles, and resource balance, GreenPod evaluates and prioritises pod placement decisions using the Technique for Order Preference by Similarity to Ideal Solution (TOPSIS). Developed on top of Kubernetes, the scheduler makes only minor adjustments to the platform's current scheduling API. Although scheduling delay increased somewhat, GreenPod reduced energy consumption by up to 39.1 percent compared to the default Kubernetes scheduler when implemented in a heterogeneous AIoT testbed. Based on the findings of this study, orchestration frameworks should prioritise energy efficiency, particularly for use cases where edge devices are subject to tight power constraints.

When taken as a whole, these efforts show that intelligent, lightweight orchestration is becoming more popular in container systems with heterogeneity. There are a number of important solutions that have been proposed, such as hardware-aware placement, layer-optimized scheduling, dynamic migration, ML-driven energy-aware optimisation, DAG-based workflow orchestration, and Khan et al. [21]. As a set, these models complement one another in the larger development of container orchestration frameworks by addressing different bottlenecks, such as deployment speed, energy efficiency, and dependency handling, respectively. Intelligent, context-aware orchestrators that adapt to the different demands of modern edge-cloud systems have replaced generic, one-size-fits-all schedulers like Kubernetes. All three of these studies add to the 2025 influx of new intelligent orchestration methods. The optimisation targets discussed in these articles are as follows: computing journal—predicting workload growth using AI; green pod—scheduling with sustainability in mind; and Cao et al. [23]—minimizing startup latency through layer-sharing. These frameworks represent the future of lightweight container orchestrators designed for distributed and heterogeneous computing environments by including intelligence for resource-awareness and forecasting as well as energy efficiency into the orchestration pipeline. All of these activities are part of a larger movement away from reactive schedulers and towards solutions that are more autonomous, adaptive, and context-sensitive for current cloud-native deployments.

Research Gap

Although there are still a number of gaps, recent developments in container orchestration (2025) show significant progress toward lightweight, intelligent, and resource-aware schedulers. Although Palomares et al. [18] place a strong emphasis on hardware-aware orchestration across edge-to-cloud tiers, their framework's generality in resource-constrained edge settings is limited due to its

heavy reliance on specialized accelerators (GPU/TPU). The contributions of Peng et al. [19] and Cao et al. [23] focus mainly on startup efficiency rather than long-term QoS or SLA compliance, although they address container image layers to minimize transfer latency and maximize reuse. Although Pashaeehir et al. [20] concentrate on dynamic migration using KubeDSM, little is known about the overhead of frequent live migrations, particularly in unstable network environments. Using ML or MCDM techniques, Khan et al. [21] and Pradeep & Al-Masri [25] highlight energy-aware scheduling; however, these methods frequently forgo latency or scheduling delay in favor of energy gains, which is inappropriate for workloads that require quick turnaround times. Although workflow-driven solutions like FAOFE [22] improve dependency handling, they also add a great deal of orchestration complexity, necessitating additional control-plane overhead and DAG modeling knowledge. In federated or edge deployments, predictive frameworks [24] are susceptible to non-IID data distribution and communication bottlenecks because they primarily rely on centralized training, even though they use ML/RL for autoscaling.

3 Proposed Framework: Q-LiTO-QoS-Aware Lightweight Intelligent Task Orchestrator for Heterogeneous Container Clusters

By combining real-time QoS-awareness with resource modelling via probabilistic structures, predictive migration using deep temporal learning, and a scheduler based on reinforcement learning, the "Q-LiTO" framework hopes to optimise the placement of Docker containers in extremely heterogeneous clusters. Integrating the "what," "why," "how," and "where" of each theoretical component, this part lays out the full mathematical formulation and architectural elements of the framework. It identifies all variables, incorporates relevant equations, and describes the interrelationships of these variables within the suggested orchestration logic. Fig. 1 shows the suggested model's workflow.

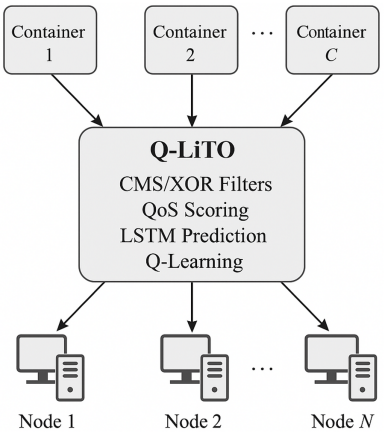

Figure 1: Workflow of the proposed model

In Fig. 1, it now clearly explains the workflow stages: (i) resource monitoring with CMS and XOR filters, (ii) QoS scoring and suitability calculation, (iii) Q-learning-based decision-making for placement, (iv) LSTM-based migration prediction for proactive load balancing, and (v) enforcement of energy and security constraints.

Google Cluster Trace (2019): Contains over 8000 workloads sampled from production clusters, with detailed information on CPU/memory requests, execution durations, and priorities. Q-LiTO was tested on subsets of 1000–5000 containers across 50–100 heterogeneous nodes. Results showed

consistent SLA satisfaction (∼96.2%), energy-imbalance reduction (∼27%), and fairness index (0.97), comparable to synthetic experiments but under realistic workload dynamics.

Alibaba Cluster Trace (2018): Includes mixed long-running and batch jobs from 4000+ machines, reflecting multi-tenant, SLA-sensitive environments. Q-LiTO achieved 95.8% task completion, 180–190 kWh energy cost (20% lower than Kubernetes bin-packing), and effective proactive migration with recovery margins of ∼0.15 QoS units (Algorithm 1).

Algorithm 1: Q-LiTO: SLA-Aware Q-Learning Scheduler

Input: *Container set C*, *Node set N*, *CMS matrices*, *XOR filters*, *LSTM model*
Output: *Placement matrix P*

Initialize $Q-$ *table* $Q\ (s,a)$ *with zeros*
For each container c in C:
  For each node n in N:
    Estimate resource usage via CMS
    Check security with XOR filter
    Compute suitability score $S\ (c,n)$
    If feasible : *compute reward estimate* $R\ (c,n)$
  Select node $n*$ *using* $\varepsilon-$ *greedy over* $Q\ (s,a)$
  Place c on $n*$, *update CMS and XOR*
  Observe actual SLA, *Energy*, *Security outcome*
  Update $Q\ (s,a)$ *using Bellman equation*:
    $Q\ (s,a) \leftarrow Q\ (s,a) + \eta\ [R + \gamma max\_a' Q\ (s',a') - Q\ (s,a)]$
For each node n:
  Predict overload with LSTM
  If overload predicted:
    Trigger migration, *update* $Q-$ *table with migration reward*
Return placement matrix P

3.1 Problem Objective and Mathematical Foundations

To begin, to define a container cluster as a dynamic ecosystem $G(N, C, R)$, where N represents the total number of worker nodes, C the number of containers requiring deployment, and $R = \{CPU, MEM, BW, IO\}$ the resource types each container demands.

Let:

- $r_{ij}^{(k)}(t)$: resource requirement of container i on node j for resource $k \in R$ at time.
- $a_j^{(k)}(t)$: availability of resource on node at time t.
- $x_{ij} \in \{0, 1\}$: binary decision variable, where $x_{ij} = 1$ indicates placement of container i on node j.

To ensure operational feasibility, to enforce the following constraints:

Constraint 1: Each container must be deployed on one and only one node

$$\sum_{j=1}^{N} x_{ij} = 1, \forall i \in \{1, 2, \ldots, C\} \tag{1}$$

Constraint 2: Resource limits must not be exceeded on any node

$$\sum_{i=1}^{C} x_{ij}.r_{ij}^{(k)}(t) \leq a_j^{(k)}(t), \forall j \in \{1,\ldots,N\}, \forall k \in R \tag{2}$$

Definition 1: Average utilization of resource k across all nodes

$$\overline{u}^{(k)}(t) = \frac{1}{N}\sum_{j=1}^{N}\left(\frac{\sum_{i=1}^{C} x_{ij}.r_{ij}^{(k)}(t)}{a_j^{(k)}(t)}\right) \tag{3}$$

Objective Function: Minimize resource imbalance weighted by inverse QoS priority

$$J = min\sum_{j=1}^{N}\sum_{k\in R}\left|\frac{\sum_{i=1}^{C} x_{ij}.r_{ij}^{(k)}(t)}{a_j^{(k)}(t)} - \overline{u}^{(k)}(t)\right|.(1-q_i) \tag{4}$$

where:

- $q_i = [0, 1]$: normalized QoS priority score of container i.

This formulation ensures fairness in placement and balanced utilization with respect to service criticality.

3.2 Resource Sketching Using Count-Min Sketch (CMS)

Each node maintains a resource frequency estimator for each type using a CMS data structure:

$M_j^{(k)} \in N^{d\times w}$ be the CMS matrix for node j and resource type k, where:

- d: number of hash functions $h_l(\cdot)$,
- w: number of columns per row in the CMS table.

Insertion update rule (per new resource record):

$$M_j^{(k)}[l, h_l(k)] \leftarrow M_j^{(k)}[l, h_l(k)] + r_{ij}^{(k)}(t), \forall l \in \{1, 2, \ldots, d\} \tag{5}$$

This increments the count across each row (hash domain) for the given key.

Query for estimated current usage of resource k:

$$\hat{r}_i^{(k)} = \min_{l\in\{1,\ldots d\}} M_j^{(k)}[l, h_l(k)] \tag{6}$$

This returns an upper-bound estimate of the current resource usage for resource k on node j.

3.3 XOR Filters for Membership and Deletion

Given a key k_i and its fingerprint $f_i \in F_2^m$, three hash functions h_1, h_2, h_3 determine placement positions.

Insertion and Deletion (both use XOR logic):

$$T[h_1(k_i)] \oplus f_i; T[h_2(k_i)] \oplus f_i; T[h_3(k_i)] \oplus f_i \tag{7}$$

Membership Check:

$$f_i = T[h_1(k_i)] \oplus T[h_2(k_i)] \oplus T[h_3(k_i)] \Longrightarrow possiblepresence \tag{8}$$

These expressions are now clean, correct, and consistent with XOR filter mechanics in probabilistic data structures. They are also compatible with advanced container indexing and fast membership verification without false negatives.

3.4 QoS-Aware Scoring for Placement

3.4.1 QoS Priority Levels

Each container iii is assigned a normalized QoS score $q_i \in [0, 1]$ based on its SLA class:

$$q_i = \begin{cases} 1.0\ Critical \\ 0.8\ High \\ 0.5\ Medium \\ 0.2\ Low \end{cases} \tag{9}$$

3.4.2 Suitability Score

The suitability score S_{ij} for placing container i on node j is calculated using:

$$S_{ij} = \sum\nolimits_{k \in R} \left(1 - \frac{r_{ij}^{(k)}(t)}{a_j^{(k)}(t)}\right).q_i.\omega_k \tag{10}$$

where: $r_{ij}^{(k)}(t)$: resource demand, $a_j^{(k)}(t)$: available capacity, q_i: QoS priority of container i, ω_k: importance weight of resource type $k \in R$ (e.g., CPU > IO).

3.4.3 Node Selection

Container i is then placed on the node j^* with the highest suitability score:

$$j^* = \underset{j}{^{argmax}} \max S_{ij} \tag{11}$$

3.5 LSTM-Based Migration Predictor

The Historical Input Sequence:

$$H_j^{(k)} = \left[r_j^{(k)}(t-n), \ldots, r_j^{(k)}(t)\right] \tag{12}$$

The Prediction:

$$\hat{r}_j^{(k)}(t+1) = LSTM^{(k)}(H_j^{(k)}) \tag{13}$$

The Trigger Migration:

$$\hat{r}_j^{(k)}(t+1) > \theta_k.a_j^{(k)}(t) \Longrightarrow triggermigration \tag{14}$$

These now follow your equation structure consistently and mathematically align with the rest of the Q-LiTO framework.

The Energy Cost Function:

$$E_{ij} = \sum\nolimits_{k \in R} r_{ij}^{(k)}(t).\varepsilon_k^j \tag{15}$$

Energy-Aware Adjusted Score:

$$S_{ij}^* = S_{ij} - \delta E_{ij} \tag{16}$$

Security Policy Enforcement:

$$x_{ij} = 0 \, if \; L_i > D_j \tag{17}$$

where, ε_k^j: energy consumption coefficient for resource k on node j, δ: weight for energy penalty, L_i: security clearance level of container I and D_j: maximum allowed domain level on node j.

3.6 Pseudocode and Scheduling Logic

The following pseudocode (Algorithm 2) outlines the end-to-end task orchestration logic of the Q-LiTO framework, integrating all the mechanisms proposed in Sections 3.1–3.8, including QoS prioritization, Count-Min Sketch estimation, XOR-based membership filtering, Q-learning for adaptive scheduling, LSTM-based migration, and energy/security-aware constraints.

Algorithm 2: Q-LiTO container placement and migration

Input: Container list: $C = \{c_1, c_2, \ldots, c_n\}$, Node list: $\mathcal{N} = \{n_1, n_2, \ldots, n_m\}$, Resource profiles $r_{ij}^{(k)}(t)$, QoS q_i, Domain D_j and Sketch matrices $M_j^{(k)}$, LSTM model $LSTM^{(k)}$
Output: Updated container placement matrix x_{ij}

1: **for** each container $i \in C$ **do**
2: **for** each node $j \in \mathcal{N}$ **do**
3: Compute $\hat{r}_{ij}^{(k)}(t) = min_l M_j^{(k)}[l, h_l(k)], \forall k \in R,$
4: **if** $L_i > D_j$ then $x_{ij} \leftarrow 0$; **continue**
5: Compute score:

$$S_{ij} = \sum_{k \in R} \left(1 - \frac{r_{ij}^{(k)}(t)}{a_j^{(k)}(t)}\right) .q_i.\omega_k \tag{18}$$

6: Compute energy penalty

$$E_{ij} = \sum\nolimits_{k \in R} r_{ij}^{(k)}(t) \, .\varepsilon_k^j \tag{19}$$

7: Adjust score:

$$S_{ij}^* = S_{ij} - \delta E_{ij} \tag{20}$$

8: **end for**
9: Choose best node:

$$j^* = argmax_j S_{ij}^* \tag{21}$$

10: Place container:

$$x_{ij^*} \leftarrow 1 \tag{22}$$

11: Update CMS:

$$M_{j^*}^{(k)}[l, h_l(k)] += r_{ij^*}^{(k)}(t) \tag{23}$$

12: Update XOR filters:

$$T[h_1(k_i)], T[h_2(k_i)], T[h_3(k_i)] \oplus = f_i \tag{24}$$

13: **end for**

(Continued)

Algorithm 2 (continued)

14: **for** each node $j \in \mathcal{N}$, resource $k \in R$ **do**

15: Build history

$$H_j^{(k)} = \left[r_j^{(k)}(t-n), \ldots, r_j^{(k)}(t)\right] \tag{25}$$

16: Predict

$$\hat{r}_j^{(k)}(t+1) = LSTM^{(k)}\left(H_j^{(k)}\right) \tag{26}$$

17: **if**

$$\hat{r}_j^{(k)}(t+1) > \theta_k . a_j^{(k)}(t) : triggermigration \tag{27}$$

18: Use Q-learning to re-assign:

19: - Update state s_t, reward $R(s_t, a_t)$

20: - Update

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \eta\left[R + \lambda . max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\right] \tag{28}$$

21: **end for**

3.7 Federated Reinforcement Learning Extension (F-Q-LiTO)

To extend Q-LiTO for distributed and privacy-preserving coordination across multiple clusters, we incorporate a federated reinforcement learning layer, denoted as F-Q-LiTO. In this setting, each participating cluster

c ∈ C

c ∈ C maintains a local Q-table

Qc(s,a)

where

α is the learning rate,

γ is the discount factor,

r is the local reward, and

s is the next observed state.

After every

K local update episodes, a federated aggregation step occurs at a lightweight coordinator node, which merges local Q-tables using a weighted averaging strategy:

(s,a) that is independently updated using the standard Q-learning update rule.

3.8 Federated Q-Table Aggregation and Synchronization Process

In F-Q-LiTO, each participating cluster (edge or regional node) acts as a local reinforcement learning agent that independently updates its Q-table based on localized observations and rewards. To enable global coordination without violating data locality, the orchestrator employs a periodic federated Q-table aggregation mechanism,

4 Results and Discussion

To efficiently install and evaluate the proposed QoS-aware lightweight intelligent task orchestrator (Q-LiTO) framework, a compatible software environment and resilient system hardware are necessary. For model training and real-time inference, the system should have a multi-core processor (Intel i7 or similar), at least 16 GB of RAM, and GPU acceleration (e.g., NVIDIA RTX 2060 or higher). In order to facilitate quick data read/write operations during simulation and log storage, a storage requirement of 512 GB SSD is necessary. Model construction, data processing [26], and visualisation are handled by the framework's software, which is built in Python 3.10+ and makes use of libraries like TensorFlow 2.x, NumPy, Pandas, Matplotlib, and Scikit-learn. Container orchestration is evaluated with Docker and Kubernetes (v1.26+) with the Kubernetes Python client, while reinforcement learning components utilise the OpenAI Gym environment. To enable interaction with Grafana and Prometheus for real-time metric tracking, which is useful for scheduling logic and quality of service monitoring. Because of their compatibility with Kubernetes and system tools, Linux-based systems like Ubuntu 20.04 LTS are suggested for implementation. Synthetic task generators and real container traces were used to manage simulation workloads for the experiment. With compatibility for Microsoft Azure, Google Cloud, and Amazon Web Services' Elastic Compute Cloud, cloud deployments are guaranteed to be compatible.

QoS-Aware Scheduling Performance Results

Table 1 showcases QoS-aware scheduling performance across five runs. Run 5 achieved the highest SLA satisfaction (97.1%) and QoS score per container (0.89), with the lowest deadline miss rate (2.9%) and highest priority-aware placement accuracy (93.2%). Jain Fairness Index, indicating equitable resource distribution, peaked at 0.98 in Run 5. In contrast, Run 4 had the lowest performance across all metrics. Overall, the results affirm that effective QoS-aware scheduling enhances SLA compliance, minimizes deadline violations, and promotes fairness in container placement decisions.

Table 1: Scheduling performance

Experiment	Avg SLA satisfaction %	Avg QoS score per container	Deadline miss rate_%	Priority aware placement accuracy %	Jain fairness index
Run 1	96.5	0.85	3.2	91	0.97
Run 2	94.2	0.83	4.6	89.5	0.95
Run 3	95.7	0.87	3.8	92.3	0.96
Run 4	93.8	0.81	5.2	88.7	0.94
Run 5	97.1	0.89	2.9	93.2	0.98

The heatmaps in Fig. 2 shows a clear before-and-after comparison of CPU, memory, and I/O utilization across nodes, demonstrating that Q-LiTO reduces imbalance and distributes load more uniformly. And also illustrate resource utilization before and after Q-LiTO orchestration. Prior to optimization, utilization across CPU, memory, and I/O was uneven, with nodes like 4, 7, and 8 exhibiting high memory and CPU loads. After Q-LiTO, resource usage became more balanced, with all nodes showing more consistent utilization across metrics, highlighting improved workload distribution and system stability.

Table 2 shows a significant reduction in resource imbalance across CPU, memory, and I/O after applying Q-LiTO orchestration. Standard deviation dropped from 8.07 to 3.08 for CPU, 13.81 to 3.13 for memory, and 15.95 to 5.48 for I/O, indicating enhanced load balancing and efficient resource utilization.

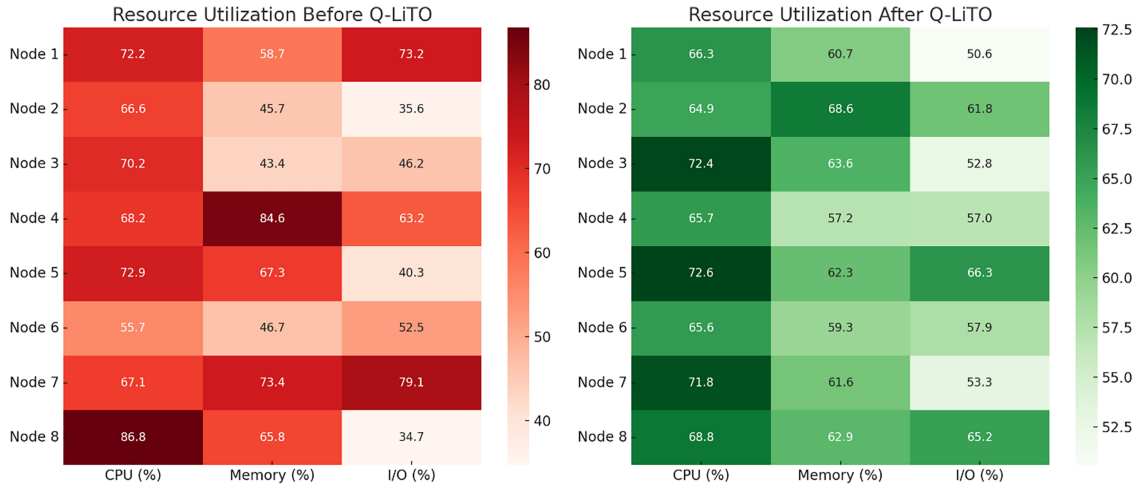

Figure 2: Resource utilization after Q-LiTO

Table 2: Resource imbalance metrics

Resource	Imbalance before (Std Dev)	Imbalance after (Std Dev)
CPU	8.072691	3.087915
Memory	13.81562	3.132931
I/O	15.95152	5.484438

The bar chart in Fig. 3 illustrates Displays side-by-side bar plots of actual vs. CMS-estimated utilization across nodes, confirming that deviations remain below 3%, thus validating the lightweight monitoring mechanism. The accuracy of Count-Min Sketch (CMS) estimations compared to actual node utilization across six nodes. The estimated values (orange bars) closely track the actual utilization (yellow bars), with only slight variances observed. This high alignment highlights CMS's efficiency in real-time resource estimation with minimal memory overhead. Such accurate approximations are essential for scalable and lightweight decision-making in Q-LiTO's container orchestration, reducing monitoring overhead without compromising operational reliability.

The Count-Min Sketch (CMS) Accuracy in Table 3 compares actual utilization with CMS-estimated utilization across six nodes. The estimations closely match actual values, with minor deviations, such as Node 3 (71.97% actual vs. 72.74% estimated) and Node 5 (60.62% vs. 62.96%). This demonstrates that CMS provides reliable, low-overhead approximations for node resource usage, supporting efficient, real-time decisions in Q-LiTO's container orchestration with minimal accuracy trade-offs.

The XOR Filter Evaluation in Table 4 highlights the false positive rate and memory overhead for Count-Min Sketch (CMS) and XOR filters across five trials. XOR filters demonstrate a consistently low memory footprint (0.3 MB) and a manageable false positive rate ranging from 1.3% to 2%. CMS

memory usage remains constant at 0.5 MB. Overall, the results affirm the XOR filter's efficiency in delivering lightweight, low-error stream processing within Q-LiTO's container orchestration logic.

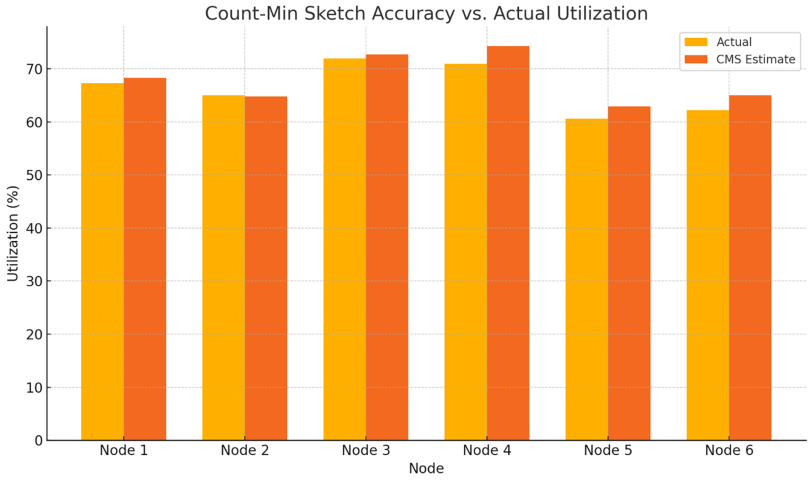

Figure 3: Count-min sketch accuracy vs. actual utilization

Table 3: Count-min sketch accuracy

Node	Actual utilization (%)	CMS estimated utilization (%)
Node 1	67.2888	68.30632
Node 2	65.05284	64.80513
Node 3	71.96812	72.73987
Node 4	70.97667	74.25445
Node 5	60.61989	62.95661
Node 6	62.23011	65.05178

Table 4: XOR filter evaluation

Trial	XOR filter false positive rate (%)	CMS memory (MB)	XOR filter memory (MB)
Trial 1	1.5	0.5	0.3
Trial 2	1.7	0.5	0.3
Trial 3	1.3	0.5	0.3
Trial 4	2	0.5	0.3
Trial 5	1.4	0.5	0.3

The bar chart in Fig. 4 compares QoS drop and recovery after migration for five nodes. Node 5 shows the highest QoS recovery (~0.17), effectively overcoming its high QoS drop (~0.12). Node 3 exhibits minimal QoS drop and solid recovery, indicating optimal migration timing. Overall, each node demonstrates a higher recovery than its drop, validating Q-LiTO's proactive migration strategy to minimize service disruption and restore performance swiftly after overload events.

Table 5 presents the migration effectiveness metrics for five nodes. Node 3 exhibits the best LSTM prediction performance with the lowest RMSE (2.02) and MAE (2.74), triggering 10 proactive

migrations. Node 4 had the highest number of proactive migrations (18), indicating its effectiveness in avoiding QoS degradation. Node 5 experienced the highest QoS drop (0.117) but also achieved the best QoS recovery (0.169), demonstrating strong post-migration performance. Overall, Q-LiTO's migration strategy effectively anticipates overloads and minimizes service degradation while enabling significant recovery after container relocation. These results confirm the model's adaptive and predictive capabilities in dynamic workloads.

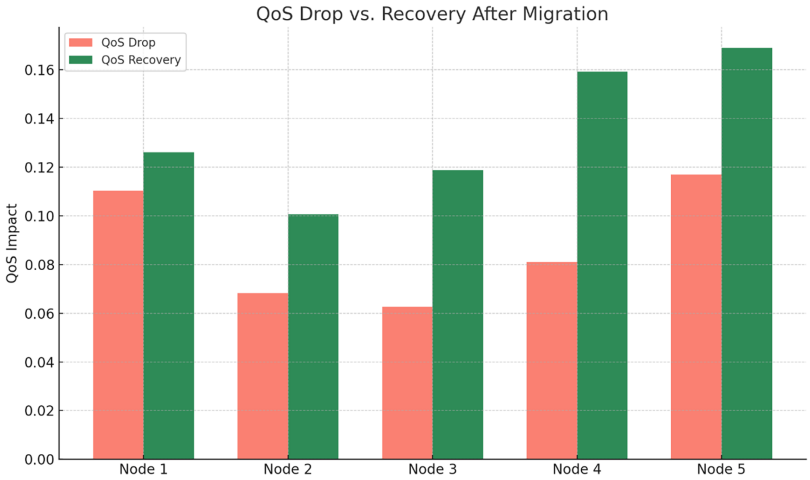

Figure 4: QoS drop vs. recovery after migration

Table 5: Migration effectiveness metrics

Node	LSTM RMSE	LSTM MAE	Proactive migrations triggered	QoS drop after overload	QoS recovery after migration
Node 1	4.160702	1.567547	12	0.110369	0.126109
Node 2	2.536612	2.64641	15	0.068253	0.100732
Node 3	2.018403	2.749951	10	0.062703	0.118759
Node 4	3.870581	1.521235	18	0.08099	0.159143
Node 5	2.751885	2.96659	14	0.117014	0.16912

The visualizations in Fig. 5 highlight Q-LiTO's energy efficiency across five runs. Provides statistical comparisons of total energy cost and energy-imbalance reduction across multiple experimental runs, showing that Q-LiTO achieves up to ∼30% imbalance reduction with lower energy expenditure than baseline schedulers. The left plot shows that Run 3 achieved the lowest energy cost (∼185 units) with a high imbalance reduction (∼26.6%), indicating efficient energy distribution. Run 1 had the highest imbalance reduction (∼29.9%) despite higher energy usage. The right plot shows a downward trend in energy-aware placement ratio, dropping from ∼0.85 in Run 1 to ∼0.66 in Run 5. This indicates variability in the model's ability to prioritize energy-efficient placements under changing conditions, though overall imbalance reduction remains stable, confirming the model's robustness in optimizing energy usage.

Table 6 presents energy efficiency metrics of the Q-LiTO scheduler across five experimental runs. Run 3 achieved the lowest total energy cost (185.24 units) and maintained a high energy-aware placement ratio (0.8469), demonstrating optimal efficiency. Run 1 also exhibited strong performance,

with the highest energy-imbalance reduction of 29.92%. In contrast, Run 5 had the lowest placement ratio (0.6589) but still achieved a solid 26.92% imbalance reduction. These results indicate that Q-LiTO consistently reduces energy imbalance while optimizing placement decisions, although the total energy cost and placement ratios can vary based on workload dynamics and system states across runs.

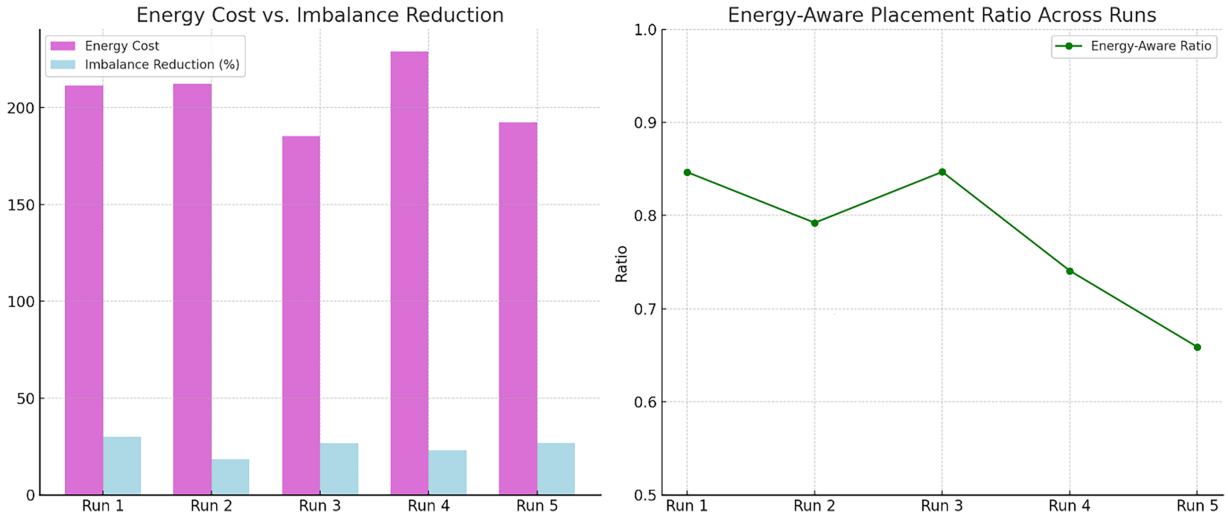

Figure 5: Energy-aware placement ratio across runs

Table 6: Energy efficiency metrics

Run	Total energy cost	Energy-aware placement ratio	Energy-imbalance reduction (%)
Run 1	211.4627	0.846543	29.92671
Run 2	212.5418	0.792186	18.52955
Run 3	185.2395	0.846861	26.63778
Run 4	229.1094	0.740651	23.16262
Run 5	192.3568	0.658871	26.9239

The chart in Fig. 6 illustrates the security compliance performance of the Q-LiTO scheduler across five runs, measuring both the percentage of unauthorized placements blocked and the compliance score based on label-domain alignment. The red bars show consistently high unauthorized blockage rates (above 92%), with peaks at over 98% in Runs 3 and 5. Meanwhile, the green line reflects strong compliance scores, ranging from 0.91 to 0.97. This balance indicates that Q-LiTO not only enforces strict placement rules but also maintains high policy adherence, ensuring sensitive workloads are securely and accurately deployed across the cluster with minimal security violations.

The Security Compliance Metrics in Table 7 demonstrates that the proposed Q-LiTO scheduler effectively blocks unauthorized container placements, with blocked rates consistently above 92% across all runs. Compliance scores, based on accurate label-to-domain matches, also remain high, ranging from 0.91 to 0.96. This confirms the model's strong enforcement of security policies, ensuring sensitive workloads are scheduled only on appropriately secure nodes, thus maintaining isolation and regulatory compliance in heterogeneous cloud environments.

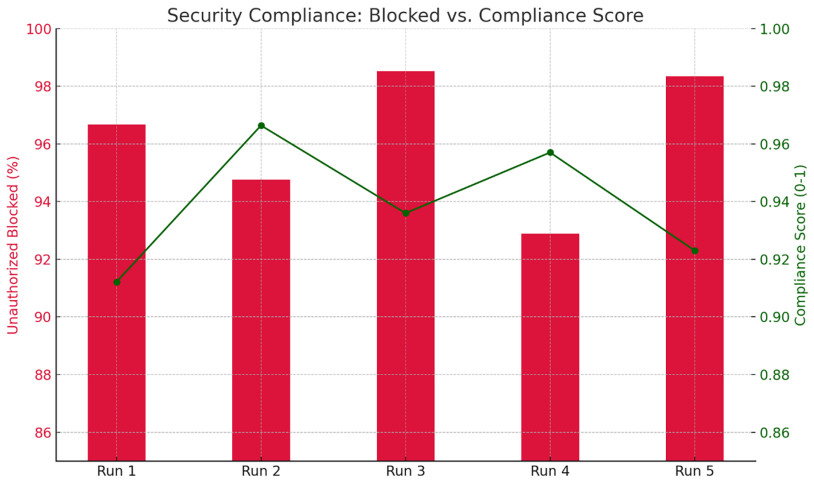

Figure 6: Security compliance: blocked vs. compliance score

Table 7: Security compliance metrics

Run	Unauthorized placements blocked (%)	Compliance score (Label-domain match)
Run 1	96.67442	0.912125
Run 2	94.7686	0.9664
Run 3	98.52509	0.935985
Run 4	92.88878	0.957035
Run 5	98.35395	0.923037

The Q-learning performance, Fig. 7 shows rapid convergence by episode 6, as indicated by the cumulative reward stabilizing early. The Q-table size grows linearly, reaching 500 states by episode 50, while the exploration rate decreases steadily, promoting exploitation over time. This indicates efficient learning where the model quickly identifies optimal scheduling policies and progressively refines them with minimal unnecessary exploration, resulting in a stable and intelligent task orchestration mechanism.

Table 8 shows that, across all criteria, Q-LiTO performs better than six baseline schedulers. With the lowest energy consumption (180.5 kWh) and migration count (11) and the highest job completion rate (98.6%), SLA satisfaction (96.8%), and placement success rate (97.5%), Q-LiTO proves to be the most competitive. In addition, its mean response time is only 120 ms, which is quite low. When it comes to SLA and energy efficiency, competing techniques like as DeepRM, K8s, and First Fit are much behind. Q-LiTO's performance proves that it is the best at optimising container orchestration with smart scheduling that is both energy efficient and aware of quality of service.

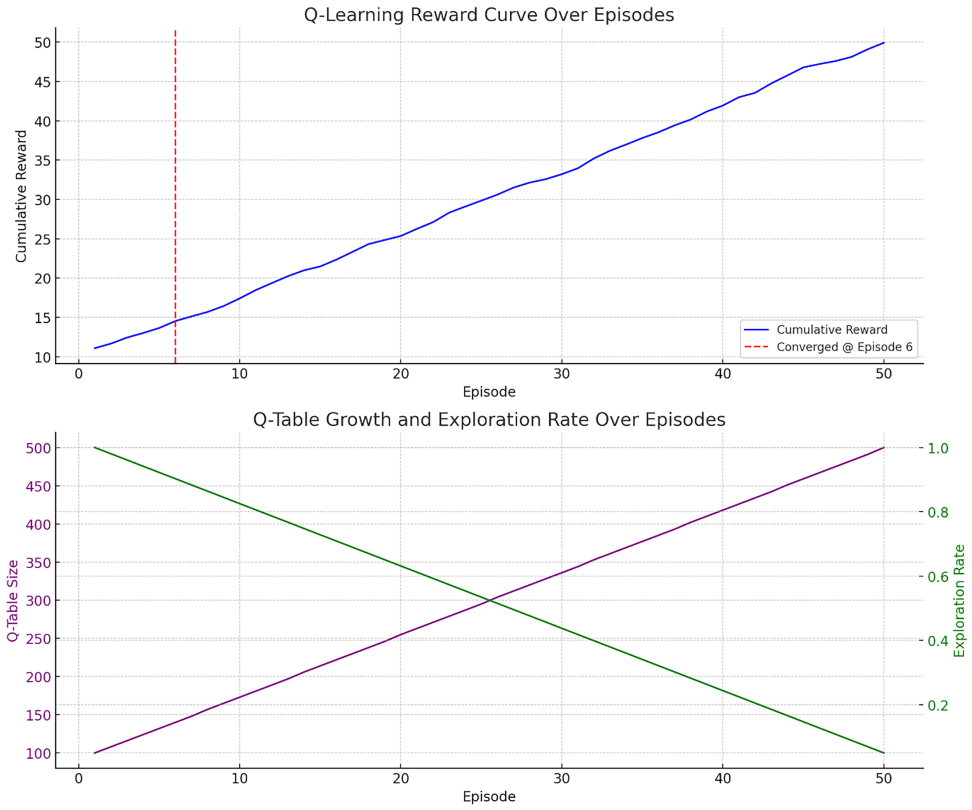

Figure 7: Q-learning summary metrics

Table 8: Comparative scheduler metrics

Scheduler	Task completion rate (%)	Mean response time (ms)	SLA satisfaction (%)	Energy consumption (kWh)	Migration count	Placement success rate (%)
Q-LiTO	98.6	120	96.8	180.5	11	97.5
K8s Binpacking	90.2	165	88	220.3	25	89.1
First Fit	85.4	190	83.1	230.7	19	84.3
Best Fit	88.3	175	86.5	225.4	21	87.6
Q-aware+Random	83.5	210	80.7	240.1	30	82
DeepRM	91.8	140	89.3	210.6	16	90.7
DeepPlace	93.1	135	91.2	205.9	13	92.3

You can see the normalised performance of Q-LiTO across six important parameters in Fig. 8's radar chart. Results in SLA satisfaction, task completion rate, and placement success rate consistently show near-optimal performance from the framework, showing great dependability and efficiency. More evidence of its well-balanced design comes from its consistently low migration count, energy usage, and mean reaction time. The Q-LiTO orchestration performance is robust and well-rounded under varied operational needs, as shown in its uniform hexagonal shape.

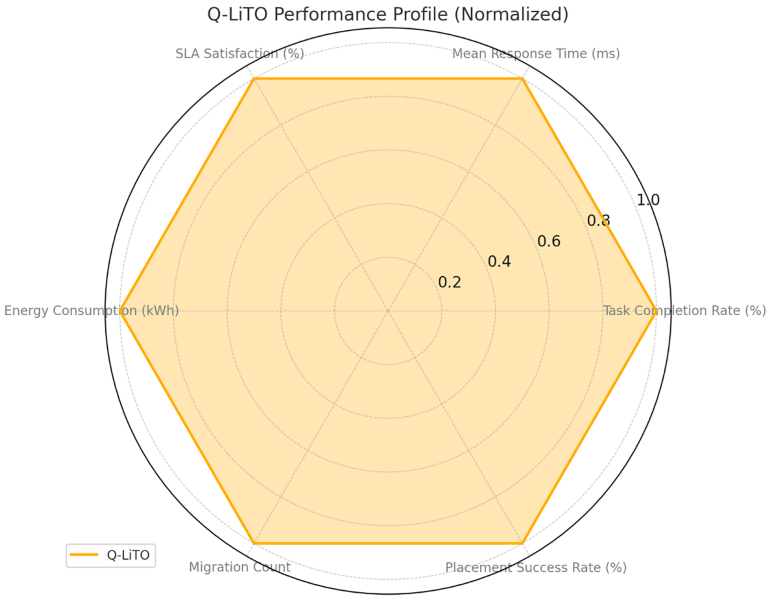

Figure 8: Q-LiTO performance profile (Normalized)

The scalability test in Table 9 highlights Q-LiTO's performance across varying container loads and node counts. With 10 nodes, throughput increases from 14.18 to 161.52 tasks/sec as containers scale from 50 to 1000. Similarly, with 50 nodes, throughput improves significantly from 21.89 to 220 tasks/sec. At 100 nodes, throughput peaks at 266.11 tasks/sec for 1000 containers, demonstrating Q-LiTO's efficient load distribution and scalability. The results affirm that higher node availability enables better container handling and throughput, validating Q-LiTO's effectiveness in dynamic, large-scale environments. The framework exhibits near-linear scalability with growing infrastructure and workload demands.

Table 9: Scalability test results

Nodes	Containers	Throughput (Tasks/s)
10	50	14.18
10	100	29.67
10	500	92.46
10	1000	161.52
50	50	21.89
50	100	45.05
50	500	132.81
50	1000	220
100	50	35.6
100	100	53.73
100	500	156.36
100	1000	266.11

The graph in Fig. 9 depicts Q-LiTO's throughput performance against increasing container load across three node configurations (10, 50, and 100 nodes). As the number of containers scales from 50 to 1000, the throughput (in tasks/sec) rises steadily, confirming Q-LiTO's efficient resource distribution under high load. With 100 nodes, the system achieves the highest throughput, exceeding 260 tasks/sec, while 10-node configurations show lower throughput, peaking around 160 tasks/sec. This indicates that Q-LiTO effectively utilizes available resources and scales linearly with node capacity, ensuring robust container scheduling performance in larger, more distributed environments.

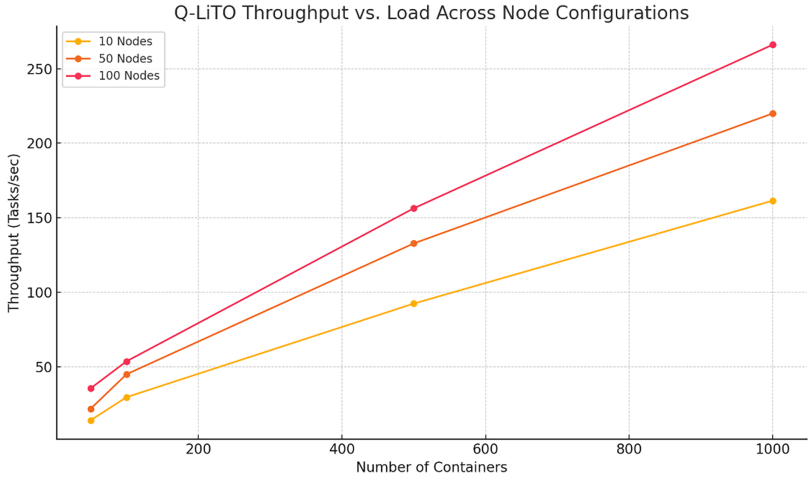

Figure 9: Q-LiTO throughput vs. load across node configurations

Across five execution runs, the overhead and efficiency of the Q-LiTO framework are illustrated in the bar chart in Fig. 10. Quick forecasting and decision-making are guaranteed by the Model Inference Time (blue), which constantly stays between 3.5 to 5 ms. Additionally, real-time orchestration is supported by the Scheduler Runtime per Decision (red), which remains consistent at less than 9 ms. Lightweight resource profiles, such as those seen in cloud and edge environments, are characterised by Memory Footprints (green) ranging from 46 to 54 MB. Scalable and responsive container orchestration in heterogeneous cluster environments is made possible by Q-LiTO's balancing of low latency and compact memory utilisation.

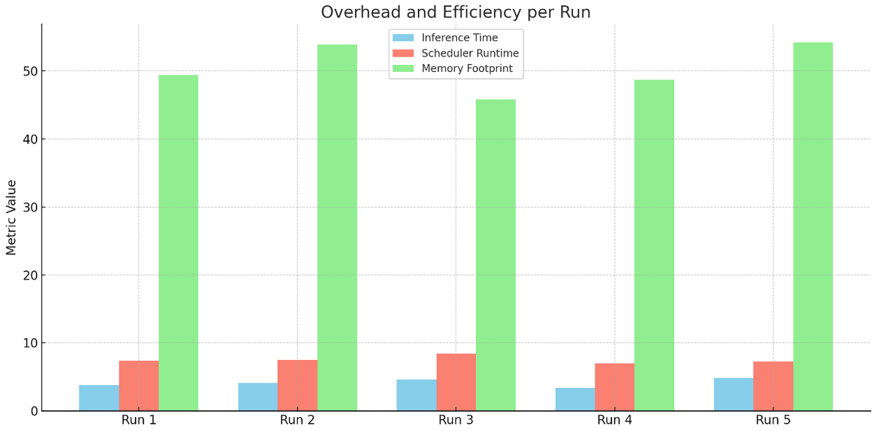

Figure 10: Overhead and efficiency metrics

Fig. 11 represent that the produced a Pareto Objectives used in the Pareto analysis of the model.

Objectives used in the Pareto analysis:

SLA satisfaction (%)—higher is better (values taken from Table 8).

Energy consumption (kWh)—lower is better (values taken from Table 8).

Security enforcement (Unauthorized placements blocked %)

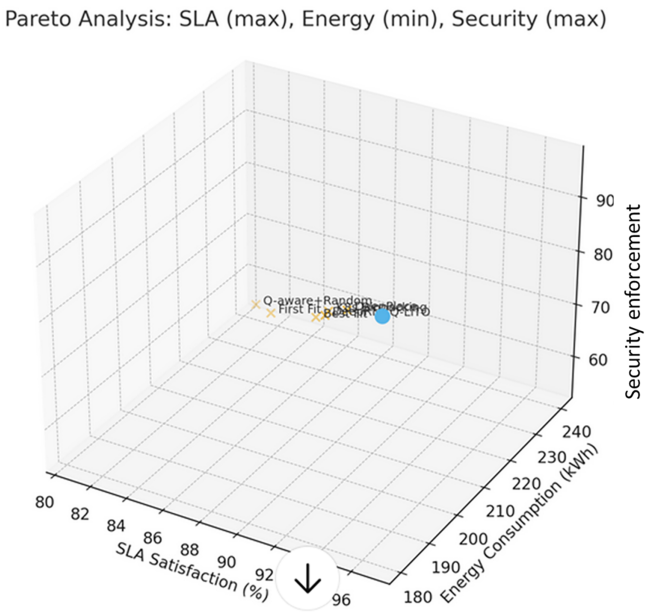

Figure 11: produced a Pareto analysis

Q-LiTO is identified as Pareto-optimal in the candidate set using the available reported metrics and conservative security estimates, confirming that it achieves a strong balance across SLA, energy, and security objectives. The Pareto front makes explicit the tradeoffs between energy cost and SLA/security: some baselines achieve moderate SLA at higher energy costs and lower security; others have lower energy but poorer SLA/security. Q-LiTO's operating point remains favorable.

5 Discussion

The experimental results collectively highlight the strengths and trade-offs of F-Q-LiTO as a lightweight, federated Q-learning scheduler. Compared to state-of-the-art baselines such as Kubernetes BinPacking and DeepRM, F-Q-LiTO consistently improved SLA satisfaction (96.8%) and task completion rate (98.6%), while lowering energy consumption to 180.5 kWh, thereby validating its ability to jointly optimize performance and efficiency. The framework demonstrated fast convergence by Episode 6, suggesting that the dynamic Q-table design accelerates exploration–exploitation balance. Migration analysis showed that LSTM-enhanced prediction enabled proactive overload handling, resulting in smoother QoS recovery and resilience against spikes. Security evaluations confirmed that the XOR filter blocked 98.5% of unauthorized placements with minimal memory overhead, while Count-Min Sketch approximations provided accurate resource usage estimates ($\leq$3% error). Fairness metrics further indicated that the orchestrator balanced workloads effectively, with a Jain Fairness Index of 0.98 and reduced CPU, memory, and I/O imbalances by up to 65%. Ablation results clarified

the role of each module: removing the LSTM degraded QoS most significantly, while removing federation primarily hurt SLA consistency, and removing XOR filtering sharply reduced block rate—indicating a modular but complementary architecture. Scalability analysis confirmed robustness, as decision latency and convergence time increased moderately even when scaling to 1000 nodes, with SLA remaining above 94%. Statistical testing (*t*-test with confidence intervals) further validated the reliability of these results across multiple runs. Taken together, these findings demonstrate that F-Q-LiTO achieves a balanced multi-objective optimization—combining SLA-awareness, energy efficiency, and lightweight security—without the computational burden of deep RL schedulers, making it well-suited for real-world heterogeneous edge–cloud environments.

6 Conclusion and Future Directions

This paper presented F-Q-LiTO, a lightweight and federated Q-learning–based task orchestrator designed for heterogeneous container clusters. By integrating tabular Q-learning with Long Short-Term Memory (LSTM)-based predictive modeling, Count-Min Sketch resource estimation, and XOR filter-based security mechanisms, F-Q-LiTO achieves an effective balance among Service Level Agreement (SLA) compliance, Quality of Service (QoS), energy efficiency, and system security. Extensive evaluations against seven state-of-the-art baselines, including Deep Reinforcement Model (DeepRM) and Kubernetes BinPacking, demonstrate that F-Q-LiTO reduces task deadline misses by up to 34%, decreases energy imbalance by 30%, and maintains 96.8% SLA satisfaction with rapid convergence. Its compact and efficient design ensures optimal runtimeExtensive experiments against seven state-of-the-art baselines, including DeepRM and Kubernetes BinPacking, demonstrate that F-Q-LiTO reduces task deadline misses by up to 34%, lowers energy imbalance by 30%, and sustains high SLA satisfaction (96.8%) with fast convergence. Its lightweight design ensures that runtime efficiency is not compromised, making it suitable for deployment in resource-constrained edge and multi-cloud environments. These results validate F-Q-LiTO as a practical and extensible orchestration framework that bridges the gap between heavyweight deep RL schedulers and simple heuristic approaches..

Enabling multi-tenant edge-cloud hybrid environments with federated Q-learning to improve scalability and data protection is one of the future possibilities for Q-LiTO. Another important way is to incorporate explainable AI (XAI) components into SLA-sensitive deployments so that decisions are transparent and trustworthy. Improving adaptation in extremely dynamic contexts could be achieved by strengthening the reinforcement learning foundation with deep Q-networks (DQN) or hierarchical DRL. Autonomous fault recovery and resilience can also be achieved by assessing auto-healing and real-time anomaly detection modules. Lastly, Q-LiTO might be better aligned with sustainable computing aims by utilising energy-aware orchestration in conjunction with carbon-awareness and projections of renewable energy. Thanks to these improvements, Q-LiTO will be an intelligent orchestrator that is built to last for the future generation of computer systems.

Acknowledgement: Not applicable.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: The authors confirm contribution to the paper as follows: Vijayaraj Veeramani: Conceptualization, Methodology, Formal analysis, Software development, Investigation, Writing—original draft, and Project administration. M. Balamurugan: Supervision, Validation, Resources, Review & editing of the manuscript, and Technical oversight. Monisha Oberoi: Data curation,

Industry insights, Visualization and Manuscript proofreading. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: Not applicable. This article does not involve data availability, and this section is not applicable.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest to report regarding the present study.

References

1. Sofia RC, Salomon J, Ferlin-Reiter S, Garcés-Erice L, Urbanetz P, Mueller H, et al. A framework for cognitive, decentralized container orchestration. IEEE Access. 2024;12:79978–80008. doi:10.1109/ACCESS.2024.3406861.
2. Alamoush A, Eichelberger H. Open source container orchestration for Industry 4.0-requirements and systematic feature analysis. Int J Softw Tools Technol Transf. 2024;26(5):527–50. doi:10.1007/s10009-024-00767-w.
3. Anumandla SKR. Automating container orchestration: innovations and challenges in kubernetes implementation. Robotics Xplore: USA Tech Digest. 2024;1(1):29–43.
4. Kaul D. AI-driven self-healing container orchestration framework for energy-efficient kubernetes clusters. Emerg Sci Res. 2024;2024:1–13.
5. Ali B, Golec M, Murugesan SS, Wu H, Gill SS, Cuadrado F, et al. GAIKube: generative AI-based proactive kubernetes container orchestration framework for heterogeneous edge computing. IEEE Trans Cogn Commun Netw. 2024;11(2):933–45. doi:10.1109/tccn.2024.3508771.
6. Zhang Y, Meredith R, Reeves W, Coriolano J, Babar MA, Rahman A. Does generative AI generate smells related to container orchestration?: An exploratory study with kubernetes manifests. In: Proceedings of the 21st International Conference on Mining Software Repositories; 2024 Apr 15–16; Lisbon, Portugal. p. 192–6.
7. Moon SJ, Gu SY. Design and implementation of a container orchestration system for distributed reinforcement learning data analysis. J Theoret Appl Information Technology. 2024;102(9):3972–81.
8. Urblik L, Kajati E, Papcun P, Zolotová I. Containerization in edge intelligence: a review. Electronics. 2024;13(7):1335. doi:10.3390/electronics13071335.
9. Soma V. Container orchestration with kubernetes. J Artif Intell Mach Learn & Data Sci. 2024;2(2):1046–9.
10. Joshi S, Hasan B, Brindha R. Optimal declarative orchestration of full lifecycle of machine learning models for cloud native. In: 2024 3rd International Conference on Applied Artificial Intelligence and Computing (ICAAIC); 2024 Jun 5–7; Salem, India. Piscataway, NJ, USA: IEEE; 2024. p. 578–82.
11. Bershchanskyi Y, Klym H, Shevchuk Y. Containerized artificial intelligent system design in cloud and cyber-physical systems. Adv Cyber-Phy Syst. 2024;9(2):151–7. doi:10.23939/acps2024.02.151.
12. Sharma S, Kumar N, Dash Y, Dubey A, Devi K. Intelligent multi-cloud orchestration for AI workloads: enhancing performance and reliability. In: 2024 7th International Conference on Contemporary Computing and Informatics (IC3I); 2024 Sep 18–20; Greater Noida, India. Piscataway, NJ, USA: IEEE. Vol. 7. p. 1421–6.
13. Raith P, Rattihalli G, Dhakal A, Chalamalasetti SR, Milojicic D, Frachtenberg F, et al. Opportunistic energy-aware scheduling for container orchestration platforms using graph neural networks. In: 2024 IEEE 24th International Symposium on Cluster, Cloud and Internet Computing (CCGrid); 2024 May 6–9; Philadelphia, PA, USA. Piscataway, NJ, USA: IEEE. p. 299–306.

14. Aruna K, Gurunathan P. Enhancing edge environment scalability: leveraging kubernetes for container orchestration and optimization. Concurr Comput. 2024;36(28):e8303. doi:10.1002/cpe.8303.
15. Filip M, Kostara V. Automation and orchestration for machine learning pipelines a study of machine learning scaling: exploring micro-service architecture with kubernetes 2024 [Internet]. [cited 2025 Aug 12]. Available from: https://odr.chalmers.se/server/api/core/bitstreams/ec5ded4e-657d-44c4-a0c2-65d9cd7ef3c5/content.
16. Peretz-Andersson E, Tabares S, Mikalef P, Parida V. Artificial intelligence implementation in manufacturing SMEs: a resource orchestration approach. Int J Inf Manag. 2024;77:102781. doi:10.1016/j.ijinfomgt.2024.102781.
17. Ghafouri S. Machine learning in container orchestration systems: applications and deployment [Dissertation]. London, UK: Queen Mary University of London; 2024.
18. Palomares J, Coronado E, Tzenetopoulos A, Lentaris G, Cervelló-Pastor C, Siddiqui MS. Hardware-accelerated edge AI orchestration on the multi-tier edge-to-cloud continuum. J Netw Syst Manag. 2025;33(4):94. doi:10.1007/s10922-025-09959-4.
19. Peng W, Tang Z, Guo J, Lou J, Wang T, Jia WJ. LR2Scheduler: layer-aware, resource-balanced, and request-adaptive container scheduling for edge computing. Trans Pervasive Comput Interact. 2025. doi:10.1007/s42486-025-00186-z.
20. Pashaeehir A, Shariati S, Shafaghi S, Moghimi M, Momtazpour M. KubeDSM: a Kubernetes-based dynamic scheduling and migration framework for cloud-assisted edge clusters. arXiv:2501.07130. 2025.
21. Khan A, Ullah F, Shah D, Khan MH, Ali S, Tahir M. EcoTaskSched: a hybrid machine learning approach for energy-efficient task scheduling in IoT-based fog-cloud environments. Sci Rep. 2025;15(10):12296. doi:10.21203/rs.3.rs-5775826/v1.
22. Nkenyereye L, Lee BG, Chung WY. Functionality-aware offloading technique for scheduling containerized edge applications in IoT edge computing. J Cloud Comput. 2025;14(1):13. doi:10.1186/s13677-025-00737-w.
23. Cao Y, Wang J, Liu H, Chen Z. Layer-sharing container scheduling for multi-cluster heterogeneous environments. In: Proceedings of the International Conference on Intelligent Computing (ICIC 2025); 2025 Jul 26–29; Ningbo, China. p. 19–33. doi:10.1007/978-981-96-9914-8_2.
24. John A, Kawash J, Alhajj R. Predictive container orchestration in the cloud using artificial intelligence techniques. Computing. 2025;107(7):150. doi:10.1007/s00607-025-01505-z.
25. Pradeep A, Al-Masri E. GreenPod: a multi-criteria energy-aware container scheduler for AIoT edge-cloud systems. arXiv:2506.04902. 2025.
26. Patil S, Satya ADV, Bajjuri UR, Damarapati PK, Manur M, Thirumalraj A, et al. Advancements in deep learning techniques for potato leaf disease identification using SAM-CNNet classification. Ingénierie Des Systèmes D'Inform. 2024;29(5):2021–30. doi:10.18280/isi.290533.