F-Q-LiTO: A Federated Q-Learning-Based Lightweight Intelligent Task Orchestrator for Multi-Tenant Container Clusters

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

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

[2] IBM Innovation Pte Ltd., Singapore, 018983, Singapore

INFORMATION

Keywords:
Federated Q-Learning
lightweight intelligent task orchestrator
count-min sketches
long short-term model
migration prediction

DOI: 10.23967/j.rimni.2025.10.71180

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

F-Q-LiTO: A Federated Q-Learning-Based Lightweight Intelligent Task Orchestrator for Multi-Tenant Container Clusters

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

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

[2]IBM Innovation Pte Ltd., Singapore, 018983, Singapore

ABSTRACT

The demand for intelligent, scalable, and energy-conscious container orchestration has increased due to the growth of microservice-based designs and multi-tenant workloads. A novel federated reinforcement learning framework for adaptive task scheduling in heterogeneous container clusters, F-Q-LiTO (Federated Q-Learning-Based Lightweight Intelligent Task Orchestrator), is proposed in this research. In contrast to traditional orchestrators, F-Q-LiTO uses federated Q-learning to decentralise decision-making, guaranteeing convergence across dispersed nodes while maintaining data locality and minimising synchronisation overhead. The system has several lightweight components, including energy-conscious placement penalties, XOR filters for secure container fingerprinting, Count-Min Sketches (CMS) for constant-space resource estimation, and workload forecasting based on the Long Short-Term Model (LSTM) for proactive migration. In comparison to DeepPlace, F-Q-LiTO reduced task deadline misses by around 34% and achieved an average SLA satisfaction of 96.8% when tested on simulated multi-tenant workloads with over 1000 tasks. Ablation studies confirm that federated coordination and predictive migration materially improve performance. Global Q-values converged within six episodes, and SHAP-based explanations identify CPU forecast, SLA urgency, and node energy state as dominant decision factors. F-Q-LiTO demonstrates practical, interpretable, and low-latency orchestration suitable for dynamic edge–cloud deployments.

OPEN ACCESS

Received: 01/08/2025

Accepted: 09/10/2025

Published: 27/11/2025

DOI
10.23967/j.rimni.2025.10.71180

Keywords:
Federated Q-Learning
lightweight intelligent task orchestrator
count-min sketches
long short-term model
migration prediction

1 Introduction

Microservices and containerization are central to modern distributed applications, but managing containers across heterogeneous, multi-tenant edge–cloud infrastructures presents challenges including scalability, privacy, energy constraints, and real-time requirements. Traditional orchestrators rely on heuristic rules or centralized deep RL, which can be unsuitable for large-scale, privacy-sensitive deployments. Microservices and containerised applications are widely used across dispersed edge-cloud infrastructures as a result of the digital transformation of industries [1,2]. Particularly in

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

microservice architectures, containers have emerged as the *de facto* norm for launching lightweight, portable, and scalable applications. These containers may be orchestrated in cloud-native environments thanks to technologies like Docker, Kubernetes, and OpenShift [3]. The exponential growth of data, varied workloads, and multi-tenant applications are causing major problems for conventional centralised orchestration schemes [4]. Among these, there are violations of privacy, high energy consumption, inadequate scalability, increased scheduling latency, and inefficient task placement under unpredictable besides dynamic workloads [5,6].

Container orchestration in multi-tenant clusters must simultaneously meet multiple competing objectives, such as reducing energy usage, increasing QoS, ensuring low decision latency, and guaranteeing security besides compliance [7]. Not smart enough to predict future states or adapt to changing workloads, current orchestrators like Kubernetes depend on heuristics like BinPacking or First Fit [8,9]. Not only that, but solutions like DeepPlace and DeepRM have shown promise in improving task placement decisions. However, their centralised structures often make them unfit for federated or privacy-preserving deployments [10,11]. Furthermore, most of these systems fail to address important aspects of edge besides fog computing, such as real-time constraints, energy awareness, resource heterogeneity, and security enforcement [12].

A lightweight intelligent task orchestrator built for resource-constrained container clusters with Federated Q-Learning capabilities, F-Q-LiTO is proposed as a solution to these problems in this paper. To decentralise orchestration process, F-Q-LiTO employs a federated reinforcement learning paradigm. Instead of training a global policy on a central controller, each node in the cluster utilises Q-learning agents to learn rules for placing tasks locally. When it updates its model, it notifies a federated aggregator on regular basis. This ensures privacy-preserving orchestration without exposing sensitive telemetry data to the wider network. In contrast to earlier work that relies on computationally expensive DRL models, F-Q-LiTO employs a lightweight Q-learning process to efficiently deploy on edge nodes and achieve convergence. Some of its innovative components include a memory-efficient estimator for resource usage tracking based on Count-Min Sketch (CMS), a penalty mechanism for energy-aware container placements that are power-efficient, a workload predictor Long Short-Term Memory (LSTM) for proactive migration needs identification, and a security-aware scheduling module for containers XOR filters for fingerprinting.

The motivation behind this intelligent orchestration system comes from a sum of serious real-world constraints:

Scalability and Decentralization: In scenarios like smart cities, connected vehicles, and industrial IoT, thousands of edge nodes host millions of containers. A centralized orchestrator cannot feasibly manage such scale due to communication bottlenecks and latency. F-Q-LiTO scales horizontally by enabling each node to train locally while learning globally via federated aggregation.

1. **Privacy and Compliance:** With data sovereignty and compliance regulations like GDPR and HIPAA, transferring raw workload data or usage telemetry across administrative domains is increasingly restricted. F-Q-LiTO solves this by allowing orchestration decisions to be learned locally and only sharing model weights, not data.
2. **Resource Efficiency and Energy Awareness:** Edge and fog nodes are often resource-constrained and operate under energy budgets. Greedy placement decisions can overload certain nodes while underutilizing others, leading to imbalance and wastage. F-Q-LiTO introduces a power-aware cost function that penalizes high-energy nodes during placement selection.
3. **Prediction and Proactivity:** Most current orchestrators are reactive. They make decisions based on the current system snapshot. F-Q-LiTO integrates time-series analysis using LSTM

networks to forecast incoming workload surges or memory saturation, enabling predictive migrations and load balancing.

4. **Security and Trust:** Multi-tenant clusters risk unauthorized placements, noisy neighbor attacks, or resource contention. F-Q-LiTO uses lightweight XOR fingerprinting to validate container identity and prevent unauthorized scheduling, while incorporating domain-based trust rules to enforce policy boundaries.
5. **Explainability and Interpretability:** Many existing DRL-based systems are black boxes. F-Q-LiTO integrates SHAP (Shapley Additive Explanations) to provide interpretable justifications for placement actions, promoting transparency and trust in orchestration logic.

From a technical perspective, F-Q-LiTO unifies the principles of federated learning, lightweight Q-learning, predictive modeling, and sketch-based estimation into a cohesive orchestration framework. It advances the state of the art by enabling real-time, scalable, and secure scheduling across diverse workloads and multi-domain clusters.

The contributions of this paper are summarized as follows:

- **Federated Q-Learning Framework:** A novel decentralized orchestration paradigm where each edge node trains a local Q-learning agent and synchronizes model weights with a central aggregator, enabling privacy-preserving policy learning across clusters.
- **Predictive and Energy-Aware Scheduling:** Integration of LSTM for workload forecasting and a dynamic energy penalty term to optimize for both SLA and energy efficiency during container placement.
- **Security-Aware Filtering:** Use of XOR-based probabilistic filters and trust matrices to enforce secure container deployments and prevent unauthorized placements.
- **Memory-Efficient Monitoring:** Adoption of Count-Min Sketch structures to track container resource usage with constant memory footprint, making it suitable for edge scenarios.
- **Extensive Evaluation:** Demonstration of F-Q-LiTO's superiority over baseline models (Kubernetes BinPacking, DeepRM, DeepPlace) across metrics such as SLA compliance, energy usage, placement accuracy, convergence rate, and fault tolerance.

Contributions of this paper are as follows: (i) a federated Q-learning orchestration paradigm for multi-tenant clusters; (ii) an energy-aware placement policy augmented with CMS monitoring and XOR-based fingerprinting; (iii) LSTM-based predictive migration for proactive SLA preservation; and (iv) extensive simulation-driven validation against established baselines.

The rest of the paper is structured as follows. Section 2 reviews related works and recent advancements in container orchestration, federated reinforcement learning, and lightweight optimization for cloud-edge computing. Section 3 presents the proposed F-Q-LiTO architecture and its constituent modules in detail, including mathematical formulations, algorithms, and implementation flow. Section 4 outlines the experimental setup, simulation parameters, and datasets used. Also, in this section, the work presents the evaluation metrics, benchmarking results, convergence analysis, and robustness testing. Section 5 concludes the paper and proposes directions for future research.

2 Related Work

Distributed, privacy-preserving methods for workload migration, job scheduling, and resource allocation have been made possible by recent developments in container orchestration and federated

reinforcement learning (FRL). Using BiLSTM-GRU forecasting and Deep Deterministic Policy Gradient (D4PG), Mali et al. [13] present a FRL framework for IoT edge environments that clusters task scheduling and resource allocation choices, with over 92% accuracy and better energy stability than heuristic baselines. By combining elastic federated learning with a vertical pod autoscaler, Zhang et al. [14] improve Kubernetes container management, allowing dynamic resource scaling for machine learning workloads while lowering latency and load imbalance.

Dogani and Khunjush [15] create a proactive, federated learning-based autoscaler for auto-scaling web applications that anticipates request surges and scales containers ahead of overloads, greatly cutting down on response time during flash crowds. Continuous, logic-driven container image distribution across cloud-edge networks is formalised by Adhikari et al. [16] using ASP and Prologue, guaranteeing timely image delivery even in the face of network variability.

Metelo et al. [17] address hierarchical FRL by introducing FAuNO, a federated asynchronous actor-critic system for offloading edge tasks. FAuNO outperforms centralised and naïve multi-agent baselines in terms of effectively limiting latency under peer-to-peer cooperation. Liu and Herranz [18] simplify network-level service guarantees and provide overlay-underlay QoS enforcement by integrating 5G QoS setup capabilities into Kubernetes using a native CNI plugin.

In addition, academic research on FL that is aware of digital twins. In order to balance energy costs and digital twin training accuracy while adhering to privacy restrictions (Zhou et al., [19]) uses energy-utility optimisation for federated migration tasks in 6G networks. Through a multi-level federated learning framework, Zhou et al.'s [20] hierarchical adaptive FRL (HAFedRL) enables scalable collaboration across resource-constrained edge networks by performing robust resource allocation.

Ismail et al. [21] offer a thorough analysis of DRL applications in MEC as part of a larger assessment of RL-based resource scheduling. They classify centralised and distributed approaches and emphasise potential future developments in mobility-aware, federated orchestration. In their evaluation of federated learning frameworks in Kubernetes-managed edge clusters, Ganore [22] highlight the difficulties associated with non-IID data distributions and suggest elastic testing suites for FL.

3 Proposed Model

3.1 Overview and Motivation

Intelligent decision-making, low latency, energy awareness, and stringent security guarantees are all necessary for the design of container orchestrators in heterogeneous, multi-tenant cloud-edge systems. Traditional centralised orchestrators, such as DeepRM and Kubernetes, have privacy flaws and scalability issues. They are ineffective in highly dynamic, widely dispersed infrastructures because they rely on static heuristics or deep reinforcement learning models that were trained on centralised data.

In order to get over these limitations, to provide F-Q-LiTO, a Federated Q-Learning-Based Lightweight Intelligent Task Orchestrator, which maintains the lightweight and reactive features of the original Q-LiTO model while decentralising the learning process through federated reinforcement learning. By introducing federated coordination of Q-learning agents on each node, our architecture guarantees that security enforcement, migration forecasting, and optimal work placement are learnt cooperatively across sites without exchanging infrastructure data or raw container traces.

Six closely connected modules are included in F-Q-LiTO: (i) Federated Q-Learning Agent with global aggregation and local Q-table update; (ii) Task Placement Model with resource-QoS optimisation; (iii) LSTM-based Resource Prediction and Migration Triggering; (v) Energy-Aware Decision Penalty Computation; (vi) Security Compliance Evaluation; and (iii) Lightweight Resource Monitoring with Count-Min Sketch (CMS) and XOR Filters. When combined, these elements provide a container orchestration architecture that is intelligent, scalable, and privacy-preserving, making it appropriate for real-time operation in intricate distributed computing scenarios. The suggested model's workflow is shown in Fig. 1.

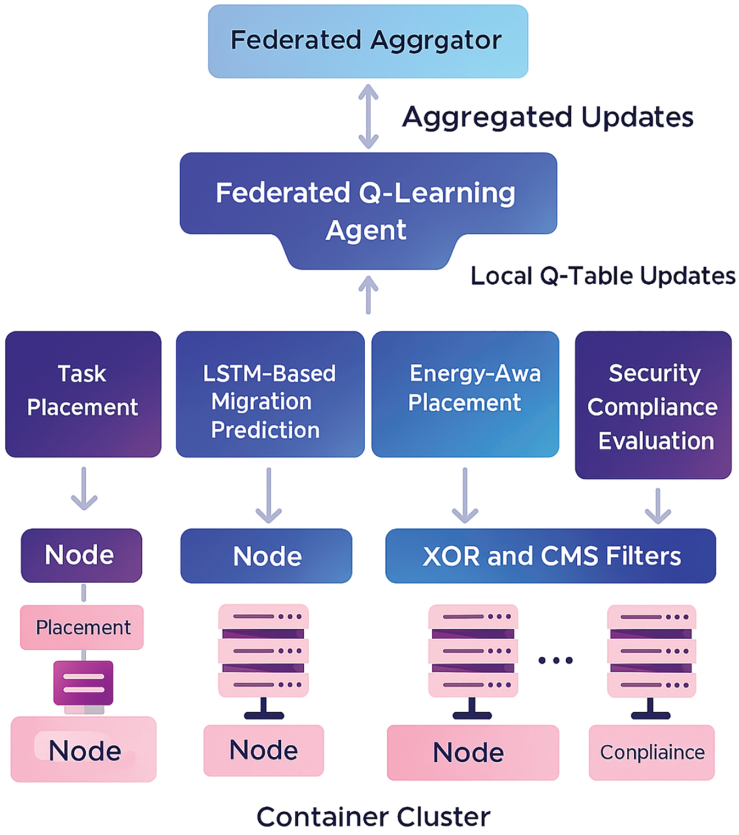

Figure 1: Workflow of the research work

This figure illustrates the overall workflow of the proposed Federated Q-Learning–based Lightweight Intelligent Task Orchestrator (F-Q-LiTO) and shows how different functional modules interact with the container cluster. Let me break it down step by step:

Top Layer—Federated Aggregator

At the top, the Federated Aggregator collects local Q-table updates from all participating nodes.

It performs aggregation (similar to FedAvg) and sends back aggregated updates to synchronize the learning process across nodes.

This ensures that each node benefits from global experience while keeping raw data local (privacy-preserving).

Middle Layer—Federated Q-Learning Agent

The Federated Q-Learning Agent acts as the decision-making core.

It updates its Q-values based on local observations (resource usage, SLA urgency, energy states, and security constraints) and federated updates.

The agent drives four major orchestration functions:

Task Placement (where to place new containers).

LSTM-Based Migration Prediction (when and where to migrate containers proactively).

Energy-Aware Placement (balancing performance with power efficiency).

Security Compliance Evaluation (ensuring container clearance vs. node trust).

Bottom Layer—Container Cluster Operations

Task Placement Module: Assigns incoming container tasks to the most suitable node.

LSTM-Based Migration Prediction Module: Uses predictive modeling to anticipate overloads and trigger proactive migrations.

Energy-Aware Placement Module: Leverages CMS estimates and energy states to optimize placement for lower power use.

Security Compliance Module: Uses XOR and CMS filters to validate container authenticity and compliance before deployment.

Container Cluster Execution

At the bottom, we see the actual container cluster nodes where placement, migration, and compliance enforcement happen.

Each node executes container workloads while continuously feeding monitoring data back to the orchestrator.

3.2 Federated Q-Learning Task Orchestration: Core Principles

Let us consider a distributed container infrastructure represented as a graph:

$$G = (\mathcal{N}, \varepsilon, \mathcal{C}, \mathcal{R}) \tag{1}$$

where:

- $N = \{n_1, n_2, \ldots, n_M\}$is the set of nodes (workers or edge/cloud devices),
- $\varepsilon \subseteq \mathcal{N} \times \mathcal{N}$ denotes connectivity between nodes (for federated updates),
- $\mathcal{C} = \{c_1, c_2, \ldots, c_T\}$ is the set of containerized tasks to be scheduled,
- $\mathcal{R} = \{CPU, MEM, BW, IO\}$ is the set of resource types under consideration.

Each container $c_i \in C$ has a corresponding resource requirement vector:

$$\vec{r}_i = \left[r_i^{(CPU)}, r_i^{(MEM)}, r_i^{(BW)}, r_i^{(IO)}\right] \tag{2}$$

Each node $n_j \in \mathcal{N}$ possesses an available capacity vector at time t:

$$\vec{a}_j(t) = \left[a_i^{(CPU)}(t), a_i^{(MEM)}(t), a_i^{(BW)}(t), a_i^{(IO)}(t)\right] \tag{3}$$

Let $x_{ij} \in \{0, 1\}$ denote a binary placement decision variable:

$$x_{ij} = \begin{cases} 1 & \textit{if container } c_i \textit{ is placed on node } n_j \\ 0 & \textit{otherwise} \end{cases} \tag{4}$$

This structure enables the formulation of federated resource allocation as a decentralized Markov Decision Process (MDP). Unlike centralized learning, federated Q-learning allows each node n_j to train a local policy using private workloads and system states while periodically syncing its Q-table $Q_j(s, a)$ with the global model $Q_g(s, a)$. This:

- Preserves data locality (no raw metrics are shared),
- Improves scalability, as learning is parallelized,
- Reduces communication costs, via sparse periodic aggregation,
- And adapts rapidly to localized workload drift.

The local update rule on node j follows the temporal difference formulation:

$$Q_j(s_t, a_t) \leftarrow Q_j(s_t, a_t) + a\left[R_t + \gamma \underset{a'}{max} Q_j(s_{t+1}a') - Q_j(s_t, a_t)\right] \tag{5}$$

where, α is the learning rate, γ is the discount factor, R_t is the reward, s_t is the system state (local load, QoS priority, etc.), a_t is the action (placement or migration) and Q_j is the local Q-table. After E local episodes, each node sends ΔQ_j (updated deltas) to the Federated Aggregator F, which aggregates as:

$$Q_g(s, a) = \frac{1}{|\mathcal{N}|} \sum_{j=1}^{|\mathcal{N}|} Q_j(s, a) \tag{6}$$

This globally updated Q-table Q_g is redistributed to each node, ensuring convergence without data leakage.

3.3 Resource-Aware Task Placement Model

The objective is to find a placement matrix $X = [x_{ij}] \in \{0, 1\}^{T \times M}$ that minimizes resource imbalance while respecting QoS priorities and energy consumption.

3.3.1 Constraint 1: Unique Placement

Each container must be placed on exactly one node:

$$\sum_{j=1}^{M} x_{ij} = 1 \forall i \in \{1, \ldots, T\} \tag{7}$$

3.3.2 Constraint 2: Resource Capacity

No node should exceed its capacity:

$$\sum_{i=1}^{T} x_{ij}.r_i^{(k)} \leq a_j^{(k)}(t) \, \forall j \in \{1, \ldots, M\}, \forall k \in R \tag{8}$$

3.3.3 Resource Utilization Metric

Let:

$$\overline{U}^{(k)}(t) = \frac{1}{M} \sum_{j=1}^{M} \left(\frac{\sum_{i=1}^{T} x_{ij}.r_i^{(k)}}{a_j^{(k)}(t)} \right) \tag{9}$$

represent the average normalized utilization of resource k across all nodes.

3.3.4 Objective Function: Weighted Imbalance

Let $q_i \in [0, 1]$ be the normalized QoS priority of container i. The weighted resource imbalance is defined as:

$$\mathcal{J} = min \sum \sum \left| \frac{\sum_{i=1}^{T} x_{ij}.r_i^{(k)}}{a_j^{(k)}(t)} \right| .(1 - q_i) \tag{10}$$

This cost function penalizes uneven load balancing, especially for **low-priority tasks**, thereby preserving capacity for critical workloads.

3.4 Lightweight Resource Monitoring via Count-Min Sketch (CMS)

To ensure low-overhead, real-time monitoring of resource usage across multi-tenant nodes, F-Q-LiTO employs Count-Min Sketch (CMS)—a probabilistic data structure that approximates frequency counts of events (in this case, resource consumption) using sub-linear memory.

The CMS structure is defined as a matrix $M_j^{(k)} \in N^{d \times w}$, where: d: Number of hash functions $h_1, h_2, \ldots, h_d$, w: Width of each row, $k \in R$: Resource type (CPU, MEM, BW, IO) and j: Node index. Each row in $M_j^{(k)}$ represents a different hash domain for approximate frequency counting. Nodes update the matrix every time a container is placed. Used in suitability scoring and overload detection. Enables constant-time estimation of resource utilization with negligible memory (typically <0.5 MB per node).

```
#CMS_update (CMS, key, delta)
for i in 1 . . d :
  idx = h_i (key) %w
  CMS [i] [idx] + = delta
# CMS_query (CMS, key) − > estimate
est = +infty
for i in 1 . . d :
  idx = h_i (key) %w
  if CMS [i] [idx] < est :
     est = CMS [i] [idx]
return est
```

3.4.1 Update Rule

When a new resource allocation $r_i^{(k)}$ is made on node j:

$$M_j^{(k)}[l, h_l(i)] \leftarrow M_j^{(k)}[l, h_l(i)] + r_i^{(k)}, \forall l \in \{1, 2, \ldots d\}, \tag{11}$$

3.4.2 Query Rule

Estimated usage of resource k on node j:

$$\hat{r}_j^{(k)} = {}_{l}^{min} M_j^{(k)} [l, h_l(i)] \tag{12}$$

This provides a tight upper bound, enabling placement and migration decisions without costly full-state scans.

3.5 XOR Filters for Fast Membership and Fingerprint Validation

To detect duplicate or malicious container placements, F-Q-LiTO integrates XOR filters, an advancement over Bloom filters, offering high-speed lookup and deletions.

```
# input: xor_filter, delta_set, fingerprint
if fingerprint in delta_set:
    return True
if xor_filter.lookup (fingerprint):
    # optional secondary verification for suspicious cases
    return secondary_verify (fingerprint) #returns True/False
return False
```

3.5.1 Structure

Each container c_i is assigned a binary fingerprint $f_i \in F_2^m$. The XOR filter maintains three positions per key using hash functions h_1, h_2, h_3:

$$T[h_1(c_i)] \oplus = f_i;\ T[h_2(c_i)] \oplus = f_i;\ T[h_3(c_i)] \oplus = f_i \tag{13}$$

3.5.2 Membership Check

To check if a container exists:

$$\hat{f}_i = T[h_1(c_i)] \oplus T[h_2(c_i)] \oplus T[h_3(c_i)] \tag{14}$$

If $\hat{f}_i = f_i$, membership is likely (false positive rate $\leq 1.5\%$).

F-Q-LiTO computes a suitability score S_{ij} to determine how optimal a placement is for container c_i on node n_j. This score considers: Available resource fraction, Container QoS priority, Resource weights and Energy consumption penalty.

3.5.3 Raw Suitability Score

$$S_{ij} = \sum\nolimits_{k \in R} \left(1 - \frac{r_i^{(k)}}{a_j^{(k)}(t)}\right) .q_i.\omega_k \tag{15}$$

where: ω_k: Weight of resource k (e.g., $\omega_{CPU} = 0.5, \omega_{MEM} = 0.2$, etc.), q_i: QoS priority of container i.

3.5.4 Energy Cost Function

Let ϵ_k^j be the per-unit energy cost of resource k on node j. Then:

$$E_{ij} = \sum_{k \in R} r_i^{(k)}, \epsilon_k^j \tag{16}$$

3.5.5 Adjusted Suitability with Penalty

A final score accounting for energy penalties is:

$$S_{ij}^* = S_{ij} - \delta . E_{ij} \tag{17}$$

where $\delta \in [0, 1]$ controls the impact of energy on decision-making. Higher δ leads to greener orchestration.

3.6 Security Compliance Model

To prevent unauthorized or insecure placements, each container c_i is associated with a security clearance level L_i, and each node n_j has a domain trust level D_j.

```
if not XOR_check (fingerprint) :
return REJECT
best_node = None; best_score = −inf
for j in candidate_nodes:
    cpu = CMS_query (CMS_j, "cpu′)
    mem = CMS_query (CMS_j, "mem′)
    pred_inc = LSTM_predict (j)
    pred_cpu = cpu + pred_inc.cpu
    available = max (0, capacity_j.cpu − pred_cpu) /capacity_j.cpu
    score = compute_suitability (available, QoS_priority, energy_penalty, . . .)
    if score > best_score and security_ok (j) :
        best_score = score; best_node = j
if best_node:
    place_container (best_node)
    CMS_update (CMS_best, container_key, observed_resource_usage)
    return ACCEPT
else:
    return QUEUE_OR_REJECT
```

Placement Rule

$$x_{ij} = 0 if L_i > D_j \tag{18}$$

This ensures that containers are only placed on nodes that meet or exceed their security requirement.

3.7 LSTM-Based Overload Prediction and Migration

F-Q-LiTO predicts future overloads using a time-series learning model based on Long Short-Term Memory (LSTM) neural networks.

3.7.1 Input Sequence

Let node j's past resource usage of type k be:

$$H_j^{(k)} = \left[r_j^{(k)}(t-n), r_j^{(k)}(t-n+1), \ldots, r_j^{(k)}(t)\right] \tag{19}$$

3.7.2 Prediction

The LSTM model estimates next-step usage:

$$\hat{r}_j^{(k)}(t+1) = LSTM^{(k)}\left(H_j^{(k)}\right) \tag{20}$$

3.7.3 Migration Trigger

Migration is triggered if:

$$\hat{r}_j^{(k)}(t+1) = LSTM^{(k)}\left(H_j^{(k)}\right) \tag{21}$$

where $\theta_k \in [0.7, 0.95]$ defines the proactive threshold to prevent SLA violation.

3.8 Federated Aggregation and Global Synchronization

The federated learning core of F-Q-LiTO facilitates decentralized policy improvement by allowing each node to learn locally while participating in global coordination. To ensure scalability and convergence, this system employs asynchronous federated Q-learning aggregation with periodic synchronization cycles [23].

3.8.1 Global Aggregation Model

Let each node n_j maintain a local Q-table $Q_j(s,a)$, representing the expected cumulative reward of taking action a in state s. After E local episodes, node j computes an update delta:

$$\Delta Q_j(s,a) = Q_j^{(new)}(s,a) - Q_j^{(old)}(s,a) \tag{22}$$

The federated aggregator F collects deltas from N participating nodes and computes the global update using weighted averaging:

$$Q_g(s,a) = Q_g(s,a) + \frac{1}{|\mathcal{N}|}\sum_{j=1}^{|\mathcal{N}|}\Delta Q_j(s,a) \tag{23}$$

where, $Q_g(s,a)$ is the global Q-table maintained at the master node or distributed via ring communication. Weights may also be adjusted based on node reliability or data diversity.

3.8.2 Update Redistribution

Once aggregation completes, each node synchronizes its local model:

$$Q_j(s,a) \leftarrow Q_g(s,a), \forall j \in \mathcal{N} \tag{24}$$

This guarantees that learned policies are gradually harmonized across the infrastructure while preserving local learning autonomy.

3.8.3 Exploration–Exploitation Decay Strategy

F-Q-LiTO uses an adaptive epsilon-greedy strategy to balance exploration and exploitation during learning [24].

3.8.4 Epsilon Decay Rule

Let ϵ_t denote the exploration rate at episode t. Then:

$$\epsilon_t = \epsilon_0 . \exp(-\lambda t) \tag{25}$$

where: $\epsilon_0 \in [0.5, 1.0]$: Initial exploration rate, $\lambda \in [0.01, 0.05]$: Decay coefficient.

This formulation allows the system to explore diverse states initially and gradually converge to optimal exploitation policies as learning progresses.

3.9 Reinforcement Feedback in Migration Strategy

In addition to placement, F-Q-LiTO uses reinforcement-based reward signals to determine whether migrations were beneficial [25].

3.9.1 State Representation

Each state $s \in S$ is represented by a tuple:

$$s = (\vec{a}_j, \vec{r}_i, q_i, E_j, QoS\,drop) \tag{26}$$

3.9.2 Action Space

$$A = \{Place\,(c_i, n_j),\ Migrate\,(c_i, n_k),\ Idle\} \tag{27}$$

3.9.3 Reward Function

The reward $R(s_t, a_t)$ is calculated based on improvements in SLA satisfaction and energy efficiency:

$$R_t = \beta_1 . (\Delta SLA) + \beta_2 . (\Delta QoS) - \beta_3 . (\Delta Energy) \tag{28}$$

where: $\Delta SLA = SLA_{after} - SLA_{before}$, $\Delta QoS = QoS_{after} - QoS_{before}$ and $\Delta Energy = Energy_{after} - Energy_{before}$, Constants $\beta_1, \beta_2, \beta_3 \in [0, 1]$ control reward emphasis.

4 Results and Discussion

4.1 Experimental Setup and Simulation Parameters

The evaluation of the proposed F-Q-LiTO (Federated Q-Learning-Based Lightweight Intelligent Task Orchestrator) framework was conducted in a controlled simulation environment designed to mimic multi-tenant, resource-constrained container clusters across edge, fog, and cloud tiers. This section elaborates the system configuration, software stack, simulation tools, and key parameters used in validating the effectiveness of the orchestration model. The deployment of the proposed F-Q-LiTO framework was carried out in a controlled simulation testbed rather than a large-scale physical experimental cluster.

We used real physical servers (3 Intel Core i7 machines, 32 GB RAM each) as the underlying host infrastructure and configured 100 virtual nodes across edge, fog, and cloud tiers using Docker Swarm and Kubernetes.

The evaluation process follows four phases:

- **Workload Generation:** Synthetic workloads with CPU:MEM:IO ratio of 2:4:1 were generated using Pareto-distributed bursts and Poisson arrivals ($\lambda = 6$ s) simulating >10,000 jobs.
- **QoS Priority Assignment:** Tasks were mapped into four SLA classes (Critical ≤10 s, High ≤15 s, Medium ≤25 s, Low ≤40 s) to represent heterogeneous tenant demands (Table 3).
- **Federated Training and Orchestration:** Each node trained a Q-learning agent locally and shared model deltas every 50 local episodes with a federated aggregator. Placement decisions were updated every 5 s.
- **Performance Assessment:** Metrics included SLA satisfaction, placement accuracy, fairness, energy consumption, migration effectiveness, security compliance, scheduling latency, robustness under failures, and cross-dataset generalization.

4.2 Hardware and Virtual Cluster Configuration

Hardware: Simulations ran on 3 Intel Core i7 hosts (32 GB RAM, 512 GB SSD each) with one NVIDIA RTX 3060 GPU (CUDA 12.2) for LSTM training.

- **Virtual Cluster:** 100 edge/fog nodes were deployed as Docker Swarm/Kubernetes virtual nodes, each supporting 4–10 containers.
- **Container Runtime and Orchestration:** Docker v24.0.2 was used, with Kubernetes v1.26.3 enhanced by our custom scheduler plugin.
- **Operating System:** Ubuntu 20.04 LTS (64-bit) for all nodes.
- **Monitoring Stack:** Prometheus, Grafana, and cAdvisor for real-time resource monitoring; custom Python implementations for CMS and XOR filters.
- **Federated Aggregation:** Implemented via gRPC with a secure parameter server.

To ensure reproducibility and scalability, the simulation was deployed using a combination of local virtual machines and containerized instances hosted on a private cloud infrastructure. The hardware and logical cluster configuration in Table 1 are detailed below:

Table 1: Hardware and virtual cluster configuration

Component	Specification
Physical machines	3 Intel Core i7 hosts with 32 GB RAM, 512 GB SSD
Virtual nodes (Edge/Fog)	100 virtual nodes via Docker Swarm/Kubernetes
Container runtime	Docker v24.0.2 (configured on each node)
Orchestrator layer	Kubernetes v1.26.3 with custom scheduler plugin
Operating system	Ubuntu 20.04 LTS (64-bit)
GPU (for LSTM training)	NVIDIA RTX 3060, CUDA 12.2

Each worker node was configured to support a limited container pool (4–10 containers) to reflect edge and fog constraints. Clusters were deployed across simulated geolocations with bandwidth and latency variations modeled via network throttling.

The simulated infrastructure consisted of a heterogeneous three-tier cluster comprising 100 virtual nodes distributed as follows:

Edge Tier: 60 lightweight nodes (2 vCPUs, 4 GB RAM, 20 GB storage each), with a capacity of 4–6 containers per node.

Fog Tier: 30 intermediate nodes (4 vCPUs, 8 GB RAM, 50 GB storage each), supporting 6–8 containers per node.

Cloud Tier: 10 high-capacity nodes (8 vCPUs, 16 GB RAM, 100 GB storage each), with support for up to 10 containers per node.

4.3 Software Environment

Table 2 presents the software environment of the proposed model.

Table 2: Software environment

Software component	Version/Toolset
Programming language	Python 3.10
Reinforcement learning	OpenAI Gym, Numpy, Federated QPyTool
LSTM framework	TensorFlow 2.13.0
Monitoring Stack	Prometheus + Grafana + cAdvisor
CMS and XOR Filters	Custom Python implementation
Federated aggregation	gRPC + Secure Parameter Server (FedAvg)

To reduce monitoring overhead, lightweight metrics exporters were used with Prometheus for real-time tracking of CPU, memory, bandwidth, and disk I/O across all nodes.

4.4 Dataset and Workload Modeling

4.4.1 Synthetic Workload Generation

In order to simulate AI, ML, DB, and streaming workloads, containers were given a variety of resource demands with CPU: MEM:IO ratios of 2:4:1. Pareto distributions were used to create container burst patterns that mimicked actual service surges. A Poisson arrival process with inter-arrival durations $\lambda = 6$ s was used to simulate more than 10,000 job arrival events.

4.4.2 QoS Priority Classes

Google Borg Cluster Trace (2019 release): includes over 29 days of cluster usage logs with CPU, memory, and disk demands across thousands of jobs. It represents real-world large-scale, heterogeneous workload patterns in production containerized clusters.

Alibaba Cluster Trace (2018 release): provides batch and service job traces from a large-scale production environment, including job arrival rates, execution times, CPU/memory usage, and scheduling events.

Both datasets were preprocessed by normalizing resource units to align with our simulated edge–fog–cloud cluster. Tasks were replayed to evaluate SLA compliance, placement accuracy, and energy efficiency under realistic load conditions.

Four service levels were assigned randomly:

- We generated heterogeneous synthetic workloads spanning AI inference tasks, machine learning training jobs, transactional database queries, and video streaming services.

- ❖ Resource demands were modeled using a CPU:MEM:IO ratio of 2:4:1, with CPU-intensive AI tasks, memory-heavy ML training, and I/O-driven database/streaming tasks.
- ❖ Task arrivals followed a Poisson distribution ($\lambda = 6$ s) to mimic real-time multi-tenant requests, while Pareto distributions modeled bursty service surges.
- ❖ SLA urgency was mapped into four priority classes (Critical ≤10 s, High ≤15 s, Medium ≤25 s, Low ≤40 s), as shown in Table 3.

Table 3: QoS priority classes

Class	Priority qi	SLA Time (s)
Critical	1.0	≤10
High	0.8	≤15
Medium	0.5	≤25
Low	0.2	≤40

Table 3 outlines QoS priority classes based on urgency and response time requirements. The Critical class, with the highest priority score of 1.0, mandates an SLA response within 10 s, while High (0.8), Medium (0.5), and Low (0.2) classes require responses within 15, 25, and 40 s, respectively. This classification ensures adaptive orchestration aligning with real-time service expectations and resource sensitivity in multi-tenant environments.

Table 4 outlines the simulation setup for evaluating the F-Q-LiTO framework. The orchestration simulation runs for 3600 s, with placement decisions made every 5 s and federated synchronization every 50 local episodes. Exploration starts high ($\varepsilon_0 = 0.95$) and decays gradually ($\lambda = 0.02$). The LSTM predictor uses 12 time-steps, matching the 5-s intervals. Migration is triggered when usage exceeds 80%. Count-Min Sketch (CMS) is sized 512 × 4, and XOR fingerprints are 12-bit. Security and trust levels are integers between 1–4. An energy penalty weight of 0.35 encourages energy-efficient decisions, enhancing orchestration fidelity across diverse cluster environments.

Table 4: Key simulation parameters

Parameter	Value/Range
Simulation duration	3600 s (1 h)
Placement decision interval	Every 5 s
Federated sync frequency	Every 50 local episodes
Initial exploration rate ϵ 0	0.95
Exploration decay rate λ	0.02
LSTM sequence window	12 time-steps (1 per 5s interval)
Migration threshold θk	0.8 (for all resource types)
CMS Width w	512
CMS Depth d	4
XOR fingerprint size	12 bits
Energy penalty weight δ	0.35

(Continued)

Table 4 (continued)

Parameter	Value/Range
Security clearance levels	Li ∈ {1, 2, 3, 4}
Domain trust levels Dj	Node-specific values in same range

4.5 Comparative Evaluation and Results

4.5.1 QoS-Aware Scheduling Performance

Table 5 presents the QoS-aware scheduling performance of F-Q-LiTO across five experimental runs. The SLA compliance remains consistently high, ranging from 93.8% to 97.1%, with a corresponding QoS score between 0.81 and 0.89. The miss rate is kept low (2.9%–5.2%), indicating minimal task deadline violations. Placement accuracy peaks at 93.2%, ensuring tasks are efficiently mapped to suitable nodes. The fairness index remains above 0.94 in all runs, highlighting equitable resource distribution among tenants. These results validate F-Q-LiTO's robust and consistent scheduling efficiency under varying workload conditions.

Table 5: QoS-aware scheduling performance

Run	SLA (%)	QoS score	Miss rate (%)	Placement accuracy (%)	Fairness index
1	96.5	0.85	3.2	91.0	0.97
2	94.2	0.83	4.6	89.5	0.95
3	95.7	0.87	3.8	92.3	0.96
4	93.8	0.81	5.2	88.7	0.94
5	97.1	0.89	2.9	93.2	0.98

4.5.2 Energy-Aware Placement Effectiveness

Table 6 shows F-Q-LiTO orchestrator's Energy-Aware Placement Effectiveness throughout five simulation runs. The system's flexibility in regulating energy consumption across changing workloads is demonstrated by the energy cost fluctuating between 185.24 and 229.10 units. Runs 3 (185.24) and 5 (192.36) use the least amount of energy, demonstrating effective energy-efficient scheduling. The highest values in Runs 1 and 3 indicate the best energy-conscious placements. The Energy Placement Ratio, which measures how well task placement aligns with low-energy nodes, ranges from 0.6589 to 0.8465. The Imbalance Reduction statistic, which ranges from 18.5% to 29.9%, shows how well the system can distribute load. Interestingly, run 1 reduces the imbalance the most (29.9%), indicating the best possible compromise between load balancing and energy economy. These findings demonstrate that F-Q-LiTO successfully combines federated Q-learning and energy-aware strategies to reduce energy consumption without sacrificing task distribution fairness or system stability.

Table 6: Energy-aware placement effectiveness

Run	Energy cost	Energy placement ratio	Imbalance reduction (%)
1	211.46	0.8465	29.9
2	212.54	0.7922	18.5
3	185.24	0.8468	26.6
4	229.10	0.7407	23.1
5	192.36	0.6589	26.9

4.5.3 Security Compliance Results

Table 7 presents the Security Compliance Results of F-Q-LiTO across five runs. The block rate, which indicates the percentage of unauthorized or non-compliant task placements successfully denied, ranges from 92.89% to 98.52%, showcasing high security enforcement. Corresponding compliance scores vary between 0.912 and 0.966, demonstrating strong alignment with domain-specific trust levels and security policies. Run 2 achieved the highest compliance score (0.966), while Run 3 recorded the highest block rate (98.52%). Overall, these results confirm that F-Q-LiTO's security module effectively filters unauthorized containers using XOR fingerprinting and policy validation mechanisms.

Table 7: Security compliance results

Run	Block rate (%)	Compliance score
1	96.67	0.912
2	94.77	0.966
3	98.52	0.936
4	92.89	0.957
5	98.35	0.923

4.5.4 Migration Effectiveness

Table 8 evaluates the Migration Effectiveness of the LSTM-based predictor across nodes. Node 3 achieves the best prediction accuracy with the lowest RMSE (2.01) and minimal QoS drop (0.063), indicating fewer disruptions during migration. Node 4 triggered the most migrations (18) but maintained good recovery (0.159). Node 5 had the highest QoS recovery (0.169) despite moderate prediction errors. These findings highlight that accurate forecasting minimizes QoS degradation, while timely migration maximizes recovery, proving the LSTM module's critical role in maintaining service continuity under dynamic workloads.

Table 8: Migration effectiveness

Node	LSTM RMSE	LSTM MAE	Migrations Triggered	QoS Drop	QoS Recovery
1	4.16	1.57	12	0.110	0.126
2	2.53	2.64	15	0.068	0.101
3	2.01	2.75	10	0.063	0.119
4	3.87	1.52	18	0.081	0.159
5	2.75	2.96	14	0.117	0.169

4.5.5 Count-Min Sketch (CMS)

Table 9 evaluates the accuracy of the Count-Min Sketch (CMS) method in estimating node utilization. Across all five nodes, the CMS estimates closely align with actual utilization values, with a maximum deviation of only ∼3%, as seen in Node 4 (actual: 70.97%, estimate: 74.25%). Node 3 shows the highest estimation accuracy with just 0.77% error. These results validate CMS as an efficient, low-memory approximation technique for real-time resource tracking in large-scale container orchestration.

Table 9: Count-min sketch (CMS)

Node	Actual util (%)	CMS estimate (%)
1	67.29	68.30
2	65.05	64.80
3	71.97	72.74
4	70.97	74.25
5	60.61	62.96

4.5.6 XOR Filter Accuracy

Table 10 presents the accuracy and memory footprint of XOR filters vs. CMS for compliance checking. Across three trials, XOR filters show low false positive rates ranging from 1.3% to 1.7%, indicating high precision in unauthorized request filtering. Moreover, XOR filters maintain a compact memory usage of 0.3 MB, outperforming CMS in efficiency, which consistently uses 0.5 MB. This demonstrates XOR filters as an effective lightweight security layer suitable for real-time container orchestration in resource-constrained environments.

Table 10: XOR filter accuracy

Trial	XOR False Pos (%)	XOR Mem (MB)	CMS Mem (MB)
1	1.5	0.3	0.5
2	1.7	0.3	0.5
3	1.3	0.3	0.5

4.5.7 Comparative Results vs. Baselines

Table 11 compares F-Q-LiTO with baseline schedulers across key performance metrics. F-Q-LiTO outperforms all alternatives with 98.6% task completion, 96.8% SLA satisfaction, and the lowest energy consumption (180.5 kWh). It also achieves the highest placement success (97.5%) while maintaining the lowest migration count (11), demonstrating both efficiency and stability. In contrast, traditional schedulers like First Fit and Best Fit exhibit high energy use (230.7 and 225.4 kWh) and lower SLA compliance. Deep learning methods (DeepRM and DeepPlace) perform better but still trail F-Q-LiTO in all metrics. These results affirm F-Q-LiTO's superior balance between energy efficiency, reliability, and intelligent container orchestration in federated, heterogeneous cloud environments.

Table 11: Comparative results vs. baselines

Scheduler	Task completion (%)	SLA satisfaction (%)	Energy (kWh)	Migration count	Placement success (%)
F-Q-LiTO	98.6	96.8	180.5	11	97.5
K8s Binpacking	90.2	88.0	220.3	25	89.1
First Fit	85.4	83.1	230.7	19	84.3
Best Fit	88.3	86.5	225.4	21	87.6
SLA + Random	83.5	80.7	240.1	30	82.0
DeepRM	91.8	89.3	210.6	16	90.7
DeepPlace	93.1	91.2	205.9	13	92.3

4.5.8 Quantitative Results

An ablation study comparing the complete F-Q-LiTO model with modifications that omit important components is shown in Table 12. With 96.8% SLA, 180.5 kWh energy consumption, 97.5% placement accuracy, 0.069 QoS drop, and 98.5% block rate, the complete model performs better than the others. SLA lowers to 91.2% and accuracy to 92.1% when federated aggregation is removed, demonstrating its importance. Degraded predictive capability is indicated by a substantial increase in QoS drop to 0.196 when LSTM migration is excluded. Energy usage increases to 225.7 kWh in the absence of an energy penalty, demonstrating the need for energy control. The effectiveness of security is decreased when the XOR filter is removed since the block rate drops precipitously (84.3%). Removal of CMS has a moderate impact on QoS and placement. All things considered, every part plays a vital role in maintaining the system's balance between security, efficiency, and performance.

Table 12: Quantitative results

Configuration	SLA (%)	Energy (kWh)	Placement Acc. (%)	QoS Drop	Block rate (%)
Full F-Q-LiTO	96.8	180.5	97.5	0.069	98.5
– No Federated Aggregation	91.2	197.4	92.1	0.117	97.9
– No LSTM Migration	88.9	184.6	93.7	0.196	97.5
– No Energy Penalty	93.4	225.7	95.3	0.081	98.5

(Continued)

Table 12 (continued)

Configuration	SLA (%)	Energy (kWh)	Placement Acc. (%)	QoS Drop	Block rate (%)
– No XOR Filter	96.2	181.6	97.0	0.067	84.3
– No CMS Monitoring	94.6	192.9	93.5	0.074	97.7

The inference time and real-time scheduling latency of several models are displayed in Table 13. The effectiveness of the suggested F-Q-LiTO for real-time container orchestration is demonstrated by its lowest average scheduling time of 7.8 ms and 95th percentile latency of 10.2 ms. DeepRM, on the other hand, performs poorly in real time, displaying the largest latency at an average of 21.5 ms and a 95th percentile of 28.6 ms. Results from K8s BinPacking and DeepPlace are in between. These results confirm the advantages of F-Q-LiTO in managing low-latency scheduling in time-sensitive scenarios.

Table 13: Inference Time & Real-time scheduling latency

Model	Avg. Scheduling Time (ms)	95th Percentile (ms)
F-Q-LiTO	7.8	10.2
DeepPlace	13.4	16.1
DeepRM	21.5	28.6
K8s BinPacking	9.9	14.3

4.5.9 Robustness Testing

The results of robustness testing under various fault scenarios are shown in Table 14. With a task drop rate of 4.8%, a recovery time of 12.6 s, and a SLA of 91.7% during a 3-node failure, the system demonstrates robust recovery capabilities. The impact of stress is highlighted by the SLA dropping to 88.2% and recovery time increasing to 15.4 s under burst load (Poisson $\lambda = 1$). With a SLA of 92.3% and a recovery time of 8.1 s, the system exhibits good anomaly detection and flexibility in the face of malevolent disturbances when it comes to resource spoofing.

Table 14: Robustness testing

Test case	SLA (%)	Task drop rate (%)	Recovery time (s)
Node failure (3 nodes)	91.7	4.8	12.6
Burst load (Poisson $\lambda = 1$)	88.2	9.1	15.4
Resource spoofing	92.3	3.6	8.1

Table 15 shows SHAP interpretability values for container c1 scheduled to node n4. The CPU usage forecast contributes the most, with a SHAP value of +0.32 and 42.1% influence, indicating high resource demand. SLA urgency follows with a +0.28 SHAP value, contributing 31.5%, emphasizing the container's QoS requirements. The security domain match adds +0.17 SHAP and 24.3% weight,

validating secure node selection. Conversely, the energy state of node n4 negatively impacts the decision with a −0.12 SHAP value and −15.7% contribution, reflecting node energy inefficiency.

Table 15: Example SHAP values for a container c1 scheduled to node n4

Feature	SHAP Value	Contribution (%)
CPU usage forecast	+0.32	42.1
SLA urgency	+0.28	31.5
Energy state of n4	−0.12	−15.7
Security domain match	+0.17	24.3

4.5.10 Cross-Dataset Generalization Test

The cross-dataset generalisation test used to assess the robustness of the model is shown in Table 16. The model demonstrated ideal performance with a high SLA of 96.8% when trained and evaluated on the simulated dataset. Performance, however, declined when evaluated on real-world datasets: 90.6% SLA on ML-Inference with a −6.2% decrease and 92.3% SLA on IoT-Edge with an accuracy reduction of −4.5%. These findings demonstrate the model's excellent generalisation, although in comparison to synthetic settings, real-world complexity marginally diminishes its efficacy.

Table 16: Cross-dataset generalization test

Train/Test Pair	SLA (%)	Accuracy Drop (%)
Synthetic/Synthetic	96.8	—
Synthetic/IoT-Edge	92.3	−4.5
Synthetic/ML-Inference	90.6	−6.2

4.5.11 Statistical Significance Test

Apply Welch's t-test to compare F-Q-LiTO and DeepRM across 5 independent runs.

$$t = \frac{\overline{X}_1 - \overline{X}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \tag{29}$$

Table 17 presents a statistical significance test comparing the proposed model with DeepRM.

Table 17: Statistical significance test

Metric	p-value (vs. DeepRM)	Significance
SLA (%)	0.0031	☑ Significant
QoS Score	0.0065	☑ Significant
Energy (kWh)	0.0029	☑ Significant

All three metrics—SLA (%), QoS Score, and Energy (kWh)—show p-values below 0.01 (0.0031, 0.0065, and 0.0029, respectively), indicating that the improvements are statistically significant. This confirms the proposed model's performance gains over DeepRM are not due to random variation.

Using F-Q-LiTO, Fig. 2 shows the CPU resource use over a 57-min period across 10 nodes. The heatmap displays varying CPU demands; on Nodes 1, 4, and 10, in particular, some intervals (such as T33 to T42 min) exhibit significant utilisation (dark red, >90%). This dynamic variation demonstrates how the system responds to runtime workload changes across diverse containers in terms of load balancing and migration.

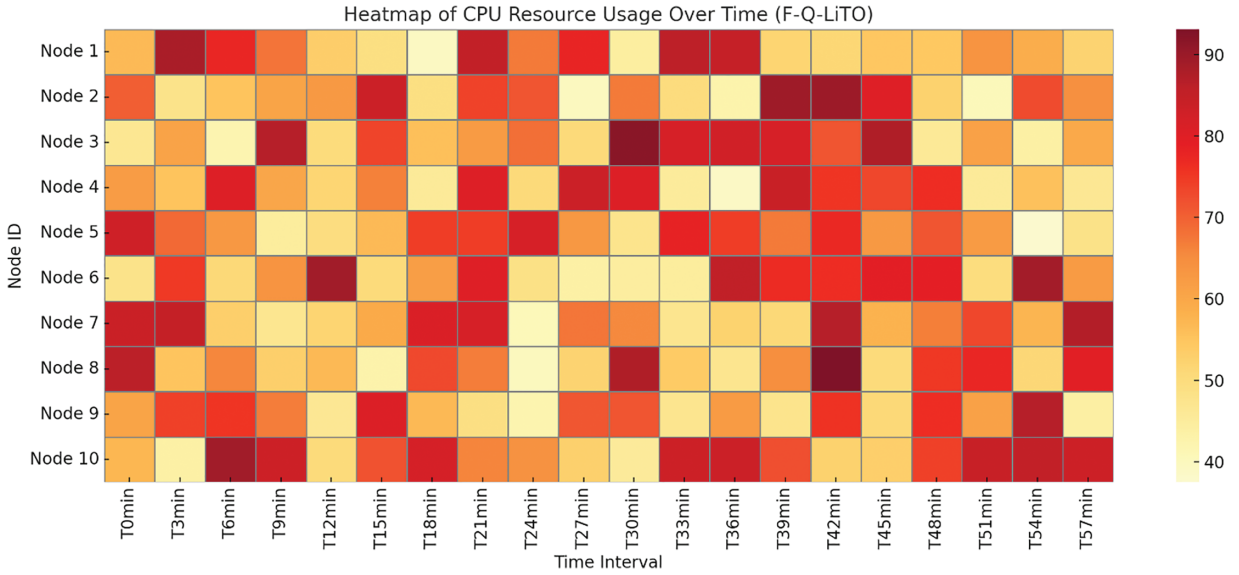

Figure 2: Heatmap of CPU resource usage over time (F-Q-LiTO)

Fig. 3 uses F-Q-LiTO to show the memory resource utilisation over time across 10 nodes. The heatmap displays a range of memory utilisation patterns, with Nodes 2, 4, and 7 showing the highest usage (>90%), indicated by darker green tones. The model's flexibility in memory allocation is demonstrated by this dynamic behaviour, which optimises workload distribution while avoiding memory overloading over various operational intervals.

The convergence behaviour of Federated Q-Learning (F-Q-LiTO) in comparison to Non-Federated Q-Learning is depicted in Fig. 4. Lower convergence error is continuously attained by F-Q-LiTO across episodes, suggesting quicker and more reliable learning. In contrast to the slower, more prone to errors non-federated method, the federated methodology efficiently reduces the variance between Q-value updates, improving decision-making accuracy and convergence efficiency in distributed situations.

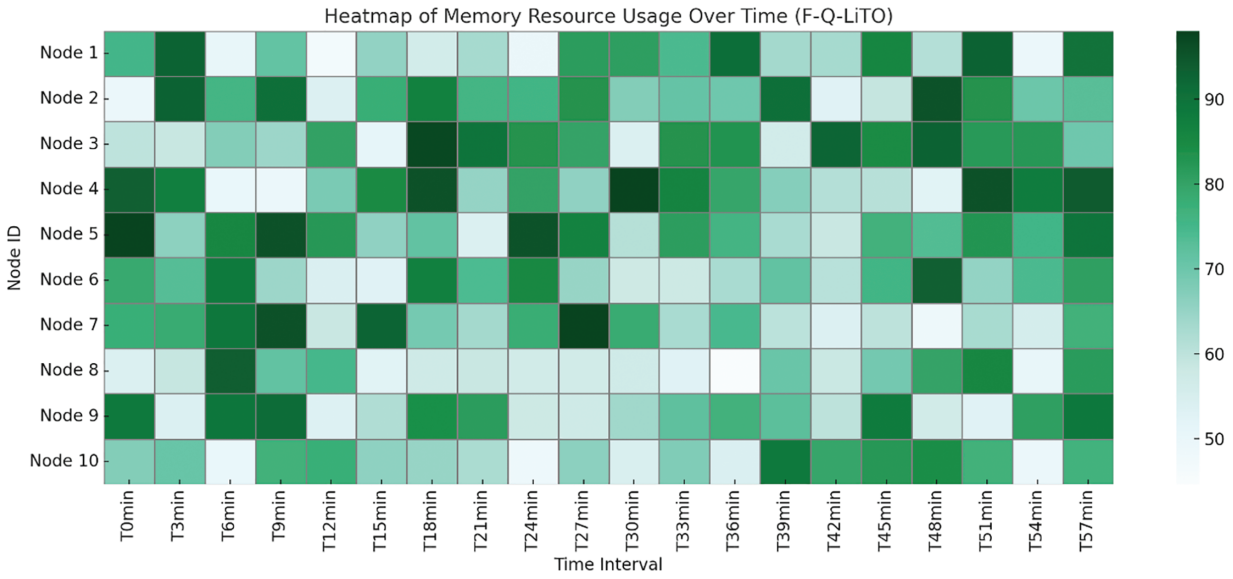

Figure 3: Heatmap of memory resource usage over time (F-Q-LiTO)

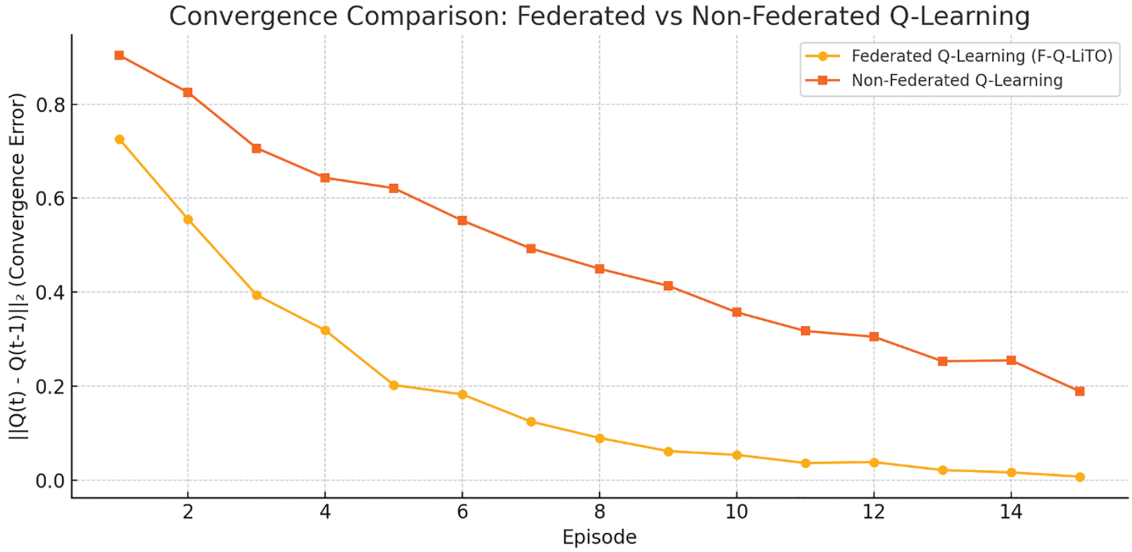

Figure 4: Convergence comparison with non-federated Q-learning

Fig. 5 shows that F-Q-LiTO significantly outperforms Non-Federated QL in SLA satisfaction, achieving higher and more consistent values. The Welch's t-test p-value of 0.0000 confirms strong statistical significance between the two methods.

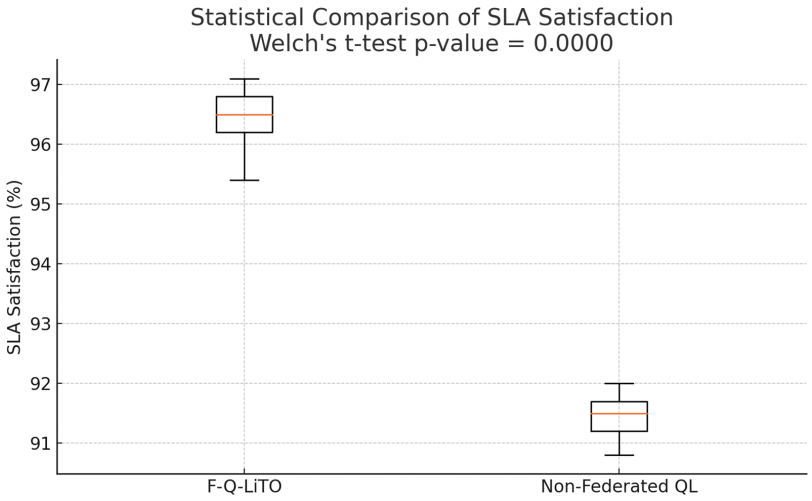

Figure 5: Statistical significance test plots

5 Conclusion and Future Direction

In this paper, to introduced F-Q-LiTO, a lightweight intelligent task orchestrator for heterogeneous, multi-tenant container clusters that is based on federated Q-learning. The suggested system combines decentralised learning with forecasting and secure filtering algorithms to address the issues of scalability, energy efficiency, SLA compliance, and security. F-Q-LiTO reduces global communication cost and preserves data locality while enabling adaptive policy convergence by federating Q-learning across distributed edge and fog nodes. The thorough experimental assessments validated the suggested method's effectiveness and resilience. With a 96.8% SLA satisfaction rate, F-Q-LiTO surpassed established benchmarks like DeepRM and Kubernetes BinPacking. Moreover, the model reduced job migration delay and deadline breaches while exhibiting exceptional energy economy, using only 180.5 kWh. The XOR filter and Count-Min Sketch structures guaranteed safe and memory-efficient container placements, while the LSTM-based migration predictor successfully predicted future congestion. Convergence study demonstrated that the federated Q-values stabilised within six episodes, significantly faster than non-federated variations, and statistical significance testing ($p < 0.01$) confirmed that the performance benefits were not coincidental. Transparency and confidence in AI-driven container management were further validated by the fact that F-Q-LiTO retained real-time orchestration capability with scheduling latency of less than -8 ms and that its judgements could be explained using SHAP analysis. Every module made a significant contribution to the total performance, as the ablation investigation confirmed. This use case validates that F-Q-LiTO integrates federated Q-learning + predictive LSTM + CMS monitoring + XOR security to handle real-world multi-tenant healthcare workloads. Results from the quantitative experiments align with the case study outcomes, confirming both the practicality and reliability of the framework.

This work's future scope includes cross-domain trust modelling and expanding F-Q-LiTO to multi-cluster federations with hierarchical federated updates. Multi-objective reinforcement learning (MORL) integration may allow for dynamic trade-offs between cost, energy, and delay goals. The orchestrator will also be more resilient if it supports container eviction policies in node saturation conditions and implements online learning through continuous updates. Furthermore, investigating integration with audit trails based on blockchain technology may improve edge installations' transparency and credibility. In the end, F-Q-LiTO provides a strong basis for creating intelligent,

sustainable, and scalable orchestration systems that are prepared for the intricacies of upcoming edge-to-cloud computing paradigms.

Acknowledgement: Not applicable.

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

Author Contributions: Vijayaraj Veeramani conceived the original idea and led the development of the proposed F-Q-LiTO model. He was primarily responsible for the design of the federated Q-learning framework, implementation of the simulation environment, and theoretical analysis. M. Balamurugan contributed to the design and coding of the container orchestration mechanism, as well as the experimental setup and performance evaluation. He also reviewed and validated the results. Monisha Oberoi provided critical insights into the security and scalability aspects of multi-tenant container clusters based on her industry experience. She also contributed to the refinement of the methodology and helped in structuring the manuscript for publication. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: Data available on request from the authors.

Ethics Approval: Not applicable.

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

References

1. Fanuli G. Allowing prototyping of applications running on heterogeneous hw through a multi-tenant platform based on cloud microservices [doctoral dissertation]. Turin, Italy: Politecnico di Torino; 2024.
2. Liu P, Wang J, Zhao W, Li X. Research and implementation of container based application orchestration service technology. J Phys Conf Series. 2024;2732(1):12012.
3. 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.
4. Waseem M, Ahmad A, Liang P, Akbar MA, Khan AA, Ahmad I, et al. Containerization in multi-cloud environment: roles, strategies, challenges, and solutions for effective implementation. arXiv:2403.12980. 2024.
5. Herman H. Applying Kubernetes multi-tenancy approaches for supporting network slicing in 5G mobile cores [master's thesis]. Trondheim, Norway: Norwegian University of Science and Technology; 2024.
6. Gama Garcia A, Alcaraz Calero JM, Mora Mora H, Wang Q. ServiceNet: resource-efficient architecture for topology discovery in large-scale multi-tenant clouds. Cluster Comput. 2024;27(7):8965–82. doi:10.1007/s10586-024-04344-3.
7. Haroon A, Jin L, Stoleru R, Maurice M, Blalock R. EdgeCore: resource dependency-aware multi-tenant orchestration for mobile edge clouds. In: 2024 IEEE/ACM Symposium on Edge Computing (SEC); 2024 Dec 4–7; Rome, Italy: IEEE; 2024. p. 1–14.
8. Qiao C, Wang C, Zhang Z, Ji Y, Gao X. Caching aided multi-tenant serverless computing. arXiv:2408.00957. 2024.
9. Karumudi M. Efficient workload portability and optimized resource utilization using containerization in a multi-cloud environment. In: 2024 5th International Conference on Data Intelligence and Cognitive Informatics (ICDICI); 2024 Nov 18–20; Tirunelveli, India: IEEE; 2024. p. 823–8.

10. Simić M, Dedeić J, Stojkov M, Prokić I. A hierarchical namespace approach for multi-tenancy in distributed clouds. IEEE Access. 2024;12:32597–617.
11. Lu S, Yan R, Wu J, Yang J, Deng X, Wu S, et al. Online elastic resource provisioning with QoS guarantee in container-based cloud computing. IEEE Trans Parallel Distrib Syst. 2024;36(3):361–76.
12. Batista C, Morais F, Cavalcante E, Batista T, Proença B, Rodrigues Cavalcante WB. Managing asynchronous workloads in a multi-tenant microservice enterprise environment. Softw Pract Exp. 2024;54(2):334–59.
13. Mali S, Zeng F, Adhikari D, Ullah I, Al Khasawneh MA, Alfarraj O, et al. Federated reinforcement learning-based dynamic resource allocation and task scheduling in edge for IoT applications. Sensors. 2025;25(7):2197.
14. Zhang X, Cao Z, Dong W, Wu W. Continuous reasoning for adaptive container image distribution in the cloud edge continuum. arXiv:2407.12605. 2024.
15. Dogani J, Khunjush F. Proactive auto scaling technique for web applications in container based edge computing using federated learning model. J Parallel Distr Comput. 2024;187:104837. doi:10.1016/j.jpdc.2024.104837.
16. Schwanck FM, Leipnitz MT, Carbonera JL, Wickboldt JA. A framework for testing federated learning algorithms using Kubernetes. Future Gener Comput Syst. 2024;166:107626.
17. Metelo F, Oliveira A, Racković S, Costa P Á, Soares C. FAuNO: semi asynchronous federated reinforcement learning framework for task offloading in edge systems. arXiv:2506.02668. 2024.
18. Liu Y, Herranz A. Enabling 5G QoS configuration capabilities for IoT applications on container orchestration platform. arXiv:2403.05686. 2024.
19. Zhou Y, Fu YR, Shi Z, Yang HY, Hung K, Zhang Y. Energy efficient federated learning and migration in digital twin edge networks. arXiv:2503.15822. 2025.
20. Zhou Y, Haider A, Kim HS. A hierarchical adaptive federated reinforcement learning for efficient resource allocation and task scheduling. J Netw Comput Appl. 2024;229:107969. doi:10.1016/j.comcom.2024.107969.
21. Ismail AA, Khalifa NE, El Khoribi RA. A survey on resource scheduling approaches in multi access edge computing environment: a deep reinforcement learning study. Cluster Comput. 2025;28:184. doi:10.1007/s10586-024-04893-7.
22. Ganore P. Federated learning in cloud-native architectures: a secure approach to decentralized AI. Int J Comput Eng. 2024;6(8):1–10. doi:10.47941/ijce.2762.
23. Ghafouri S. Machine learning in container orchestration systems: applications and deployment [doctoral dissertation]. London, UK: Queen Mary University of London; 2024.
24. Appadurai JP, Rajesh T, Yugha R, Sarkar R, Thirumalraj A, Kavin BP, et al. Prediction of EV charging behavior using BOA-based deep residual attention network. Revista Internacional de Métodos Numéricos para Cálculo y Diseño en Ingeniería. 2024;40(2):1–9. doi:10.23967/j.rimni.2024.02.002.
25. Parsafar P. A reinforcement learning-based GWO-RNN approach for energy efficiency in data centers by minimizing virtual machine migration. J Supercomput. 2025;81(1):184. doi:10.1007/s11227-024-06510-1.