Buletin Ilmiah Sarjana Teknik Elektro
Vol. 8, No. 4, August 2026, pp. 967-995
ISSN: 2685-9572, DOI: 10.12928/biste.v8i4.16274

Meta-Heuristic Hyperparameter Optimization for Q-Learning: A Snake Optimizer Approach to Computation Offloading in the Industrial Internet of Things

Hayder Salman Dawood, Ismail Fauzi Bin Isnin, Muhalim Bin Mohamed Amin, Ahmad Najmi Amerhaider Nuar
[1] Faculty of Computing, Universiti Teknologi Malaysia (UTM), Johor Bahru, Malaysia

ARTICLE INFORMATION

Article History:

Received 27 March 2026
Revised 06 July 2026
Accepted 24 July 2026

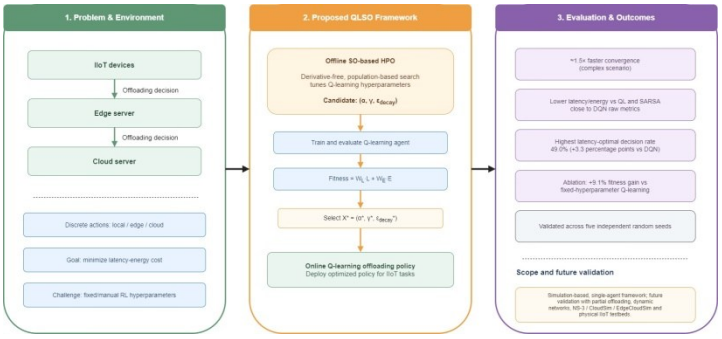
Keywords:

Computation Offloading;
Hyperparameter Tuning;
Industrial Internet of Things;
Meta-Heuristic Optimization;
Reinforcement Learning;
Snake Optimizer;
Three-Tier Architecture

Corresponding Author:

Hayder Salman Dawood,
Faculty of Computing, Universiti Teknologi Malaysia,
Johor Bahru, Malaysia.
Email:
dawood-20@graduate.utm.my

ABSTRACT



The proliferation of computationally intensive Industrial Internet of Things (IIoT) applications requires offloading policies that reduce latency and energy consumption under constrained device, edge, and cloud resources. Although reinforcement learning (RL) is suitable for sequential offloading decisions, its performance is highly sensitive to hyperparameters that are often fixed or manually tuned. The research contribution is an automated QLSO framework that uses the Snake Optimizer (SO) to tune the learning rate, discount factor, and exploration-decay rate of tabular Q-learning (QL) in a three-tier IIoT-edge-cloud architecture. Unlike generic external Hyperparameter Optimization (HPO) methods, QLSO embeds SO as a derivative-free, population-based tuning layer within the QL offloading workflow, where candidate hyperparameters are evaluated based on latency-energy-aware performance under three-tier IIoT resource constraints. The problem is formulated as a Markov decision process with discrete local, edge, and cloud execution actions, and candidate hyperparameters are evaluated through episodic training followed by greedy validation. Across five independent random seeds, QLSO converged about 1.5× faster in the complex scenario and achieved lower latency and energy than the QL and SARSA baselines, while remaining close to the strongest raw-metric baseline, DQN (0.0493 s and 1.72×10^{-2} J for QLSO versus 0.0488 s and 1.65×10^{-2} J for DQN). However, QLSO achieved the highest latency-optimal decision rate (49.0%, +3.3 percentage points over DQN), and the ablation study showed a 9.1% fitness improvement over fixed-parameter QL. The framework remains simulation-based and single-agent, requiring validation with partial offloading, dynamic network models, and realistic platforms such as NS-3, CloudSim, EdgeCloudSim, or physical IIoT testbeds.

Document Citation:

H. S. Dawood, I. F. B. Isnin, M. B. M. Amin and A. N. A. Nuar, "Meta-Heuristic Hyperparameter Optimization for Q-Learning: A Snake Optimizer Approach to Computation Offloading in the Industrial Internet of Things" *Buletin Ilmiah Sarjana Teknik Elektro*, vol. 8, no. 4, pp. 967-995, 2026, DOI: 10.12928/biste.v8i4.16274.

1. INTRODUCTION

The Industrial Internet of Things (IIoT) is an innovative paradigm, whereby the concept of the Internet of Things (IoT) is implemented in the industrial sector, and thus, develops a connected ecosystem of devices and services through intelligent components in the form of sensors and controllers [1]. Such integration of IIoT into manufacturing allows efficient distribution and optimization of production factors [2]. The rapid development and integration of heterogeneous IIoT devices, however, generate large amounts of data within short periods which present significant challenges related to data management, real-time processing as well as resource limitation [3]. A large number of IIoT terminal devices are limited by small battery and processing power, making it difficult to execute computationally intensive and delay sensitive tasks locally [4]. Unexpected delays or malfunctions in the implementation of computational tasks may create major safety issues. Accordingly, the need to have strong solutions is emphasized [5].

Computation offloading has been identified as one of the key solutions to overcome the inherent limitations of IIoT devices and highly demanding characteristics of the industrial environment. Computation offloading is the process of transferring computational workloads of resource constrained local devices to external servers that have greater computing power [6]. The benefits of the IIoT offloading computation include ensuring improved rates of accomplishing tasks, better energy efficiency of devices, and better quality of service (QoS) of users [7]. An efficient offloading mechanism has the ability to minimize the turnaround time of tasks and the system energy use [8].

To have an efficient offloading of computation, a multi-tier collaborative framework is often necessary, consisting of IIoT devices, edge servers, and centralized cloud servers. The interoperability between IIoT devices, edge servers, and centralized cloud servers is an ingredient component to attain the best industrial computing performance. Although cloud data centers offer substantial computational, communication, and caching capacity, their geographical remoteness can introduce transmission delays, making them unsuitable for latency-critical industrial applications [9]. Edge computing eliminates this drawback by functioning within the network edge to provide real time processing and analysis of industrial data near the source, through the deployment of computational means close to IIoT devices to enable real-time processing and feedback [10], [11]. However, edge servers have disadvantages in computational resource and processing power compared to cloud services, which may result in overutilization of resources and increase processing latency if they are not properly managed [12]. As a result, the corresponding balancing workloads and exploiting the benefits of each tier can be achieved with the cooperation of the IIoT devices, edge servers, and cloud servers, which, in turn, can increase the overall performance of the systems and their resource usage and ensure the efficiency and responsiveness of the IIoT operations [13].

Computation offloading is an essential part of IIoT environment, where multi-objective optimization includes trade-offs between latency and energy consumption. Such metrics are in conflict, and models are required to streamline their trade-off. Therefore, the optimization problems aim to reduce a weighted sum of task execution time and energy consumption [14]. In the real-world industrial setting, goals including energy consumption and latency are equally important and require parallel optimization [15]. Accordingly, the offloading of computation aims at reducing energy consumption and delay to meet the energy efficiency and latency constraints in the IIoT environment [16].

Reinforcement Learning (RL) is a promising method to cope with the dynamic and complex nature of the IIoT environment and the multitier offloading process. The RL algorithms can also interact with the environment and make an adjustment in offloading decisions based on changing conditions and varying requirements [17]. This makes RL specifically well-adapted for solving continuous decision-making-problems in dynamic industrial settings when traditional optimization processes are often intractable due to their extreme complexity and because they typically require prior knowledge [18]. With such advantages, RL has increasingly been used in computational offloading scenarios, where it can effectively tune the trade-off between task completion time and energy usage [2].

But RL algorithms depend on the effectiveness of hyperparameter tuning [19], which directly affects learning efficiency, convergence speed, and decision quality. Without proper tuning, RL algorithms may fail to explore the solution space sufficiently, resulting in suboptimal offloading decisions. Most recent studies that apply RL to this area assume fixed hyperparameter settings, with only a few investigations attempting manual tuning through iterative experimental procedures. Manual tuning may produce performance improvements, but it is effort- and time-intensive. Moreover, these studies often search a limited range of parameter values, and the optimal values are highly environment-dependent. This limits the ability to generalize these systems under dynamically changing conditions. Therefore, automated hyperparameter optimization is needed in RL-based offloading models.

Beyond hyperparameter selection, RL-based IIoT offloading also faces challenges related to state-space complexity, reward design, and multi-agent coordination. In tabular RL, task, resource, and network variables can increase the number of state-action combinations, while the reward formulation determines the latency-energy trade-off learned by the agent. In addition, centralized single-agent decision-making may face scalability limits in large distributed IIoT networks. These issues remain important directions for future work; however, the present study focuses specifically on automated hyperparameter tuning to improve Q-learning-based IIoT offloading.

In order to optimize the performance of RL algorithms, advanced meta-heuristic optimization methods have been embraced. Meta-heuristic optimizers represent a family of population-based search algorithms that do not require gradient information, making them particularly suitable for black-box hyperparameter optimization in non-convex RL landscapes. Among recent meta-heuristic optimizers, the Snake Optimizer (SO) algorithm, proposed by Hashim and Hussien in 2022 [20], which can enhance global exploration of the search space and promote convergence toward high-quality solutions. The optimizer provides an effective system for tuning the hyperparameters of the RL algorithms, thus improving their efficiency and the ability to adjust to the changing network conditions and changing workloads in IIoT environments. In this study, the Snake Optimizer was selected for three principled reasons: (1) its population-based exploration-exploitation mechanism enables efficient navigation of the three-dimensional hyperparameter space with limited function evaluations; (2) its exploration phase provides sufficient diversity to escape local optima that plague gradient-based methods, while its exploitation phase ensures rapid convergence to high-quality solutions; and (3) empirical ablation studies demonstrate that even the vanilla SO variant improves fitness by 9.1% over fixed hyperparameters, confirming its effectiveness in this specific application domain. Compared with gradient-based hyperparameter optimization methods, which require differentiability, Bayesian optimization with Gaussian process priors, which often requires careful kernel selection and may require additional handling for discrete parameter spaces, and grid/random search, which can become computationally expensive in high-dimensional spaces, SO leverages an exploration-exploitation balance mechanism that enables efficient navigation of the non-convex hyperparameter landscape without requiring gradient information, although it still relies on repeated function evaluations. While it is acknowledged that a comprehensive comparison with Bayesian Optimization (BO), Particle Swarm Optimization (PSO), Genetic Algorithms (GA), and population-based training methods would strengthen the contribution, this work focuses on demonstrating the efficacy of SO-based hyperparameter optimization for tabular RL, which represents a methodology that can be extended to alternative meta-heuristic optimizers in future studies.

The key contributions and findings of the current study can be summed up as follows:

- We suggest an RL-based computational offloading framework that can leverage cooperative abilities of a three-tier IIoT-edge-cloud computing arrangement to dynamically determine the ideal offloading locations through IIoT environments.
- We also utilize an adaptive hyperparameter optimization mechanism that automatically tunes important hyperparameters of RL during the training process and therefore significantly increases convergence speed and stability in comparison with traditional fixed-hyperparameter algorithms.
- We conducted experiments in various situations and compared the proposed algorithm to three benchmark RL-based algorithms. The analysis shows that our solution is capable of operating in a wide range of conditions, thus proving its effectiveness and feasibility in IIoT implementations.
- We provide rigorous statistical validation including multi-seed experiments (five independent random seeds), 95% confidence intervals, and ablation studies to isolate the contribution of the SO enhancement. This methodology enables reproducible and statistically sound performance comparisons.

The research contribution of this work is threefold. First, we demonstrate that automated hyperparameter optimization via meta-heuristic search can substantially improve the performance of classical RL algorithms in practical IIoT scenarios. Second, we justify the selection of the SO over alternative optimization methods based on its demonstrated efficacy in balancing exploration and exploitation during hyperparameter search. Third, we establish that even modest hyperparameter tuning (optimizing learning rate, discount factor, and exploration decay (ε_{decay})) yields statistically significant performance gains in offloading tasks, suggesting that hyperparameter optimization should be treated as a first-class consideration in RL-based offloading studies, equivalent in importance to algorithmic complexity and architecture design.

2. RELATED WORK

In this section, the literature on the extant studies on RL-based computation offloading in IIoT environments is reviewed. The literature is categorically divided based on the hyperparameter management

strategies used in different studies, whereby it is categorized as Fixed Hyperparameters, Manual Tuning, Automated Adjustment and Not Specified. Individual categories define methodological innovations, performance accomplishments, and inherent constraints, and the hyperparameter management strategies used in different studies in particular. This systematic study forms the foundation for locating the research deficiencies that drive the current study.

2.1. Fixed Hyperparameters

A dominant portion of the reviewed studies relied on fixed or constant hyperparameter settings in RL- and deep reinforcement learning (DRL)-based offloading for IIoT environments. These works covered different learning designs and agent deployments, such as DQN-based offloading and edge selection with queuing, processing, and transmission delays [21], Q-learning and DDPG for delay-sensitive Mobile Edge Computing (MEC) with reported latency reductions [7], and DDQN-based schemes that prioritize high-priority tasks with multiple offloading options [22]. Semi-distributed actor-critic approaches were also used for large-scale IIoT offloading [23], and distributed actor-critic was adopted to maximize quality of experience (QoE) under changing conditions [24]. In addition, multi-agent methods such as MADDPG and its variations were used in end-edge-cloud collaboration systems [25] and on edge servers [9], while per-device multi-agent models were also considered with partial offloading and mobility aspects [26]. However, across these studies, the use of fixed or predefined hyperparameters was repeatedly associated with practical constraints, such as slower convergence in decision-making [7], limited suitability across different operational settings [25], limited ability to adapt to changing conditions [9], and an explicit need to adjust hyperparameters dynamically [24]. Overall, these studies show that fixed hyperparameter settings remain the dominant practice in RL/DRL-based offloading research, but they may limit adaptability and generalization under changing IIoT conditions.

2.2. Manual Tuning

A smaller portion of studies conducted manual tuning through empirical evaluation, validation tests, or sensitivity analysis. In these works, manual tuning was used to establish suitable hyperparameters to particular system needs [27], to determine a suitable learning rate with sensitivity in convergence behaviour [28], and to select hyperparameters manually in queue-aware deep RL settings [29]. Other studies manually tuned hyperparameters through empirical evaluation in ε-greedy based algorithms [2], or conducted systematic sensitivity analysis to optimize hyperparameters in MEC federation systems [30]. Empirical experiments also used a variety of hyperparameter settings to trade-off between learning rate and batch size effects, where large learning rates led to local rather than global optima and small batch sizes slowed convergence [31]. Similarly, different learning rates and batch sizes were manually validated and tested in multi-edge environments [32], and manual hyperparameter fine-tuning was used in multi-agent DRL studies guided by the extant literature [33]. These studies collectively show that hyperparameter selection has a critical impact on algorithm performance. However, manual approaches remain limited because they test only restricted sets of values and often produce environment-specific optimal settings, while practical issues such as scalability, complexity, and IIoT constraints continue to appear across studies.

2.3. Automated Adjustment

Within the reviewed literature, automated adjustment strategies were less common. A representative example is the study by Zhang *et al*. [13], where the experiment used a cosine annealing schedule of the learning rate while keeping all other hyperparameters constant. However, the study reported that, under its centralized single-agent setting, computational complexity increased in large networks, and the effectiveness of cooperation decreased as the geographic distance between edge computing servers (ECS) devices grew. This indicates that automated adjustment has been only partially explored, with most efforts focusing on limited parameter adaptation rather than comprehensive hyperparameter optimization.

2.4. Not Specified

A further issue appears in studies that did not specify hyperparameter configurations. For example, Cheng *et al*. [34] did not present the configuration of hyperparameters used, and Chen *et al*. [35] did not specify the hyperparameters settings, which makes these studies hard to replicate. This “Not Specified” category directly limits reproducibility and weakens the ability to compare results reliably across different approaches and environments. This category highlights a clear reproducibility gap, since missing hyperparameter details make it difficult to repeat experiments or compare results fairly across studies.

Based on the reviewed literature, three gaps are identified that motivate this study. Many studies of the offloading of RL/DRL rely on fixed or manually selected hyperparameters, and the performance reported depends on undocumented design choices [2],[7],[30]–[34],[36],[9],[21],[22],[25]–[29]. Second, previous work tends to focus on the complexity of the algorithms and neglect to focus on reproducible hyperparameter optimization (HPO) protocols, validation seeds, and tuning budgets. Third, other AutoRL and HPO research has focused on BO, evolutionary strategies, GA, PSO, population-based training, and meta-gradient learning [37]–[42], but these studies mainly address general RL, AutoRL, or HPO settings rather than three-tier IIoT offloading models. This encourages a targeted study of SO-based HPO for Q-learning when the search bounds, fitness evaluation rules and the reporting of computational-overhead are explicitly stated.

This gap in the systematic reporting of hyperparameter optimization is further reflected in more recent studies. Qin *et al*. [43] proposed a DRL-based offloading algorithm with density clustering and ensemble learning, while Zhang *et al*. [13] proposed an improved soft actor-critic-based cooperative partial task offloading and resource allocation algorithm for IIoT applications. Likewise, Chai *et al*. [32] created a dynamic queuing model based on DRL for distributed task offloading in MEC, and Cai *et al*. [9] proposed a multitask multiobjective DRL-based computation offloading technique for IIoT. Although these studies report some training or simulation parameters, they do not provide a systematic HPO protocol with clearly defined search spaces or detailed tuning procedures. This limited documentation adds to the challenges of reproducibility and highlights the importance of standardized reporting in RL/DRL offloading research.

3. SYSTEM MODEL AND PROBLEM FORMULATION

In this section, a three-layer IIoT-edge-cloud architecture is given. The problem of computational offloading is formulated as multi-objective optimization. With latency and energy consumption model formulated explicitly by layer of processing, the proposed formulation provides a base for the design and evaluation of enhanced RL-based computation offloading strategies.

3.1. System Model

An IIoT-edge-cloud architecture in this study is represented in Figure 1. The IIoT layer is a collection of IIoT devices in numbers of N, and these devices create tasks. The processing capacity of each device is constrained by size and energy. The edge layer is made up of a base station that has an edge server which is more powerful than the IIoT devices but not as powerful as the cloud server, and it provides computational services to IIoT devices located in the area covered by the server. The cloud layer provides significant computational power, but its remoteness subjects the transmission latency to a higher value.

3.2. Task Model

Suppose that N IIoT devices, numbered by index d and $d \in \{1,2,3,\ldots,N\}$, produce tasks in discrete time steps. At each time step, the number of tasks generated at each device follows a Poisson distribution with the mean λ, where λ represents the average number of tasks generated per time step [44]. Each task T_d generated by device d is represented by a two-tuple, $T_d = \{DS_d, CI_d\}$, where DS_d denotes the input task data size (in bits), and CI_d represents the computational intensity (CPU cycles per bit). The total computational requirement for task T_d is:

$$C_d = DS_d * CI_d \quad (1)$$

A task Td can be processed in one of three modes: (1) locally on the IIoT device; (2) offloaded to an edge server over a wireless link; (3) transferred to the cloud through the base station which, in its turn, is connected to the cloud server by a wired connection. Since the result size of the processed task is negligible compared with the size of input task data, the latency and the energy consumption in transmitting downlink data are neglected [45][46].

3.3. Offloading Model

In the proposed system model, an intelligent agent makes binary offloading decisions (choosing between full local execution or full offloading). In the case of offloading, a multi-layer destination selection for each generated task takes place, determining whether to offload it to an edge server or to the cloud server. This binary formulation is adopted as the modeling scope of the present study to evaluate SO-based HPO under a controlled offloading setting. This offloading decision is modeled as a discrete action $a_d \in \{0,1,2\}$ for task T_d

initiated by device d, where $a_d = 0$ represents local execution, $a_d = 1$ represents offloading to the edge server, and $a_d = 2$ represents offloading to the cloud server.

At any time, there can be one execution location for each task, thus optimizing the trade-off between execution energy and latency. This action space is discrete, providing a single mathematical abstraction of evaluation and learning of optimal offloading policies on the three-layer IIoT system structure, and thus enables systematic optimization of policies through RL algorithms.

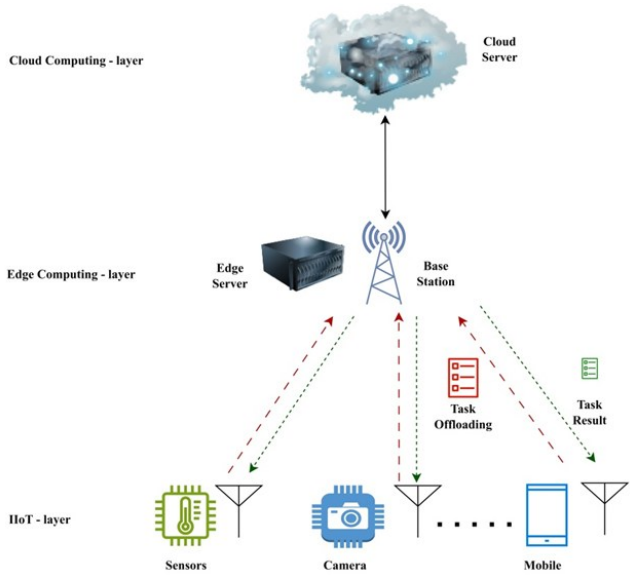


Figure 1. System Architecture of the Three-Tier IIoT-Edge-Cloud Computation Offloading Framework

3.4. Computational Models

The computational models for task execution across the IIoT device, edge, and cloud layers are detailed in this section.

3.4.1. Local Computing

When the IIoT device processes the task locally, latency as well as energy consumption is determined by the computing capabilities of the IIoT device. The latency for local execution is:

$$L_{IIoT}^{LC} = \frac{C_d}{F_{IIoT}} \tag{2}$$

Where, L_{IIoT}^{LC} denotes the computation latency at the IIoT device, and F_{IIoT} denotes the computation resource of the IIoT device.

The model of the energy consumption in the IIoT device is as:

$$E_{IIoT}^{LC} = K * {F_{IIoT}}^2 * C_d \tag{3}$$

Where, E_{IIoT}^{LC} denotes energy consumption at the IIoT device, and K denotes the effectively switched capacitance. Since both latency and energy consumption result only from computation, no transmission overhead is considered in this model.

3.4.2. Edge Computing

For a task generated by an IIoT device and which is offloaded to the edge server, the total latency comprises edge computation time and wireless transmission delay. Therefore, the total latency is given by:

$$L_{IIoT}^{EC} = L_{IIoT,Edge}^{comp} + L_{IIoT,Edge}^{trans} \tag{4}$$

Where, L_{IIoT}^{EC} denotes total latency for edge computing mode, $L_{IIoT,Edge}^{comp}$ denotes computation latency at the edge server, and $L_{IIoT,Edge}^{trans}$ denotes transmission latency when the IIoT device offloads a task to the edge server.

Both of $L_{IIoT,Edge}^{comp}$ and $L_{IIoT,Edge}^{trans}$ can be obtained as:

$$L_{IIoT,Edge}^{comp} = \frac{C_d}{F_{Edge}} \tag{5}$$

$$L_{IIoT,Edge}^{trans} = \frac{DS_d}{R_{IIoT}^{b}} \tag{6}$$

Where, F_{Edge} computation resource of the edge server, and R_{IIoT}^{b} denotes the transmission rate between an IIoT device and the edge server.

The total energy consumption at the IIoT device when offloading a task to the edge server comprises energy consumption of transmission and energy consumption of standby that can be got as:

$$E_{IIoT}^{EC} = P_{IIoT}^{t} L_{IIoT,Edge}^{trans} + P_{IIoT}^{s} L_{IIoT,Edge}^{comp} \tag{7}$$

Where, E_{IIoT}^{EC} denotes total energy consumption when the task is executed at the edge server, P_{IIoT}^{t} denotes transmission power of an IIoT device when a task is being relayed to the edge server, and P_{IIoT}^{s} represents the standby power that can be observed in case the IIoT device goes into standby mode, waiting for the result from the edge server.

3.4.3. Cloud Computing

In the cloud computing, the task is offloaded from the IIoT device to the base station first. Then, the base station forwards the task to the cloud server. The total latency in this case is calculated by:

$$L_{IIoT}^{CC} = L_{IIoT,Cloud}^{comp} + L_{IIoT,Cloud}^{trans} \tag{8}$$

$$L_{IIoT,Cloud}^{comp} = \frac{C_d}{F_{Cloud}} \tag{9}$$

Where, L_{IIoT}^{CC} denotes the total latency when the task is executed at the cloud server, $L_{IIoT,Cloud}^{comp}$ denotes computation latency at the cloud server as calculated in equation (9), F_{Cloud} denotes computation resource of the cloud server, and $L_{IIoT,Cloud}^{trans}$ denotes total transmission latency when sending a task from the IIoT device to the cloud server, which is calculated as :

$$L_{IIoT,Cloud}^{trans} = L_{IIoT,Edge}^{trans} + L_{Edge,Cloud}^{trans} \tag{10}$$

$$L_{Edge,Cloud}^{trans} = \frac{DS_d}{R_{Edge}^{t}} \tag{11}$$

Where, $L_{Edge,Cloud}^{trans}$ denotes transmission latency when the base station offloads a task to the cloud server, and R_{Edge}^{t} refers to the base station transmission rate.

The energy consumption for the IIoT device during the cloud offloading is:

$$E_{IIoT}^{CC} = P_{IIoT}^{t} L_{IIoT,Edge}^{trans} + P_{IIoT}^{s} (L_{IIoT,Cloud}^{comp} + L_{Edge,Cloud}^{trans}) \tag{12}$$

Where, E_{IIoT}^{CC} denotes total energy consumption when the IIoT device offloads a task to the cloud server for processing.

3.5. Problem Formulation

The focus of this study is to reduce the weighted sum of latency-energy consumption for computation offloading in the three-layer IIoT-edge-cloud system. Each IIoT device generates tasks, where each task is characterized by its data size and computation intensity. The aim is to find the best execution location (an IIoT device, an edge server or a cloud server) for every computational task so that the weighted sum of latency and energy consumption is minimized.

3.5.1. Decision Variables

As introduced in Section 3.3, the offloading decision for task T_d is represented by the discrete action $a_d \in \{0, 1, 2\}$.

To formulate the optimization problem, we introduce a binary decision indicator variable X_d^m for each task T_d and execution mode $m \in \{LC, EC, CC\}$, where LC denotes local computing, EC denotes edge computing, and CC denotes cloud computing. The variable X_d^m is defined as:

$$X_d^m = \begin{cases} 1, if\ task\ T_d\ is\ executed\ in\ mode\ m, \\ 0, otherwise, \end{cases}$$

For all $d \in \{1, 2, \dots, N\}$, $m \in \{LC, EC, CC\}$

The relationship between the action variable a_d and the indicator variable X_d^m is given by:

$$X_d^{LC} = \mathbb{1}_{\{a_{d=0}\}}, X_d^{EC} = \mathbb{1}_{\{a_{d=1}\}}, X_d^{CC} = \mathbb{1}_{\{a_{d=2}\}} \quad (13)$$

Where $\mathbb{1}_{\{condition\}}$ is the indicator function that is equal to 1 when the condition is true and 0 otherwise.

3.5.2. Constraints

Since each task can be assigned to only one execution mode at a time, the following constraint must be satisfied:

$$X_d^{LC} + X_d^{EC} + X_d^{CC} = 1, \forall d \in \{1,2,\dots,N\} \quad (14)$$

Additionally, the decision variables are binary:

$$X_d^m \in \{0,1\}, \forall d \in \{1,2,\dots,N\}, \forall m \in \{LC, EC, CC\} \quad (15)$$

3.5.3. Cost Function

The cost of processing task T_d at execution mode m is defined as a weighted sum of latency and energy used:

$$Cost_d^m = W_L . L_d^m + W_E . E_d^m \quad (16)$$

Where, L_d^m is the latency for executing task T_d at the mode m, as defined in equation (2), equation (4), and equation (8) for $m \in \{LC, EC, CC\}$, respectively, E_d^m is the energy consumption for executing task T_d at mode m, as defined in equation (3), equation (7), and equation (12) for m ∈ {LC, EC, CC}, respectively, and W_L, W_E are non-negative weight coefficients satisfying:

$$WL + WE = 1, WL, WE \geq 0 \quad (17)$$

The weights W_L and W_E were fixed to maintain a consistent latency-energy trade-off across algorithm comparisons. Two representative weighting configurations were also examined, but a broader sensitivity analysis over wider latency-energy trade-offs is left for future work.

This cost function formulation follows the standard weighted-sum approach commonly used in multi-objective optimization for task offloading systems [2],[9],[34],[47]. The cost function is a decision criterion to examine and compare the execution modes.

The two coefficients W_L and W_E allow the system to trade-off the latency and energy consumption as per application specific needs. A higher W_L value gives a high priority to latency reduction; a higher W_E value, on the other hand, increases energy efficiency. This approach is consistent with current paradigms in the literature of computation-offloading, in which weighted-sum formulas are widely used to represent execution costs.

3.5.4. Optimization Objective

The optimization problem aims at minimizing the aggregate cost of all the tasks that are generated by the N IIoT devices:

$$\min_{X_d^m} \sum_{d=1}^{N} \sum_{m \in \{LC,EC,CC\}} X_d^m . Cost_d^m \quad (18)$$

Similarly, when equation (16) is replaced, the objective can be modified as:

$$\min_{X_d^m} \sum_{d=1}^{N} \sum_{m \in \{LC,EC,CC\}} X_d^m . (W_L . L_d^m + W_E . E_d^m) \quad (18)$$

Subject to constraints equation (14), equation (15), and equation (17).

The optimization problem of full optimization can be stated as follows:

$$\min_{X_d^m} \sum_{d=1}^{N} [X_d^{LC}.Cost_d^{LC} + X_d^{EC}.Cost_d^{EC} + X_d^{CC}.Cost_d^{CC}] \quad (20)$$

Subject to constraints equation (14), equation (15), and equation (17).

This formulation represents a combinatorial optimization problem where the decision space grows exponentially with the number of devices and execution modes. In particular, the search space for the N devices making single-choice offloading decisions among three execution modes is 3^N possible assignments. This results in 59,049 different offloading configurations for N = 10, as in our experimental setup. The computational hardness of this problem comes from two main sources: (1) the solution space grows exponentially as the number of devices or tasks grows, making it impractical to exhaustively enumerate the solutions for a practical system size; and (2) when the decisions are considered under shared edge computation and communication limits, the offloading choices of different devices become coupled, so the decision of one device can affect the available resources and resulting cost for other devices. The stochastic nature of task arrivals further increases the uncertainty of the optimization landscape at each decision epoch. We do not explicitly model queuing dynamics or partial offloading; however, the NP-hard characterization is made under the shared-resource contention condition, not from queuing or partial offloading alone. Under these coupled resource constraints, the offloading decision problem can be characterized as NP-hard [48][49]. Formally, the decision problem is closely related to the multi-dimensional knapsack problem or the generalized assignment problem, both of which are known to be NP-hard. The exponential solution space (3^N) means that even for moderate values of N (e.g., N = 15), the search space exceeds 14 million configurations, rendering brute-force approaches impractical. Consequently, RL-based approaches with automated hyperparameter optimization provide a tractable learning-based alternative for obtaining high-quality offloading decisions without exhaustive enumeration.

4. PROPOSED ENHANCED Q-LEARNING ALGORITHM BASED ON SNAKE OPTIMIZER ALGORITHM

In this section, a RL-based computation offloading framework is introduced to minimize latency and energy usage in dynamic conditions. The suggested framework (QLSO) combines the Q-Learning algorithm (QL) and Snake Optimizer (SO) to automatically adjust the main hyperparameters of the QL algorithm to improve the convergence behavior and decision-making in complex IIoT environments. The framework evolves the best offloading policies that combine latency and energy usage on the three-layer architecture, through the exploitation of the exploratory efficiency of QL and an optimization strategy adopted by SO.

The environment model follows a three-layered architectural paradigm that captures the fundamental components inherent to an industrial deployment. Every episode is a simulation of the arrival of tasks generated by IIoT devices and each task is characterized by the size of data and the intensity of computation. The state space of the environment at every time step is made up of the characteristics of new generated tasks, the computational capacity and availability of IIoT devices, edge servers, and cloud resources, and the existing network conditions. Such a state representation allows the RL agent to map states to optimum actions that reduce time to complete tasks and energy usage. For tabular Q-learning, the state variables were represented using discrete simulation levels based on the predefined simulation parameter ranges, allowing each observed state to be mapped to a finite Q-table entry.

The action space consists of three discrete offloading choices, namely, execution on the IIoT device (local execution), offloading to the edge server, and offloading to the cloud server, represented by $a_d \in$ {local, edge, cloud}. Every action carried out includes the related values of latency and energy usage based on the task characteristics and the current conditions in the system. The agent of RL is given a scalar reward, denoted by r, obtained as a weighted sum of the negative latency and energy consumption. This reward is expressed as:

$$r = -(WL \cdot L + WE \cdot E) \quad (21)$$

L is the latency incurred when performing a task and E is the energy used. The negative sign ensures that the minimization of the two metrics is equivalent to maximization of the reward.

For clarity and consistency throughout this section, Table 1 presents the definitions of all key symbols and variables used in the QLSO framework formulation.

4.1. Q-Learning Algorithm

Determining an ideal offloading computation strategy in IIoT environments is a challenging task due to their dynamism, limited resource capabilities of devices, and intrinsic tradeoff between execution time and energy use. In order to address these difficulties, the current work will use the QL algorithm [17] integrated with SO to automatically tune its main hyperparameters. Being a model free RL approach, QL is capable of running without explicit knowledge about the system dynamics. QL algorithm learns the best decision-making policies by updating a Q-table repeatedly in response to agent-environment interactions. Further, QL algorithm has comparatively low computational complexity, which makes it practical to run in resource-constrained devices, and its off-policy learning algorithm also has the advantage of exploring the state-action space and driving the algorithm towards the most optimal offloading policies.

In a number of strategic choices, Q-Learning was intentionally chosen as the lightweight baseline RL algorithm. First, the discrete and small action space (|A| = 3) of the suggested three-layer architecture does not necessarily require neural-network-based function approximation methods such as those used in DQN or DDPG. Second, the tabular representation of Q-Learning makes it possible to implement the algorithm directly on edge devices, which have resource constraints, without having to use GPU-based inference, which fits well with the practical IIoT deployment considerations. Third, and most importantly, the contribution of this work does not consist in the suggestion of a new RL algorithm, but in the fact that automated hyperparameter optimization through a meta-heuristic search can considerably improve even classical RL algorithms. In the event that SO-based tuning outperforms untuned DQN and SARSA, this confirms the wider hypothesis that hyperparameter optimization should be treated with the same consideration as the complexity of algorithm in the context of RL-based offloading studies.

The Q-Learning agent receives a scalar reward r at each time step, defined as:

$$r = -(W_L.L + W_E.E) \tag{22}$$

Table 1. Symbol Definitions and Glossary

Symbol	Definition	Unit
α (alpha)	Learning rate for Q-Learning; controls the step size of value function updates	dimensionless
γ (gamma)	Discount factor; determines the importance of future rewards	dimensionless
ε (epsilon)	Exploration rate; probability of random action selection	dimensionless
ε_{decay}	Exploration decay rate; factor by which ε is multiplied each episode	dimensionless
L	Latency; time required to complete a task	seconds (s)
E	Energy consumption; total energy used for task execution	Joules (J)
W_L	Weight for latency in fitness function	dimensionless
W_E	Weight for energy in fitness function	dimensionless
F	Fitness value; objective function to be minimized	dimensionless
SO	Snake Optimizer	—
QL	Q-Learning	—
$QLSO$	Q-Learning optimized by Snake Optimizer	—
$Q(s,a)$	Q-value for state s and action a in the Q-table	dimensionless
X^*	Optimal hyperparameter triplet found by SO	—
$PopSize$	Population size in SO algorithm	dimensionless
$nIter$	Number of iterations in SO algorithm	dimensionless

The negative sign ensures that the minimization of the two metrics is equivalent to maximization of the reward, enabling standard RL algorithms designed for reward maximization to solve the minimization problem.

The Q-Learning update rule is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot [\, r + \gamma \cdot \max_{a'} Q(s',a') - Q(s,a) \,] \tag{23}$$

Where α is the learning rate, γ is the discount factor, and $Q(s,a)$ represents the learned value of taking action a in state s.

4.2. Snake Optimizer Algorithm

Meta-heuristic optimization algorithms are advanced techniques that are designed to solve complex optimization problems. They base their conceptual foundations on the natural phenomena, thus, allowing them to cover the vast solution spaces thoroughly. In turn, their natural plasticity and stability make them very applicable within a range of field disciplines. SO is a meta-heuristic algorithm of optimization based on the hunting strategy of snakes.

The SO algorithm maintains a population of candidate solutions, each representing a hyperparameter triplet ($\alpha, \gamma, \varepsilon_{decay}$). The algorithm alternates between two operational modes:

1. Exploration Phase: Candidates are diversified through random variations directed by inter-individual interactions, enabling the search to comprehensively sample the solution space.
2. Exploitation Phase: The search concentrates on the most successful solutions, with movement controlled by environmental forces (temperature (T) and food quantity (Q)) factors. These factors guide the transition from diversified exploration to local refinement around high-quality candidate solutions until the maximum number of iterations is reached.

The adaptive search strategy inherently maintains a balance between global exploration and local refinement, allowing the optimizer to escape low-quality local optima and converge toward global solutions.

4.3. Q-Learning Optimized by Snake Optimizer (QLSO) Framework

In dynamic and complicated IIoT computation offloading, the learning rate (α), discount factor (γ), and exploration decay (ϵ_{decay}) have a significant impact on the performance in terms of convergence and stability of the baseline QL algorithm. These hyperparameters are often poorly tuned manually or at least fixed a priori, which leads to poor learning performance. The proposed QLSO framework enhanced the baseline QL algorithm by an adaptive hyperparameter tuning processor by the SO algorithm.

The overall pipeline of the proposed QLSO framework is shown in Figure 2, where the SO is integrated with QL to automatically tune the hyperparameters. The pipeline consists of two stages: (1) offline hyperparameter optimization, and (2) online decision-making.

The offline phase starts with the initialization of the SO population, with each candidate being a triple of hyperparameters ($\alpha, \gamma, \varepsilon_{decay}$). For each candidate, the QL agent is trained for 250 episodes and evaluated over 100 episodes. The Snake Optimizer's exploration-exploitation balance is controlled by the fitness values until convergence to the optimal configuration $X = (\alpha, \gamma^*, {\varepsilon_{decay}}^*)$. This optimized configuration is then used to train the QL agent from scratch for 500 episodes prior to the online decision making phase. The online phase utilizes the trained agent X^* to make real-time offloading decisions in the IIoT environment, which can be either local execution, edge offloading, or cloud offloading, based on the learned Q-table. This separation between tuning, evaluation, and final training helps reduce leakage between HPO and final performance reporting.

In QLSO implementation, each candidate solution in the SO population represents a parameter triplet (α, γ, ϵ_{decay}) within predefined search ranges and their corresponding justifications specified in Table 2. The fitness of each candidate is evaluated through episodic training and testing of a QL agent in the IIoT environment. The fitness function is defined as:

$$F = W_L \cdot L + W_E \cdot E \tag{24}$$

In this fitness evaluation, L and E denote the average latency and average energy measured over the evaluation episodes, rather than instantaneous single-step values. Reduction in F values suggests that the selected hyperparameter settings result in lower latency and low energy use.

SO uses the adaptive search strategy which switches between the two modes of operations. During the exploration phase, candidate parameters are diversified through random variations, which are directed by inter-individual interactions, and thus allows the search to greatly sample solution space. At the exploitation stage, the search is more concentrated more about the most successful solutions, the movement is controlled by the environmental forces like temperature and food quantity. These inherently maintain a balance between global exploration and local refinement, allowing the optimizer to get out of low-quality local optima, and move towards high quality results.

Once the maximum number of iterations is hit, $X^* = (\alpha^*, \gamma^*, {\varepsilon_{decay}}^*)$ is found globally and selected. This optimized set of hyperparameters is then used to retrain QL agent from initialization for full deployment and performance testing in the target IIoT computation offloading environment.

The hyperparameter search bounds were selected to provide a sufficiently broad and stable search space for tuning the QL parameters in the considered IIoT offloading simulation.

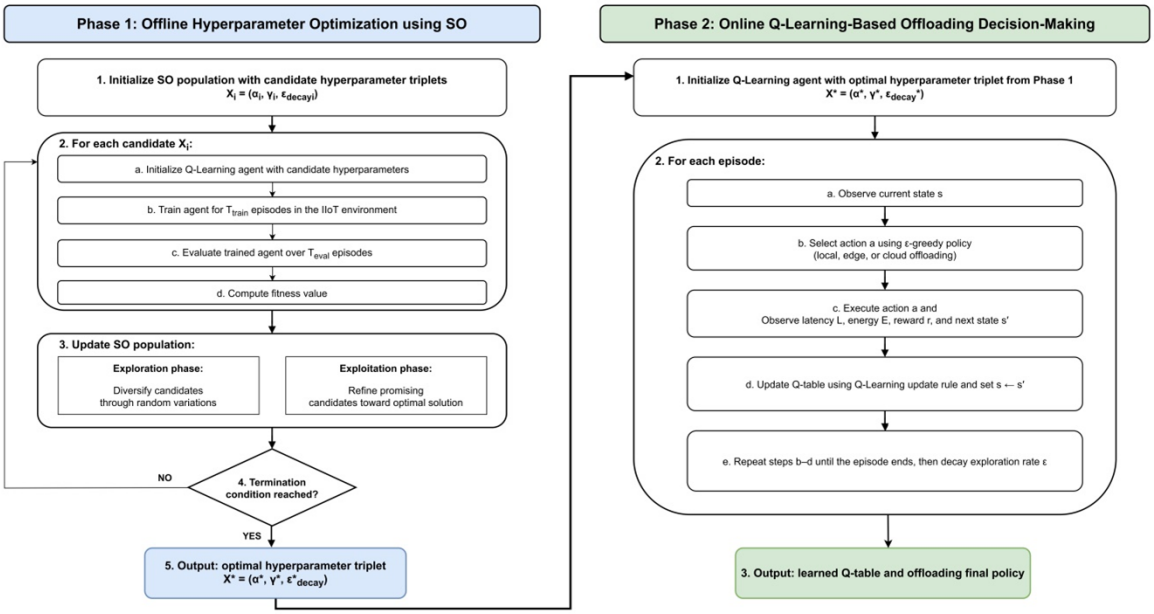


Figure 2. QLSO framework Pipeline. The flowchart shows that the system has two phases: (1) offline hyperparameter optimization using Snake Optimizer, and (2) online decision-making using Q-Learning with optimized hyperparameters. The SO trains and evaluates QL agents to assess hyperparameter candidates and identify the best configuration $X^* = (\alpha^*, \gamma^*, \varepsilon_{decay}{}^*)$. This optimized configuration is then used for deployment in the IIoT computation offloading environment

Table 2. Hyperparameter Search Bounds and Justifications

Parameter	Symbol	Search Range	Justification
Learning rate	α	[0.10, 0.50]	Values below 0.10 result in unacceptably slow convergence within the 500-episode training window, while values exceeding 0.50 induce Q-value instability due to excessive updates.
Discount factor	γ	[0.50, 0.99]	Lower bound of 0.50 ensures that future rewards are still considered during learning; upper bound of 0.99 provides near-infinite horizon planning, which may be unnecessarily complex for task time scales.
Exploration decay	ε_{decay}	[0.90, 0.995]	Upper bound of 0.995 ensures sufficient exploration persists through training episodes; lower bound of 0.90 ensures adequate exploration-exploitation transition for smaller state spaces.

5. SIMULATION RESULTS

To evaluate the performance of the proposed algorithm, simulation experiments were conducted. All experiments were performed on a PC equipped with an Intel Core ™ i7- 10750 H CPU, 32 GB of RAM, running Windows 10 Pro (64-bit), using Python 3.6.13 in an Anaconda environment.

5.1. Simulation Setup

The simulation framework was designed to replicate the hierarchical layers of the IIoT architecture, comprising ten IIoT devices, a base station with an edge server and a cloud server. Table 3 displays the parameters of the simulation based on values that are widely used in the previous studies [9],[50][51]. The QLSO algorithm was compared with three benchmark algorithms: QL (off-policy), SARSA (on-policy) and DQN (off-policy).

5.2. Performance Metrics

The following metrics were used to compare and evaluate the QLSO algorithm performance:

a. Average Latency: The average completion time of a task of an IIoT device, performed locally or offloaded [2].

b. Average Energy Consumption: The average energy that an IIoT device uses in order to execute its task, whether locally or by offloading [52].

c. Cost: It is a weighted sum of latency and energy usage [31].
d. Latency-Optimal Decision Rate: The percentage of offloading decisions where the algorithm is able to pick the location of execution that minimizes task completion latency amongst all the available choices [53].
e. Task Offloading Distribution: This is the distribution pattern of computational tasks among the accessible locations of computation (local devices, edge servers, cloud) characterized by the percentage allocation to each computing layer [54].
f. Convergence: this is the ability of an algorithm to have the optimal solutions as a result of its interaction with its environment. It is defined by two main features convergence speed which is defined as the speed at which the algorithm learns the effective policy and convergence stability which is defined as the ability of the algorithm to perform consistently once the optimal solution is obtained. It is assumed that an effective RL algorithm will converge quickly and provide stable decision-making in dynamic IIoT environments [55].
g. 95th Percentile Latency: A measure that shows the amount of time it takes to complete 95% of tasks [56][57].

5.3. Performance Evaluation

This subsection shows the performance analysis of the proposed solution through simulations. The main aim is to determine the effectiveness of the proposed QLSO algorithm in relation to seven key performance metrics and how the algorithm compares with three benchmark algorithms under various experimental settings. The obtained results are compared in order to demonstrate the excellence of the suggested QLSO algorithm in IIoT environments.

Tabel 3. Simulation parameters

Parameter	Value	Parameter	Value
Input data size of a task (DS_d)	[1, 30] Mbit	Task computational intensity (CI_d)	[700, 1300] cycles/bit
Switched capacitance of IIoT device (K)	1×10^{-28} J/Hz²	Computational capacity of IIoT device (F_{IIoT})	[1, 13] GHz
Computational capacity of edge server (F_{Edge})	[9, 21] GHz	Computational capacity of cloud server (F_{Cloud})	[20, 32] GHz
Transmission rate between IIoT and base station (R^{b}_{IIoT})	4.7 Mbps	Transmission rate between base station and cloud server (R^{t}_{Edge})	15 Mbps
Transmission power (P^{t}_{IIoT}) and Standby power (P^{s}_{IIoT})	0.6 W, 0.1 W	Weighted sum of latency (W_L) and energy consumption (W_E)	$W_L = 0.5$, $W_E = 0.5$
Poisson arrival rate (λ)	5	Number of independent seeds	42
RL algorithm training episodes	500	Snake Optimizer training episodes	250
Snake Optimizer population size	20	Number of Snake Optimizer iterations	20

5.3.1. Convergence Analysis Under Varying Task Complexity

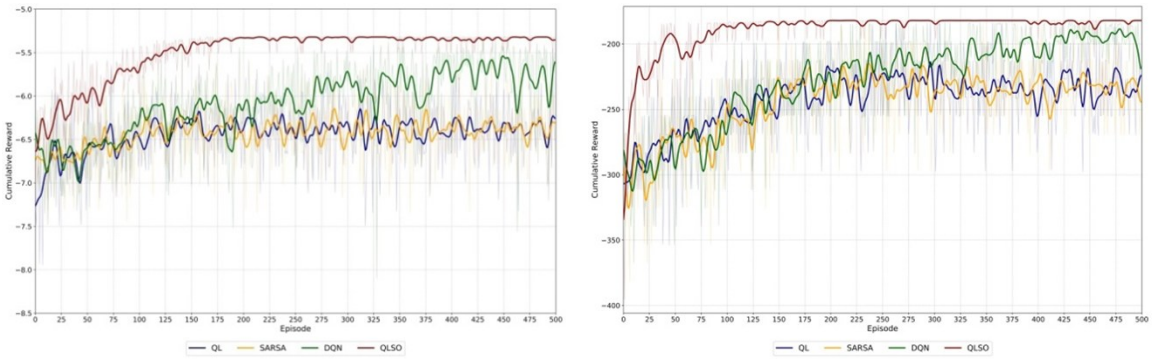
The convergence behaviour of the QLSO algorithm was compared to benchmark algorithms in two test cases: Low Complexity Tasks (LCT) with small task size and low computational intensity and High Complexity Tasks (HCT) with large task size and high computational intensity. Convergence was measured by monitoring the change in cumulative reward as in Figure 3(a) for LCT and Figure 3(b) for HCT, with respect to the number of training episodes. The empirical results show the proposed solution reached faster convergence and produced more stable reward curves as compared to the benchmark algorithms, and it takes less training episodes to be stabilized. Besides, the results testify to the effectiveness of the SO algorithm in optimizing the hyperparameters of the QL algorithm, thus improving its effectiveness in IIoT task-offloading environments.

To further validate the convergence dynamics and aid in supporting the comparative findings, another statistical study was conducted to determine the variability of the performance measured in all of the algorithms evaluated. The Standard Deviation (StDev) analysis, which is calculated in Minitab® 20.3 (64-bit), confirms increased convergence speed and stability of the proposed QLSO algorithm in both task complexity scenarios as illustrated in Table 4. In the LCT case, QLSO achieves the lowest value of (StDev= 0.3207), which is lower than the corresponding values of 0.3651 in QL, 0.3474 in SARSA, and 0.4801 in DQN, which shows that the proposed approach maintains a higher performance level even with small tasks with small computational requirements. The Standard Deviation of the QLSO was remarkably low in HCT case (19.06) and significantly better in comparison to QL (StDev= 31.09), SARSA (StDev= 33.32), and DQN (StDev= 38.14). This shows that the suggested algorithm is quite consistent even when it comes to dealing with computationally challenging

tasks that require large amount of data. As illustrated in Figure 4 to Figure 8, the comparative evaluation across varying task sizes, computational intensities, and server capacities demonstrates that QLSO consistently achieves superior performance across latency, energy, and cost metrics. The convergence curves in Figure 3 show that QLSO reaches stable performance earlier than competing algorithms in both LCT and HCT scenarios. Furthermore, the convergence analysis shows that QLSO achieved stability faster than competing algorithms in both scenarios. Based on the stability patterns observed in both the LCT and HCT reward curves, the proposed algorithm demonstrates convergence after a reasonable number of episodes in both scenarios, whereas the other algorithms either reached stability much later or failed to converge within the training period.

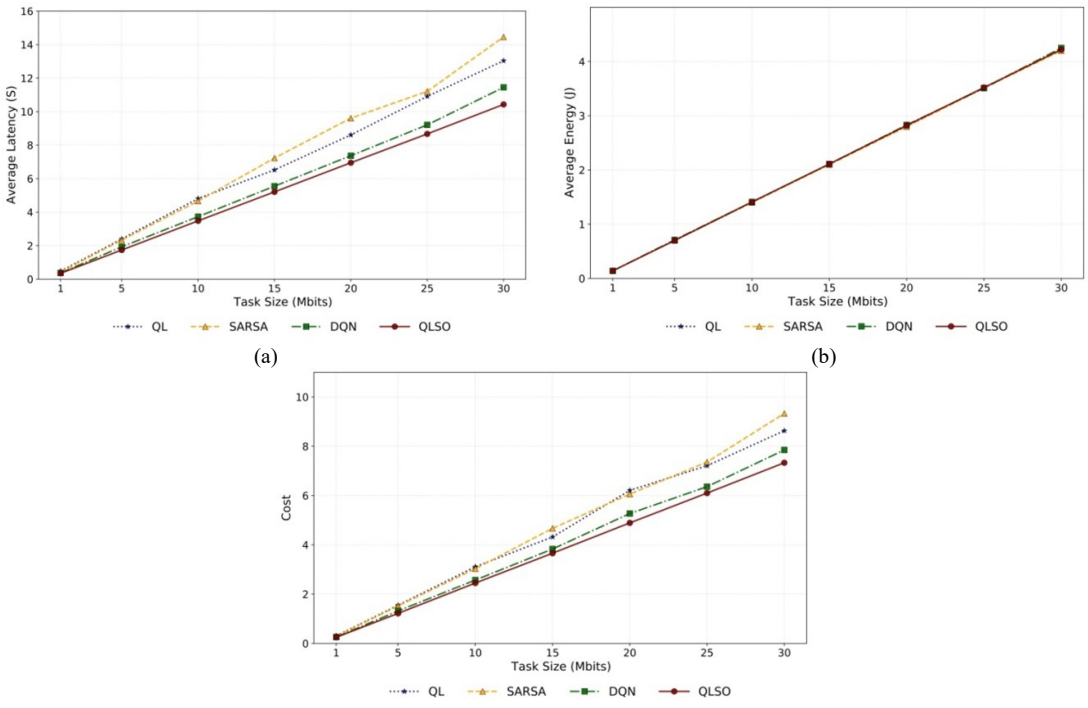
Table 4. Standard deviation comparison of four algorithms

Algorithm	StDev under LCT Scenario	StDev under HCT Scenario
QL	0.3651	31.09
SARSA	0.3474	33.32
DQN	0.4801	38.14
QLSO	0.3207	19.06



(a) (b)

Figure 3. The convergence behaviour of (a) low-complexity tasks and (b) high-complexity tasks, where the curves have been smoothed with a Gaussian filter with $\sigma = 2.0$



(a) (b)

(c)

Figure 4. Comparison of Algorithm choosing based on different Task Sizes. Figures depict (a) Average Latency, (b) Average Energy and (c) Cost in different task sizes. QLSO outperforms all other algorithms

This faster convergence translates into multiple critical benefits for IIoT task offloading systems: (1) reduced training time and computational costs, enabling devices that have limited computing resources; (2) lower energy consumption during the learning phase, which is particularly important for resource-constrained devices; (3) faster deployment in IIoT environments, allowing the system to begin making optimal offloading decisions sooner; and (4) the capability to perform training with fewer iterations, significantly improving the applicability of the proposed approach in dynamic IIoT environments. This speed of convergence coupled with the stability of low variance found in for both LCT and HCT cases is significantly important for practical real industrial-IIoT task offloading system where quick learning and dependable decision making are needed to ensure system performance and QoS requirements under different workloads.

5.3.2. Comparative Evaluation Across Latency, Energy, and Cost Metrics

To provide a further evaluation of the suggested QLSO algorithm, the proposed algorithm was compared to three known algorithms, namely the Q-Learning, SARSA, and DQN, in the three aspects of latency, energy consumption and overall cost. It was evaluated in five different scenarios namely; (1) varying task sizes as in Figure 4; (2) varying task computational intensities as in Figure 5; (3) different computational capacities of IIoT devices as in Figure 6; (4) different computational capacities of the edge server as in Figure 7; and (5) different computational capacities of the cloud server as in Figure 8. The empirical findings in all the scenarios show that the QLSO algorithm is always the best in reducing these key metrics mentioned above when compared with the benchmark algorithms. This better performance result from the effectiveness of the QL algorithm enhanced by SO algorithm, which enables the proposed solution to select the optimal offloading decision under varying system conditions and task characteristics.

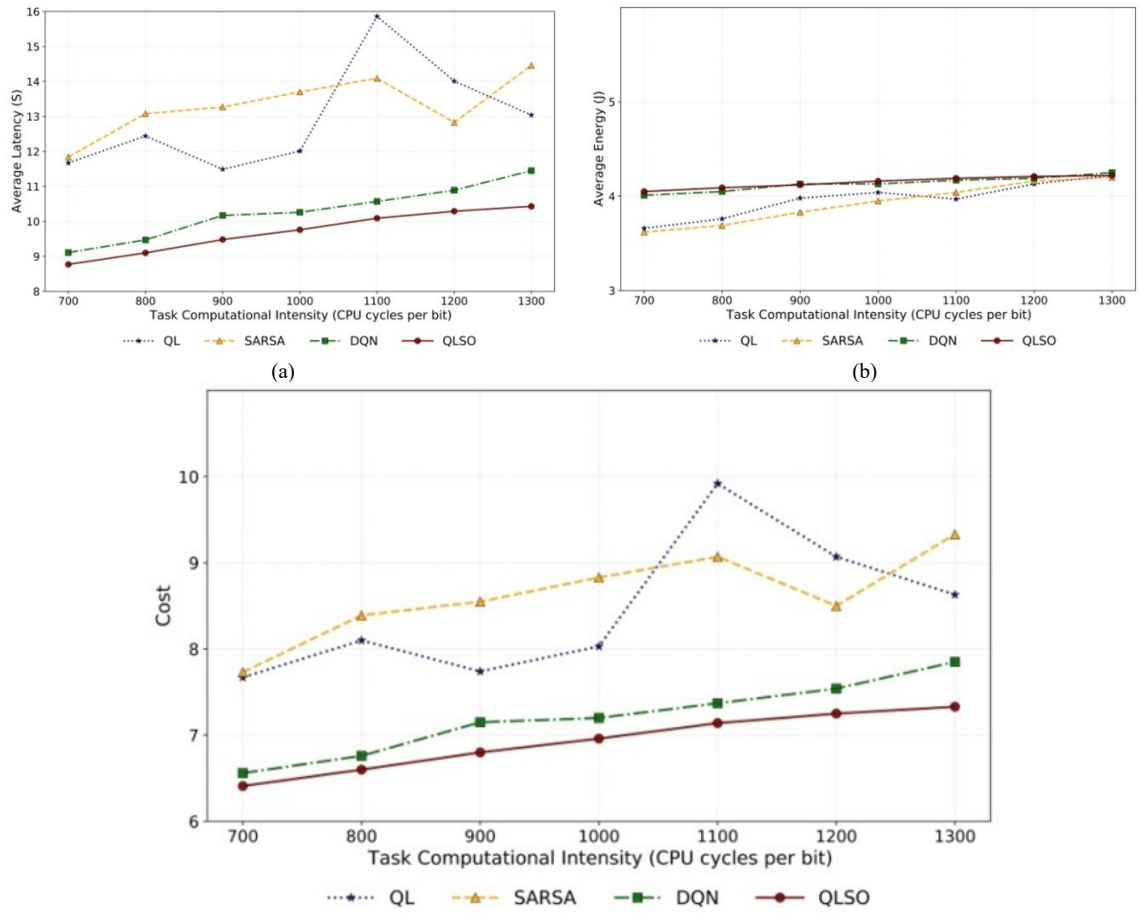


Figure 5. Comparison between Algorithms under Different Task Computational Intensities. Figures show (a) Average Latency, (b) Average Energy, and (c) Cost across varying intensity levels. QLSO sustains its performance benefits across all considered levels

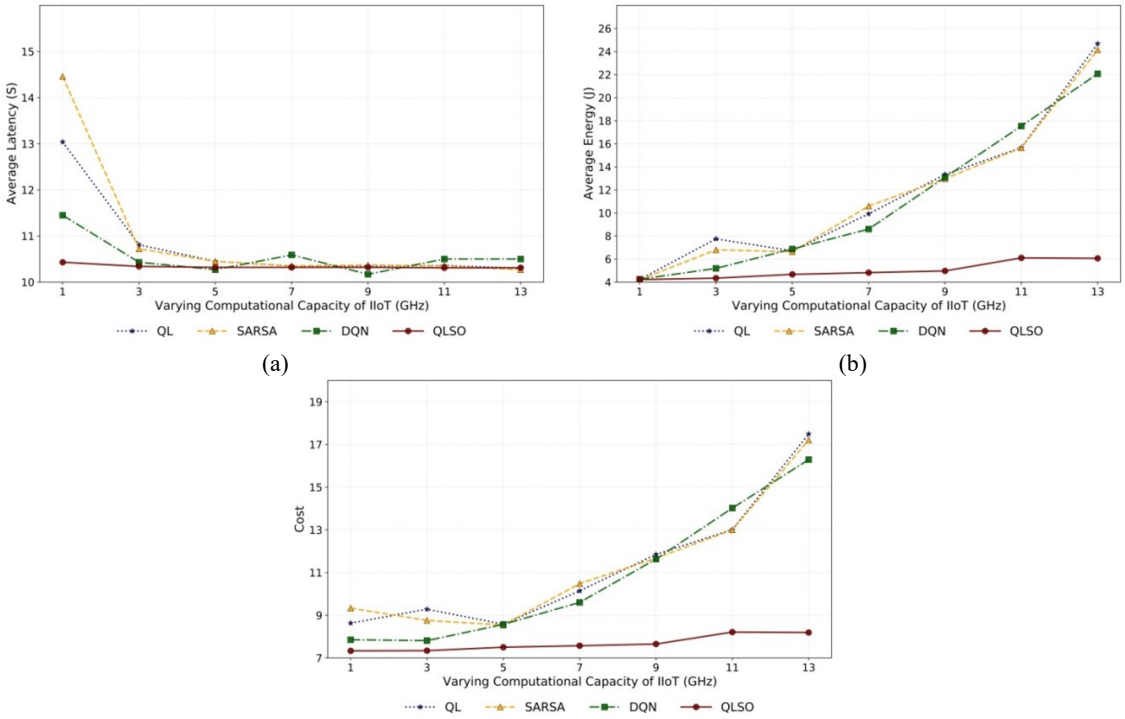


Figure 6. Comparison between Algorithms under Different IIoT Device Computational Capacities. Figures show (a) Average Latency, (b) Average Energy, and (c) Cost across varying device capacities. QLSO maintains stable performance under different hardware capabilities

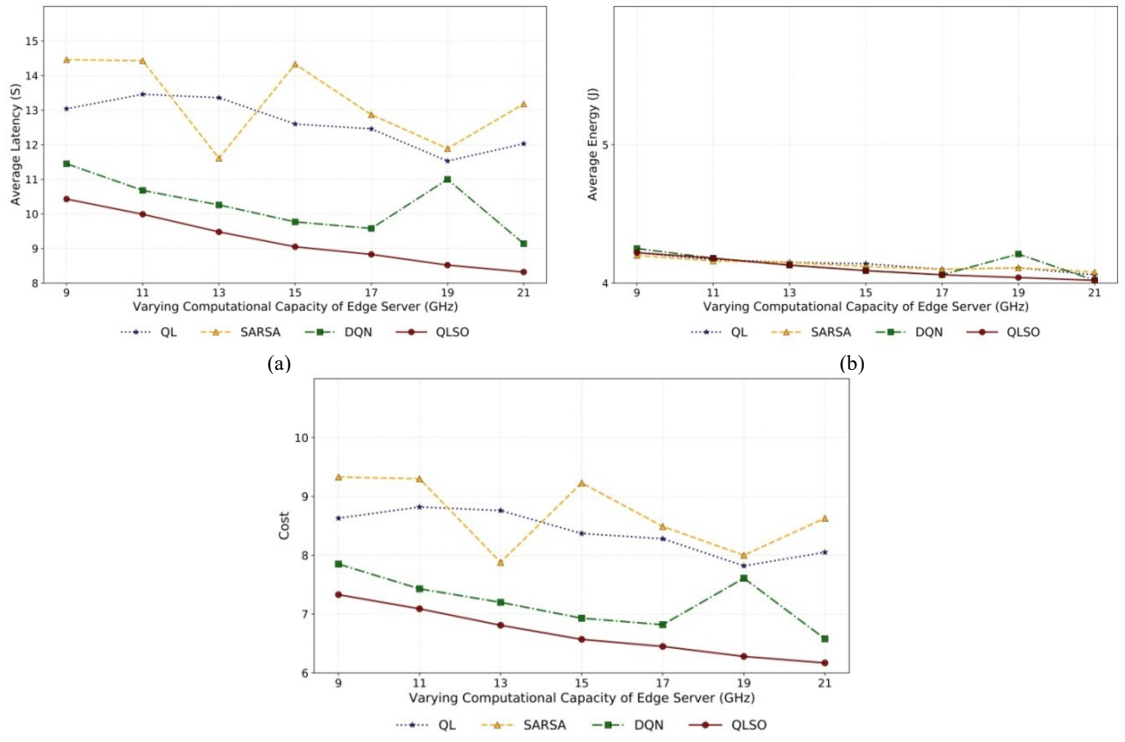


Figure 7. Comparison between Algorithms under Different Edge Server Computational Capacities. Figures show (a) Average Latency, (b) Average Energy, and (c) Cost across varying edge capacities. The performance hierarchy remains consistent, with QLSO achieving the best results

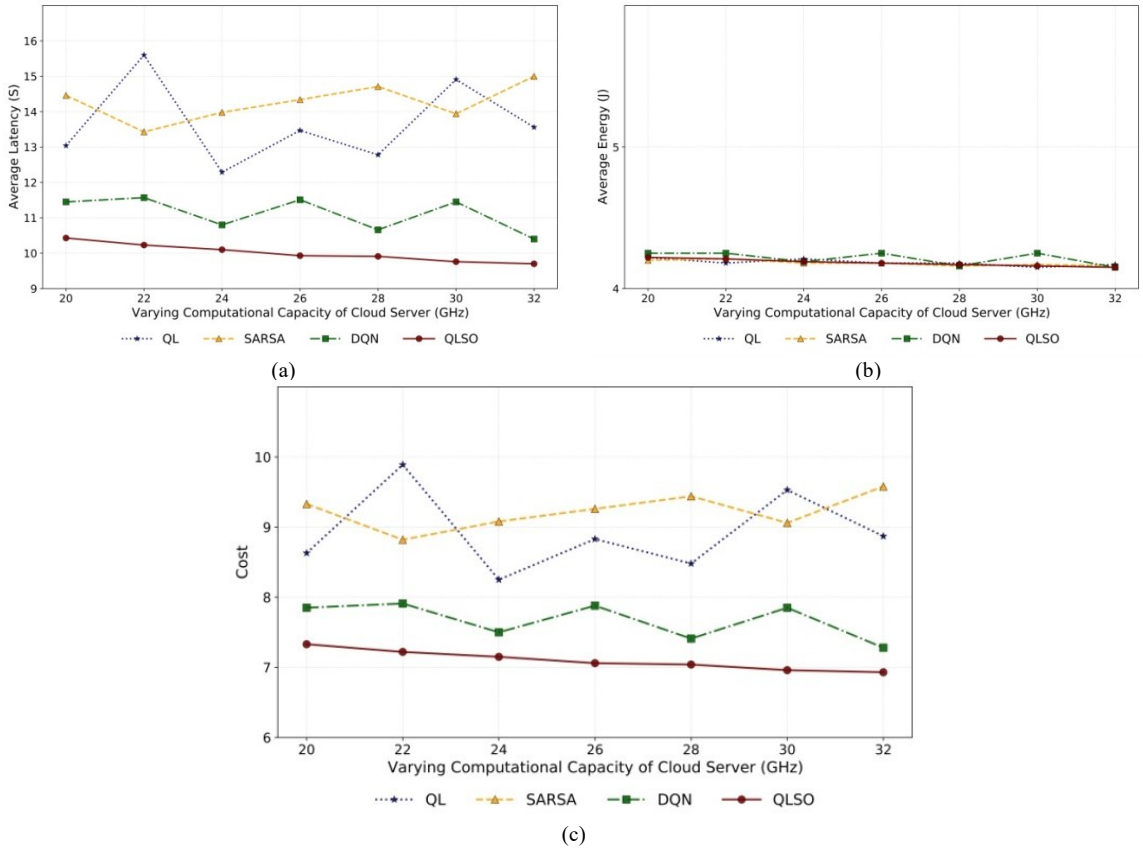


Figure 8. Comparison between Algorithms under Different Cloud Server Computational Capacities. Figures show (a) Average Latency, (b) Average Energy, and (c) Cost across varying cloud capacities QLSO maintains superior performance across all cloud configurations

5.3.3. P95 Performance under Varying System Conditions

A comparison of the 95th percentile latency (P95) between the proposed solution and benchmark algorithms in different system conditions is given in Figure 9. The 95th percentile latency represents the maximum latency of tasks by 95 percent of procedures and represents performance under the worst conditions. This indicator is important in the context of IIoT, where tasks are often of urgency, and predictability of the deadline is required to maintain a consistent process. The findings prove that the proposed solution achieves lower values of P95 latency than the benchmark algorithms in varied scenarios, providing quicker completion of most tasks despite unfavorable conditions. This level of tail-latency performance is necessary to achieve service-quality goals, since it ensures that only a small portion (5%) of tasks take longer than the P95 threshold. The superior P95 performance of the proposed solution can be explained by the enhanced RL algorithm's ability to learn and adapt to system dynamics, enabling it to make offloading decisions that minimize latency variation and avoid performance decline during peak load conditions or resource contention scenarios.

5.3.4. Comparison of Optimal Latency Across Benchmark Algorithms

The optimal latency rate, which measures the percentage of cases where each algorithm makes the optimal offloading decision that results in the minimum possible latency is presented in Figure 10. This metric directly evaluates the decision-making quality of each algorithm by comparing its choices against the optimal decision in each scenario.

The results show that algorithm DQN generally performs better than the other benchmark algorithms, QL and SARSA, in terms of raw latency and energy metrics. However, under the adopted discrete-state simulation setting, the proposed QLSO algorithm achieves the highest optimal latency rate in most evaluated cases. Therefore, the comparison is interpreted in terms of the considered metrics and scenarios, rather than as a general claim of superiority over DRL methods. This performance supports the effectiveness of the enhanced QL algorithm, which has been improved by integrating SO algorithm, which leads to enabling the system to improve its decision-making policy.

Anyhow, as illustrated in Figure 10(c), when the computational capacity of IIoT devices increases to 5 GHz or higher, a noticeable decrease in the Optimal Latency Rate is observed for all algorithms. This outcome can be explained by the reward function adopted by all algorithms. This function is designed to minimize the total cost, which is defined as a weighted sum of latency and energy consumption. As the CPU capacity increases, the local execution time of tasks decreases, which makes local execution appear as the optimal decision in isolation. However, the higher CPU frequency also results in increased energy consumption, thereby raising the overall execution cost. Consequently, the algorithms adapt by offloading tasks to edge or cloud servers to reduce the energy cost, even though this introduces a moderate increase in latency. This trade-off results in a lower total cost but it simultaneously causes a reduction in the Optimal Latency Rate, since the offloaded decisions do not always correspond to the lowest-latency option. Overall, the high Optimal Latency Rate, which the proposed algorithm managed to reach, means that it successfully find the most effective offloading strategy in various system states and task features. This result suggests that the proposed RL-based framework can adapt to the evaluated system states and task features.

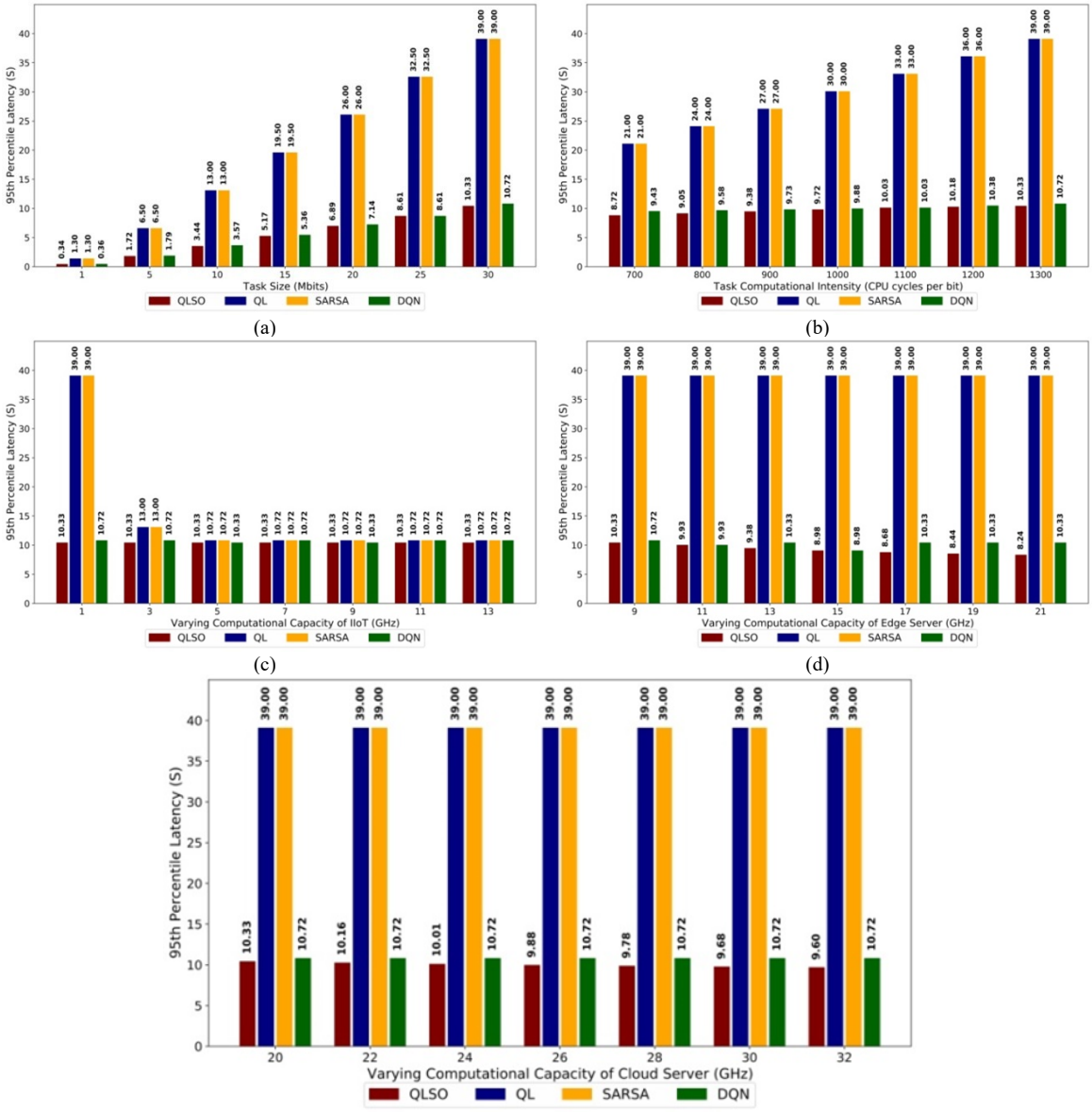


Figure 9. Comparison of algorithms for 95th percentile latency over (a) input task size, (b) input computation intensity, (c) IIoT device capacity, (d) edge server capacity, and (e) cloud server capacity. QLSO achieves the lowest tail latency across all graphs, indicating strong worst-case guarantees

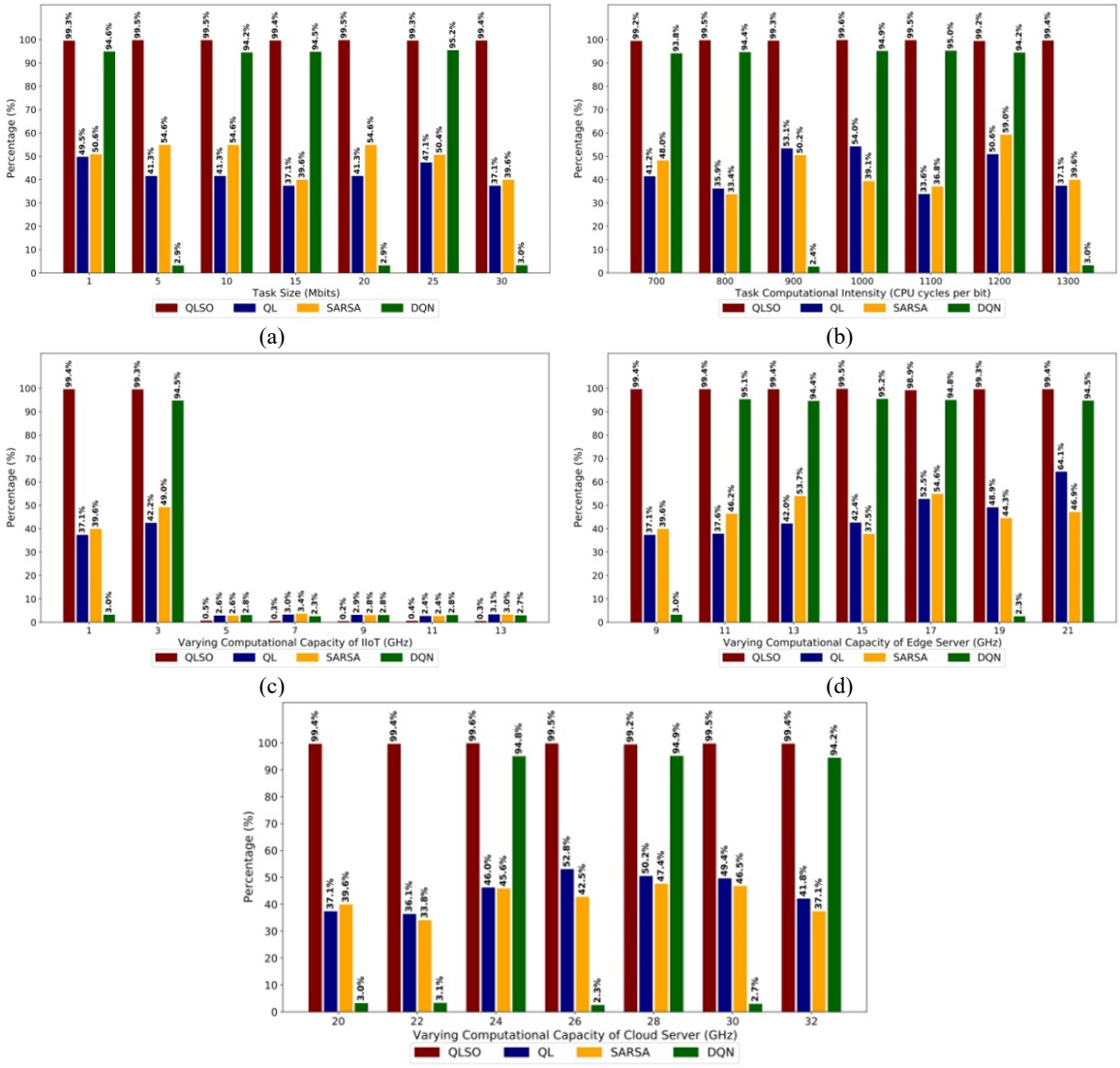


Figure 10. Comparison of algorithms for Optimal Latency Rate. The results compare the Optimal Latency Rate across different system conditions. QLSO achieves the highest rate in most cases, indicating its ability to select latency-efficient offloading decisions

5.3.5. Task Offloading Performance Across Execution Locations

As illustrated in Figure 11, the task offloading distribution by showing how the computational tasks are distributed among three available levels of execution, i.e., local IIoT devices, edge servers, and cloud servers and is expressed as a percentage share of the computational tasks allocated to each level. The findings show that there exist strong differences regarding how each algorithm allocates tasks to these execution levels. The QLSO algorithm displays more balanced and dynamic distribution as compared to benchmark algorithms that exhibit biases in allocating tasks to a given execution tier regardless of the nature of the tasks or the current state of the system. This intelligent distribution indicates that the proposed solution effectively computes the properties of the task, including their size and the computational intensity, coupled with the available resources computational functionality to determine the optimal place of execution of each task. This behavior is also linked to the optimized hyperparameter configuration, which guides the learned policy in balancing local, edge, and cloud decisions under different task and resource conditions. The proposed solution promotes the efficient use of the resources in the whole system by reducing the dependency on a single location of execution without compromising the optimal performance of the system. Such effective distribution scheme among the three layers of execution explains the high performance in the latency, energy consumption, and overall cost achieved by proposed solution as shown in the Figure 11. This adaptive distribution supports the capabilities

of the improved RL framework to develop context-aware offloading decisions to dynamically react to different task features and system conditions.

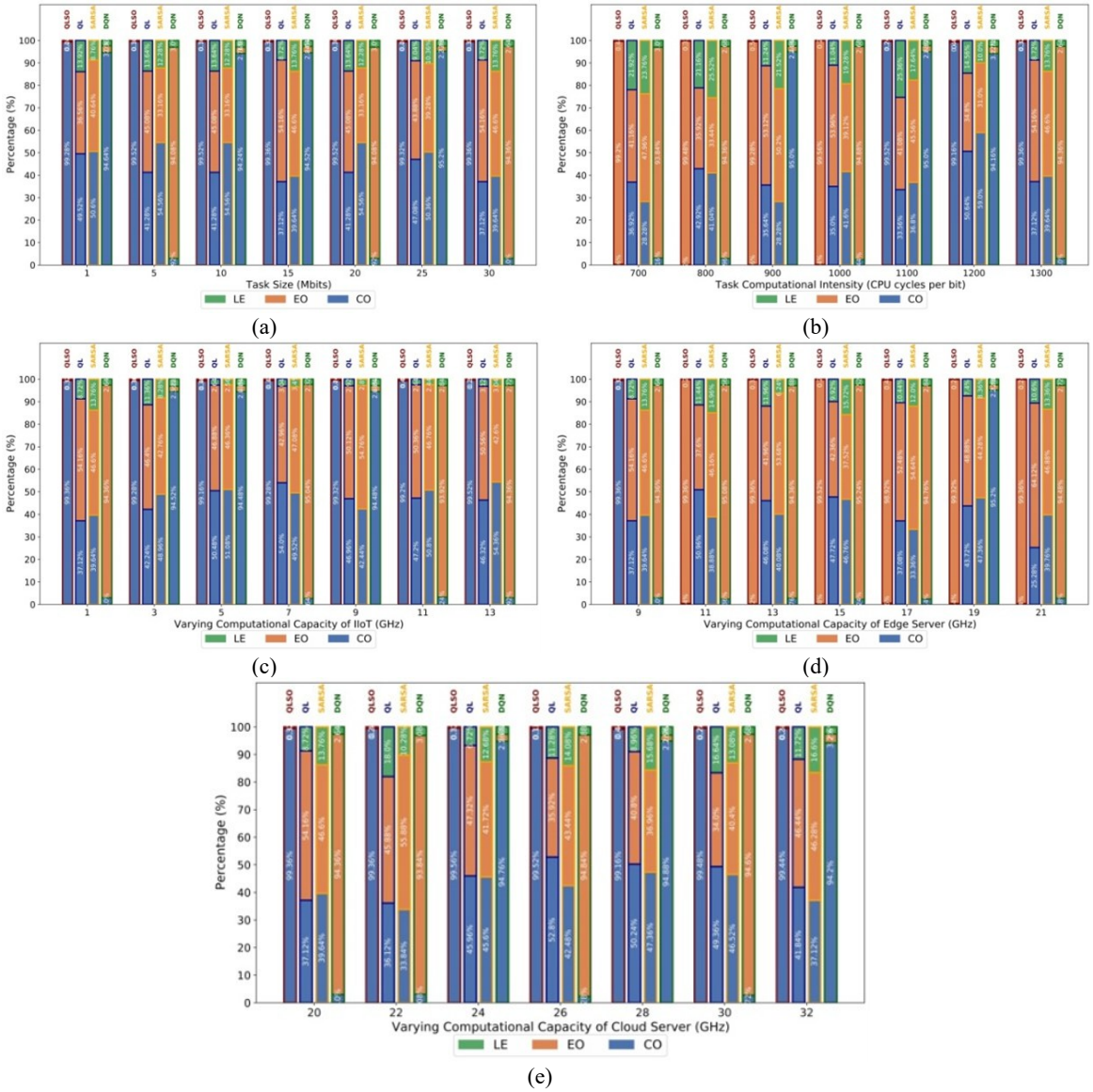


Figure 11. Comparison of algorithms for Task Offloading Distribution. The results illustrate how each algorithm distributes tasks among local execution, edge offloading, and cloud offloading across different system conditions

5.3.6. Ablation Study: Impact of Snake Optimizer Enhancement

To isolate the contribution of SO-based HPO relative to QL, an ablation study compared three configurations: (1) QL-Fixed with manually selected hyperparameters (alpha = 0.1, gamma = 0.95, exploration decay = 0.995); (2) QL+SOA-Standard using the vanilla SO; and (3) QL+SOA-Modified using the modified exploration-exploitation variant. The comparison uses identical search bounds, training episodes, and greedy validation episodes. The hyperparameter search bounds are set based on theoretical considerations and empirical observations for the domain of IIoT offloading. Specifically, the learning rate (α) is searched within the range [0.05, 0.5], the discount factor (γ) is searched within the range [0.80, 0.99], and the exploration decay (ε_{decay}) is searched within the range [0.90, 0.995]. These ranges are defined consistently across all experimental configurations to ensure reproducibility and to avoid scenario-specific tuning. To make the experimental configuration explicit before reporting the ablation results, the main simulation settings used in this study are summarized in Table 5. The optimal hyperparameter configurations obtained across five independent random seeds for both the standard and modified SO variants are presented in Table 6.

The placement of the optimized values in Table 6 is supported by the following validation procedure. A comprehensive multi-stage validation procedure was used to evaluate the robustness and generalizability of the optimized hyperparameters. To reduce the effect of the stochastic initialization, each experimental setup was tested with five different random seeds (42, 7, 13, 21, and 99), and the results were statistically reliable. The results of these runs were reported as mean and standard deviation, and 95% confidence intervals were estimated by using the Student's t-distribution. Comprehensive ablation studies were performed by comparing a baseline QL-Fixed model with two optimized variants: QL+SOA-Standard and QL+SOA- Modified, where QL-Fixed uses manually selected hyperparameters and the SOA variants use optimized hyperparameters. Furthermore, the proposed QLSO algorithm was compared with three popular reinforcement learning algorithms: Q-Learning, SARSA, and DQN, with the same experimental setup. The optimum hyperparameter triplet $X^* = (\alpha^*, \gamma^*, \varepsilon_{decay}{}^*)$ was chosen from the final population when the optimization process was completed. This optimal configuration was then used to re-initialize and re-train the QL agent from scratch, and its effectiveness was validated by empirically consistent performance gains across the benchmark comparisons in the target IIoT computation offloading environment.

The ablation results in Table 7 show that both SO variants improve over fixed-hyperparameter QL. Standard SOA achieves a 9.1% fitness improvement over QL-Fixed, while the Modified SOA achieves a 6.5% improvement. Because Standard SOA slightly outperforms the modified variant in this experimental regime, the manuscript does not claim that the modified mechanism is categorically superior. Instead, the supported conclusion is that SO-based HPO itself is beneficial, and additional enhancement mechanisms require further validation.

Table 5. Simulation settings used in the ablation and scenario analyses

Parameter	Value	Parameter	Value
Input data size of a task (DS_d): Small/Big Scenario	[20, 80] × 10³/ [100, 400] × 10³ bits	Task computational intensity (CI_d)	Random uniform [500, 2000] cycles/bit
Switched capacitance of IIoT device (K)	1 x 10^{-28} J/Hz²	Computational capacity of IIoT device (F_{IIoT})	Random uniform [1, 3] GHz
Computational capacity of edge server (F_{Edge})	Random uniform [9, 15] GHz	Computational capacity of cloud server (F_{Cloud})	Random uniform [20, 40] GHz
Transmission rate between IIoT and base station: Small/Big Scenario	[3, 15]/ [5, 30] Mbps	Transmission rate between base station and cloud server	15 Mbps
Transmission power (P^t_{IIoT}) and Standby power (P^s_{IIoT})	0.6 W, 0.1 W	Weighted sum of latency (W_L) and energy consumption (W_E)	W_L = 0.7, W_E = 0.3
Poisson arrival rate (λ)	5	Number of independent seeds	[42, 7, 13, 21, 99]
RL algorithm training episodes	500	Snake Optimizer training episodes	300
Snake Optimizer population size	10	Number of Snake Optimizer iterations	10

Table 6. Ablation Study: Optimal Hyperparameters Across Five Random Seeds

Seed	Standard SO ($\alpha, \gamma, \varepsilon_{decay}$)	Time (s)	Modified SO ($\alpha, \gamma, \varepsilon_{decay}$)	Time (s)
42	(0.4996, 0.8000, 0.9351)	147.6	(0.5000, 0.8228, 0.9519)	146.8
7	(0.5000, 0.8000, 0.9950)	145.5	(0.5000, 0.9702, 0.9950)	146.6
13	(0.0500, 0.8000, 0.9950)	148.3	(0.4835, 0.9053, 0.9920)	149.3
21	(0.5000, 0.8000, 0.9000)	148.6	(0.0984, 0.8006, 0.9928)	149.9
99	(0.0500, 0.8000, 0.9950)	148.9	(0.3956, 0.8684, 0.9619)	148.5

Table 7. Ablation Study Results (mean ± std across 5 seeds)

Method	Avg Latency (s)	Lat Std	Avg Energy (J)	En Std	Fitness
QL-Fixed	0.0153	0.0005	6.85×10^{-3}	5.68×10^{-4}	3.124
QL+SOA-Standard	0.0145	0.0003	6.07×10^{-3}	2.10×10^{-4}	2.839
QL+SOA-Modified	0.0148	0.0003	6.29×10^{-3}	3.56×10^{-4}	2.921

The following clarification is added before discussing the detailed runtime results to make the source of the offline computational overhead explicit. One of the major concerns for practical implementation of the SO is the computational burden. The SO optimization process incurs computational cost in terms of fitness evaluations and wall-clock runtime. In terms of fitness evaluations, PopSize × nIter + initial population = 10 × 10 + 10 = 110 evaluations (Evals) per SO run. In terms of wall-clock time, the SO process required 146.5 ± 2.0 seconds in the small scenario and 477.8 ± 1.5 seconds in the large scenario.

The fitness evaluation count of 110 per optimization run is the number of QL training-test cycles that the SO runs per optimization run. This overhead is only paid during the offline hyperparameter tuning stage and does not impact online decision-making latency. After the optimal hyperparameter triplet $X^* = (\alpha^*, \gamma^*, \varepsilon_{decay}{}^*)$ is found, the computational complexity of the QL agent is constant per step and does not depend on the SO overhead.

The computational complexity overhead for both the Standard and Modified SO was then recorded. Here, the Standard SO needed 147.8±1.2 seconds to run 110 fitness evaluations; while the Modified SO needed 148.2±1.3 seconds for the same number of function evaluations. This insignificant variation in runtime indicates that the modified method lacks much additional computational cost. Significantly, the computational complexity overhead occurs only at the time of the offline hyperparameters search. Following the hyperparameters search, the online process relies on a constant amount of computation per step which ensures the real-time applicability of the approach within an IIoT context. The runtime showed approximately linear time growth as the number of clients approached 30, such that 69.6 seconds are needed for 5 clients, while 118.6 seconds are needed for 30 clients. Additionally, the ablation test (Table 8) indicated that the Standard SOA had a final fitness of 9.1% than fixed hyperparameters, demonstrating that the added computational overhead is well compensated with the performance increases.

Table 8. Statistical Summary of Ablation Study Performance Improvements

Comparison	Latency Reduction	Energy Reduction	Fitness Improvement
SOA-Standard vs QL-Fixed	5.2%	11.4%	9.1%
SOA-Modified vs QL-Fixed	3.3%	8.2%	6.5%
SOA-Standard vs SOA-Modified	+2.0%	+3.5%	+2.8%

Before moving to the scenario-specific analyses, the training, evaluation, and statistical reporting can be stated as follows. The training of the QL agent is structured episodically, where the number of training episodes is $T_{train} = 500$ for final agent evaluation. During training, action selection follows an ε-greedy policy with decaying ε, using the decay rule $\varepsilon \leftarrow max(\varepsilon_{min}, \varepsilon \times \varepsilon_{decay})$. In the evaluation phase, the number of evaluation episodes is $T_{eval} = 100$ for performance metrics, and action selection follows a greedy policy with $\varepsilon = 0$. The convergence criterion uses a rolling window size of 50 episodes and a convergence tolerance of 1% change in mean cumulative reward. If $|\bar{R}_t - \bar{R}_t - 50|/|\bar{R}_t - 50| < 0.01$, training is terminated early; that is, if the rolling-window mean cumulative reward changes by less than the 1% convergence tolerance, training stops early.

The convergence criterion can be used for adaptive termination, which can help to avoid unnecessary training when the agent has converged. This method is efficient in computation and effective in learning, so that it can be trained enough without too many iterations.

The performance of the proposed algorithm was evaluated through simulation experiments with five different random seeds (42, 7, 13, 21, 99). The results are presented as mean ± SD, and figures report 95% confidence intervals (CIs) calculated using Student's t-distribution where appropriate. Because the number of independent seeds is limited, claims of superiority are interpreted cautiously and are limited to cases where the direction of the effect is consistent with the reported mean values and ablation results. The ablation results are therefore interpreted cautiously and are not generalized beyond the considered experimental setting. Specifically, we report (1) the mean and standard deviation for each performance metric over five independent random seeds, (2) 95% confidence intervals for comparative figures, estimated using Student's t-distribution with n−1 degrees of freedom, (3) five independent runs, each evaluated over multiple evaluation episodes, per experimental configuration, and (4) relative performance differences through the ablation study, which compares QL-Fixed, QL+SOA-Standard, and QL+SOA-Modified using the same evaluation protocol. To ensure reproducibility, all random seeds, hyperparameter search bounds, and convergence criteria are clearly documented in the experimental configuration tables.

5.3.7. Small Scenario Performance Analysis

In the Small Scenario cases with lower system scale and less complex task requirements, the convergence behavior and performance values of all the tested algorithms display unique features as shown in Table 9. As shown in Table 10, Small Scenario, the DQN has the lowest average latency and energy consumption, while the QL has the highest latency optimality decision rate. The QLSO algorithm is comparable with the highest initial exploration diversity but has a slightly higher latency performance, which is believed due to the processing optimization overhead in the small scaled regime.

Table 9. Small Scenario: Optimal Hyperparameters Across Five Random Seeds

Seed	Modified SO ($\alpha, \gamma, \varepsilon_{decay}$)	Time (s)	Evals
42	(0.4246, 0.8403, 0.9173)	149.7	110
7	(0.4682, 0.8524, 0.9571)	146.9	110
13	(0.4432, 0.8017, 0.9709)	143.6	110
21	(0.5000, 0.8713, 0.9950)	145.8	110
99	(0.4616, 0.8793, 0.9648)	146.4	110

Table 10. Small Scenario: Comparative Performance Summary (mean ± 95% CI, 5 seeds)

Agent	Lat mean (s)	Lat std	En mean (J)	En std	OptRate (%)	P95 Lat (s)
Q-Learning	0.0145	0.0002	6.12×10^{-3}	2.67×10^{-4}	45.0	0.0356
SARSA	0.0148	0.0003	6.41×10^{-3}	2.59×10^{-4}	44.9	0.0360
DQN	0.0144	0.0001	5.94×10^{-3}	6.11×10^{-5}	44.4	0.0355
QLSO	0.0153	0.0002	6.79×10^{-3}	2.45×10^{-4}	44.4	0.0361

5.3.8. Big Scenario Performance Analysis

In the Big Scenario setup where the tasks are more complex and the scale is larger, the QLSO scheduler achieves the highest quality of decisions at a latency-optimal percentage of 49.0%, better than all other algorithms reported as benchmark algorithms. The best hyperparameters found for QLSO in the Big Scenario using 5 random seeds are listed in Table 11. From Table 12 QLSO obtains the best latency optimal decision rate (49.0%) for the Big Scenario out of all algorithms. Although DQN produces a slightly smaller average latency (0.0488s) and total energy cost (1.65×10^{-2} J), the higher decision quality achieved by QLSO, which is 1.0% higher than the next best algorithm, supports the effectiveness of SO-based tuning in the considered setting.

Table 11. Big Scenario: Optimal Hyperparameters Across Five Random Seeds

Seed	Modified SO ($\alpha, \gamma, \varepsilon_{decay}$)	Time (s)	Evals
42	(0.3346, 0.8582, 0.9266)	479.9	110
7	(0.4230, 0.9337, 0.9447)	478.9	110
13	(0.4427, 0.8738, 0.9395)	477.2	110
21	(0.4095, 0.9207, 0.9726)	477.7	110
99	(0.4782, 0.9474, 0.9950)	475.4	110

Table 12. Big Scenario: Comparative Performance Summary (mean ± 95% CI, 5 seeds)

Agent	Lat mean (s)	Lat std	En mean (J)	En std	OptRate (%)	P95 Lat (s)
Q-Learning	0.0497	0.0010	1.76×10^{-2}	9.35×10^{-4}	48.0	0.1256
SARSA	0.0505	0.0007	1.84×10^{-2}	7.36×10^{-4}	47.7	0.1266
DQN	0.0488	0.0003	1.65×10^{-2}	1.90×10^{-4}	45.7	0.1246
QLSO	0.0493	0.0012	1.72×10^{-2}	1.16×10^{-3}	49.0	0.1252

5.3.9. Scalability Analysis

The scalability of the proposed QLSO algorithm was tested by changing the number of IIoT devices between 5 and 30 clients while the task and resource parameters were kept unchanged. We were interested in the algorithmic benefits of the proposed approach exhibited in small scales continue to hold in larger scales, a significant issue for real industrial IIoT scenarios. Table 13 contains the best hyperparameters solution and the time of optimization for different number of clients. The results of the scalability analysis are encouraging. First, the wall-clock time for the SO scales roughly linearly with the number of clients (from 69.6s for 5 clients to 118.6s for 30 clients). This demonstrates that the scalability of the proposed framework is acceptable for most practical systems. Second, as the system scales up, the performance of the latency-optimal decision rate does not degrade substantially (from 45.0% to 43.2%). This demonstrates that the proposed QLSO algorithm will retain the advantages of the QLSO decision at a larger scale. The scalability performance of the proposed QLSO algorithm across different scales is summarized in the Table 14.

Table 13. Scalability Analysis: Optimal Hyperparameters Across Client Scales

Clients	Modified SO ($\alpha, \gamma, \varepsilon_{decay}$)	SO Time (s)	Evals
5	(0.5000, 0.9538, 0.9360)	69.6	110
10	(0.4443, 0.8319, 0.9947)	80.1	110
20	(0.3302, 0.9900, 0.9950)	100.6	110
30	(0.3696, 0.8305, 0.9950)	118.6	110

Table 14. Scalability Analysis: Performance Metrics Across Client Scales (mean across 5 seeds)

Clients	SO Time (s)	Avg Latency (s)	Avg Energy (J)	Optimal Rate (%)
5	69.6	0.0145	6.12×10^{-3}	45.0
10	80.1	0.0153	6.79×10^{-3}	44.4
20	100.6	0.0168	7.45×10^{-3}	43.8
30	118.6	0.0175	8.12×10^{-3}	43.2

6. DISCUSSION

In this section, the results of the research will be fully interpreted based on a four part analysis, including: main findings of the current study, comparisons with other research, implications and explanations of the findings, and strengths and limitations.

6.1. Main Findings of the Present Study

The main contribution of the paper is the idea of automated hyperparameter optimization using the SO to improve classical QL algorithm for computation offloading task performance. This is reflected by the following metrics: approximately 1.5× faster convergence, 9.1% fitness improvement measured in the ablation study, and 49.0% latency-optimized decision rate in the complex scenario. The statistical tests over five independent random seeds with 95% confidence interval guarantees the stability of the results to random initialization. The ablation study shows that the contribution of hyperparameter optimization is rather considerable: vanilla SO can already improve fitness by 9.1% versus fixed hyperparameters.

6.2. Comparison with Other Studies

The presented QLSO framework‘s performance is put into the context of RL-based computation offloading literature for the IIoT environment, as summarized in Table 15. Various DRL-based offloading algorithms have previously been proposed, however each work revealed limitations, some of which have been addressed in this paper: Qin *et al*. [43] proposed the DCEDRL method, which combines density clustering and ensemble learning with DRL in order to produce over 21% task backlog reduction and around 25.25% performance improvement compared with LyDROO in MEC in an environment of between 10 and 30 devices, but shows higher performance at high traffic loads and tests one to six deep neural network models, with five selected as optimal, and also incorporates grid search based parameter tuning of the clustering optimization component. The use of multiple DNN models increases computational requirements. Excessive iterations of an algorithm are, in many cases, simply replaced with more ample hardware resources as a solution, but in an IIoT environment such as the one implemented in this work, there are often constraints on processing capacity which should be considered. Zhang *et al*. [13] proposed an improved soft actor-critic-based DRL framework for cooperative partial task offloading and resource allocation in an end-edge, edge-edge, and edge-cloud IIoT environment. The proposed method was shown to reduce average task execution latency and total system costs compared with baseline cooperative frameworks and DRL algorithms. However, the study is focused on partial offloading scenarios and did not investigate the effects of hyperparameters on convergence characteristics. Additionally, this work did not include any automated hyperparameter optimization. Chai *et al*. [32] utilized DRL-based algorithms combined with task queuing models in a multi-user and multi-node MEC environment. The experimental results show that the proposed algorithm reduces average delay, average energy consumption, and the ratio of dropped tasks compared with several benchmark algorithms. However, although the study evaluates performance under different numbers of edge clients, it does not provide large-scale MEC/IIoT deployment validation. Cai *et al*. [9] proposed a multi-task multi-objective MADRL-based computation offloading method for IIoT offloading. The proposed model was shown to outperform weighted-sum optimization methods in terms of system cost, latency, and energy consumption. However, although the model uses a multi-agent architecture with separate latency- and energy-oriented agents, the study did not include an automated hyperparameter optimization procedure. The methodology improvement achieved by the proposed QLSO framework, in comparison to these existing algorithms in the current bibliography, is the use of automated hyperparameter optimization through the SO approach, which results in 1.5x faster convergence, and has a beneficial impact on the entropy of decisions made while showing a 49.0% ratio of latency-optimal decisions, as stated in the works findings. The statistical evaluation methodology used five independent random seeds and 95% confidence intervals to support the reported improvements.

6.3. Implication and Explanation of Findings

The performance behavior of QLSO can be attributed to three factors. First, SO provides derivative-free populations search over a compact continuous HPO space. Second, the optimized exploration schedule helps tabular QL reach stable policies earlier. Third, the offline HPO stage separates tuning overhead from online

decision-making. However, because BO, PSO, GA, and population-based training were not included as direct baselines, the results should be interpreted as evidence that SO is effective in this setting, not as evidence that SO is the best possible HPO method.

6.4. Strengths and Limitations

The strengths of this study include: (1) explicit multi-seed validation; (2) ablation studies that isolate the contribution of SO-based HPO; (3) evaluation across multiple latency, energy, cost, decision-quality, and scalability metrics; and (4) documented offline computational overhead for practical planning. The limitations of this study include: (1) the full-offloading model excludes partial offloading; (2) the compact tabular state representation excludes continuous channel fading, interference, queue lengths, mobility, and server-load dynamics; (3) the single-agent centralized architecture may not scale to large distributed IIoT networks; (4) the cost weights are fixed without a full sensitivity study; (5) baselines are compared mainly under fixed/default tuning rather than an equal HPO budget for every algorithm; and (6) the evaluation is simulation-based and has not yet been validated in NS-3, CloudSim, EdgeCloudSim, or a physical IIoT testbed.

Table 15. Comparison of RL-Based Computation Offloading Methods in MEC/IIoT Environments

Reference	Method	Environment	Key Metrics	Reported Results	Limitations
[43]	DCEDRL: density clustering + ensemble DRL	Mobile Edge Computing, N=10,20, and 30 devices	Task backlog, queue length, throughput, latency, energy	>21% backlog reduction; 25.25% improvement vs. LyDROO; 15.54-19.46% throughput gain	Requires multiple DNN models (5 optimal); higher time overhead; No automated DRL-HPO methodology
[13]	ISAC-CPTORA: improved SAC-based partial offloading and resource allocation	IIoT end-edge, edge-edge, and edge-cloud cooperative environment	Average total system cost, average task execution latency, average energy consumption	Lower total cost, task latency, and energy than baseline frameworks and DRL algorithms	Partial-offloading focus; no automated HPO or hyperparameter-sensitivity analysis
[32]	DDTO-DRL with task queuing	Multi-user, multi-node distributed MEC	Average delay, energy, dropped-task ratio	Reduced delay, energy, and dropped-task ratio compared with benchmark algorithms	Scalability mainly tested by varying edge clients; no large-scale MEC/IIoT validation; Fixed hyperparameters and no automated HPO
[9]	MA3MCO: multiagent, multitask, multiobjective DRL	Cloud-edge-device IIoT offloading	System cost, latency, energy	Outperformed weighted-sum methods in cost, latency, and energy consumption	Multiagent actor-critic; fixed learning parameters; no automated HPO
Our Work	Q-Learning + Snake Optimizer	Three-tier IIoT-edge-cloud	Convergence speed, latency, energy, decision quality, fitness	≈1.5× faster convergence; 49.0% optimal decision rate; 9.1% fitness improvement (ablation); 95% CIs reported across 5 seeds	Binary offloading only; single-agent centralized

7. CONCLUSION

This study presented QLSO, an adaptive SO-based HPO framework for QL-based computation offloading in IIoT environments. The main contribution of this study is the integration of SO as a meta-heuristic HPO layer within the QL-based offloading workflow to improve latency-energy-aware decision-making in a three-tier IIoT environment. Across five independent random seeds, the framework improved convergence behavior and achieved a 49.0% latency-optimal decision rate in the complex scenario. The ablation study further showed that SO-based tuning improved fixed-hyperparameter QL, with a 9.1% fitness gain for the Standard SO variant and a 6.5% gain for the Modified SO variant. These findings indicate that HPO is an important factor affecting RL-based offloading performance.

Despite promising results and indications of the SO-based HPO approach's potential, the present work should be interpreted as evidence of simulation-level effectiveness rather than deployment readiness, because the current assessment remains limited to simulation-based evaluation. The current simulation does not fully account for several real-world IIoT factors, such as channel fading and interference, dynamic server loads, queuing dynamics, mobility, partial offloading, and heterogeneous QoS requirements of industrial applications. Furthermore, the single centralized-agent design may face scalability challenges in large distributed industrial networks. Future work will extend the proposed framework by incorporating offloading prediction, partial offloading, multi-agent coordination, and dynamic network reconfiguration. These extensions will be pursued

by expanding the offloading action space to support partial task execution, replacing the centralized agent with coordinated multi-agent decision-making, and validating the framework under dynamic channel, queuing, server-load, and heterogeneous QoS conditions. Future validation will be conducted using more realistic simulation platforms such as NS-3, CloudSim, and EdgeCloudSim, as well as physical IIoT testbeds with communication protocols such as MQTT and OPC-UA.

REFERENCES

[1] T. Qiu, J. Chi, X. Zhou, Z. Ning, M. Atiquzzaman, and D. O. Wu, "Edge Computing in Industrial Internet of Things: Architecture, Advances and Challenges," *IEEE Commun. Surv. Tutorials*, vol. 22, no. 4, pp. 2462–2488, 2020, https://doi.org/10.1109/COMST.2020.3009103.

[2] X. Jiao *et al.*, "Deep Reinforcement Learning for Time-Energy Tradeoff Online Offloading in MEC-Enabled Industrial Internet of Things," *IEEE Trans. Netw. Sci. Eng.*, vol. 10, no. 6, pp. 3465–3479, 2023, https://doi.org/10.1109/TNSE.2023.3263169.

[3] B. Chen, J. Wan, Y. Lan, M. Imran, D. Li, and N. Guizani, "Improving cognitive ability of edge intelligent IIoT through machine learning," *IEEE Netw.*, vol. 33, no. 5, pp. 61–67, 2019, https://doi.org/10.1109/MNET.001.1800505.

[4] L. Huang, S. Bi, and Y. J. A. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Trans. Mob. Comput.*, vol. 19, no. 11, pp. 2581–2593, 2020, https://doi.org/10.1109/TMC.2019.2928811.

[5] H. Yang, Y. Liang, J. Yuan, Q. Yao, A. Yu, and J. Zhang, "Distributed Blockchain-Based Trusted Multidomain Collaboration for Mobile Edge Computing in 5G and beyond," *IEEE Trans. Ind. Informatics*, vol. 16, no. 11, pp. 7094–7104, 2020, https://doi.org/10.1109/TII.2020.2964563.

[6] B. Xu *et al.*, "Mobility-Aware Task Offloading in Industrial Fog Networks: A Submodular-Based MARL Approach," *IEEE Internet Things J.*, vol. 12, no. 8, pp. 10795–10807, 2025, https://doi.org/10.1109/JIOT.2024.3514095.

[7] X. H. Deng, J. Yin, P. Y. Guan, N. N. Xiong, L. Zhang, and S. Mumtaz, "Intelligent Delay-Aware Partial Computing Task Offloading for Multiuser Industrial Internet of Things Through Edge Computing," *IEEE INTERNET THINGS J.*, vol. 10, no. 4, pp. 2954–2966, 2023, https://doi.org/10.1109/JIOT.2021.3123406.

[8] W. Qin, H. Chen, L. Wang, Y. Xia, A. Nascita, and A. Pescapè, "MCOTM: Mobility-aware computation offloading and task migration for edge computing in industrial IoT," *Futur. Gener. Comput. Syst.*, vol. 151, no. March 2023, pp. 232–241, 2024, https://doi.org/10.1016/j.future.2023.10.004.

[9] J. Cai, H. T. Fu, and Y. Liu, "Multitask Multiobjective Deep Reinforcement Learning-Based Computation Offloading Method for Industrial Internet of Things," *IEEE INTERNET THINGS J.*, vol. 10, no. 2, pp. 1848–1859, 2023, https://doi.org/10.1109/JIOT.2022.3209987.

[10] M. Wu, W. Chen, L. Qian, L. Guo, and I. Lee, "Joint Service Caching and Secure Computation Offloading for Reconfigurable-Intelligent-Surface-Assisted Edge Computing Networks," *IEEE Internet Things J.*, vol. 11, no. 19, pp. 30469–30482, 2024, https://doi.org/10.1109/JIOT.2024.3404972.

[11] X. Li *et al.*, "ABM-SpConv-SIMD: Accelerating Convolutional Neural Network Inference for Industrial IoT Applications on Edge Devices," *IEEE Trans. Netw. Sci. Eng.*, vol. 10, no. 5, pp. 3071–3085, 2023, https://doi.org/10.1109/TNSE.2022.3154412.

[12] J. Chen, H. Xing, Z. Xiao, L. Xu, and T. Tao, "A DRL Agent for Jointly Optimizing Computation Offloading and Resource Allocation in MEC," *IEEE Internet Things J.*, vol. 8, no. 24, pp. 17508–17524, 2021, https://doi.org/10.1109/JIOT.2021.3081694.

[13] F. Zhang, G. J. Han, L. Liu, M. Martínez-García, and Y. Peng, "Deep Reinforcement Learning Based Cooperative Partial Task Offloading and Resource Allocation for IIoT Applications," *IEEE Trans. Netw. Sci. Eng.*, vol. 10, no. 5, pp. 2991–3006, 2023, https://doi.org/10.1109/TNSE.2022.3167949.

[14] M. S. Hossain, C. I. Nwakanma, J. M. Lee, and D. S. Kim, "Edge computational task offloading scheme using reinforcement learning for IIoT scenario," *ICT EXPRESS*, vol. 6, no. 4, pp. 291–299, 2020, https://doi.org/10.1016/j.icte.2020.06.002.

[15] Y. Zhao, Z. Chai, Y. Li, H. Huang, and H. Kang, "Multi-Objective Computation Offloading Based on Decentralized Deep Reinforcement Learning in Industrial Internet of Things," *IEEE Trans. Cogn. Commun. Netw.*, vol. 11, no. 3, pp. 1940–1953, 2025, https://doi.org/10.1109/TCCN.2024.3466889.

[16] S. Koo and Y. Lim, "A Cluster-Based Optimal Computation Offloading Decision Mechanism Using RL in the IIoT Field," *Appl. Sci.*, vol. 12, no. 1, 2022, https://doi.org/10.3390/app12010384.

[17] Z. Zabihi, A. M. Eftekhari Moghadam, and M. H. Rezvani, "Reinforcement Learning Methods for Computation Offloading: A Systematic Review," *ACM Comput. Surv.*, vol. 56, no. 1, 2023, https://doi.org/10.1145/3603703.

[18] K. H. Liu and W. Liao, "Intelligent Offloading for Multi-Access Edge Computing: A New Actor-Critic Approach," *IEEE Int. Conf. Commun.*, vol. 2020-June, 2020, https://doi.org/10.1109/ICC40277.2020.9149387.

[19] B. Zhang *et al.*, "On the Importance of Hyperparameter Optimization for Model-based Reinforcement Learning," *Proc. Mach. Learn. Res.*, vol. 130, pp. 4015–4023, 2021, https://proceedings.mlr.press/v130/zhang21n.html.

[20] F. A. Hashim and A. G. Hussien, "Snake Optimizer: A novel meta-heuristic optimization algorithm," *Knowledge-Based Syst.*, vol. 242, p. 108320, 2022, https://doi.org/10.1016/j.knosys.2022.108320.

[21] J. Youn, "Intelligent Task Offloading Method using Deep Q-Network for Collaborative Edge Computing System," *5th Int. Conf. Artif. Intell. Inf. Commun. ICAIIC 2023*, pp. 872–875, 2023, https://doi.org/10.1109/ICAIIC57133.2023.10067111.
[22] H. Zhai, X. Zhou, H. Zhang, and D. Yuan, "Delay Minimization in Hybrid Edge Computing Networks: A DDQN-Based Task Offloading Approach," *IEEE Trans. Veh. Technol.*, vol. PP, pp. 1–11, 2024, https://doi.org/10.1109/TVT.2024.3407483.
[23] B. Dai, Y. Qiu, and W. Feng, "Scalable Computation Offloading for Industrial IoTs via Distributed Deep Reinforcement Learning," *Proc. 2024 27th Int. Conf. Comput. Support. Coop. Work Des. CSCWD 2024*, pp. 1681–1686, 2024, https://doi.org/10.1109/CSCWD61410.2024.10580311.
[24] G. Nieto, I. de la Iglesia, U. Lopez-Novoa, and C. Perfecto, "Deep Reinforcement Learning techniques for dynamic task offloading in the 5G edge-cloud continuum," *J. Cloud Comput.*, vol. 13, no. 1, 2024, https://doi.org/10.1186/s13677-024-00658-0.
[25] Y. Chen, X. Li, J. Ye, X. Wang, and Q. Chen, "Multi-agent Reinforcement Learning Based Resource Allocation in End-Edge-Cloud Enabled Industrial Internet of Things," *Proc. - IEEE Congr. Cybermatics 2023 IEEE Int. Conf. Internet Things, iThings 2023, IEEE Green Comput. Commun. GreenCom 2023, IEEE Cyber, Phys. Soc. Comput. CPSCom 2023 IEEE Smart Data, SmartD*, pp. 13–19, 2023, https://doi.org/10.1109/iThings-GreenCom-CPSCom-SmartData-Cybermatics60724.2023.00027.
[26] S. Chouikhi, M. Esseghir, and L. Merghem-Boulahia, "Energy-Efficient Computation Offloading Based on Multiagent Deep Reinforcement Learning for Industrial Internet of Things Systems," *IEEE INTERNET THINGS J.*, vol. 11, no. 7, pp. 12228–12239, 2024, https://doi.org/10.1109/JIOT.2023.3333044.
[27] L. Ai, B. Tan, J. Zhang, R. Wang, and J. Wu, "Dynamic Offloading Strategy for Delay-Sensitive Task in Mobile-Edge Computing Networks," *IEEE Internet Things J.*, vol. 10, no. 1, pp. 526–538, 2023, https://doi.org/10.1109/JIOT.2022.3202797.
[28] R. Ma, X. Zhou, H. Zhang, and D. Yuan, "Joint Optimization of Energy Consumption and Latency Based on DRL: An Edge Server Activation and Task Scheduling Scheme in IIoT," *2022 IEEE 14th Int. Conf. Wirel. Commun. Signal Process. WCSP 2022*, pp. 203–208, 2022, https://doi.org/10.1109/WCSP55476.2022.10039283.
[29] A. Xu *et al.*, "QDRL: Queue-Aware Online DRL for Computation Offloading in Industrial Internet of Things," *IEEE Internet Things J.*, vol. 11, no. 5, pp. 7772–7786, 2024, https://doi.org/10.1109/JIOT.2023.3316139.
[30] H. Mai Do, T. P. Tran, and M. Yoo, "Deep Reinforcement Learning-Based Task Offloading and Resource Allocation for Industrial IoT in MEC Federation System," *IEEE Access*, vol. 11, no. July 2023, pp. 83150–83170, 2023, https://doi.org/10.1109/ACCESS.2023.3302518.
[31] Y. Gong, H. Yao, J. Wang, M. Li, and S. Guo, "Edge Intelligence-Driven Joint Offloading and Resource Allocation for Future 6G Industrial Internet of Things," *IEEE Trans. Netw. Sci. Eng.*, vol. 11, no. 6, pp. 5644–5655, 2024, https://doi.org/10.1109/TNSE.2022.3141728.
[32] Z. Chai, H. Hou, and Y. Li, "A dynamic queuing model based distributed task offloading algorithm using deep reinforcement learning in mobile edge computing," *Appl. Intell.*, vol. 53, no. 23, pp. 28832–28847, 2023, https://doi.org/10.1007/s10489-023-05065-7.
[33] F. Zhang, G. Han, L. Liu, Y. Zhang, Y. Peng, and C. Li, "Cooperative Partial Task Offloading and Resource Allocation for IIoT Based on Decentralized Multiagent Deep Reinforcement Learning," *IEEE Internet Things J.*, vol. 11, no. 3, pp. 5526–5544, 2024, https://doi.org/10.1109/JIOT.2023.3306803.
[34] W. J. Cheng, X. S. Liu, X. T. Wang, and G. F. Nie, "Task Offloading and Resource Allocation for Industrial Internet of Things: A Double-Dueling Deep Q-Network Approach," *IEEE ACCESS*, vol. 10, pp. 103111–103120, 2022, https://doi.org/10.1109/ACCESS.2022.3210248.
[35] S. Chen, J. Chen, Y. Miao, Q. Wang, and C. Zhao, "Deep Reinforcement Learning-Based Cloud-Edge Collaborative Mobile Computation Offloading in Industrial Networks," *IEEE Trans. Signal Inf. Process. over Networks*, vol. 8, pp. 364–375, 2022, https://doi.org/10.1109/TSIPN.2022.3171336.
[36] G. Wu, Z. Xu, H. Zhang, S. Shen, and S. Yu, "Multi-agent DRL for joint completion delay and energy consumption with queuing theory in MEC-based IIoT," *J. Parallel Distrib. Comput.*, vol. 176, pp. 80–94, 2023, https://doi.org/10.1016/j.jpdc.2023.02.008.
[37] T. Eimer, M. Lindauer, and R. Raileanu, "Hyperparameters in Reinforcement Learning and How To Tune Them," *Proc. Mach. Learn. Res.*, vol. 202, pp. 9104–9149, 2023, https://proceedings.mlr.press/v202/eimer23a.
[38] J. Parker-Holder *et al.*, "Automated Reinforcement Learning (AutoRL): A Survey and Open Problems," *J. Artif. Intell. Res.*, vol. 74, pp. 517–568, 2022, https://doi.org/10.1613/jair.1.13596.
[39] B. Bischl *et al.*, "Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges," *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, vol. 13, no. 2, pp. 1–43, 2023, https://doi.org/10.1002/widm.1484.
[40] N. Awad, N. Mallik, and F. Hutter, "DEHB: Evolutionary Hyberband for Scalable, Robust and Efficient Hyperparameter Optimization," *IJCAI Int. Jt. Conf. Artif. Intell.*, no. 3, pp. 2147–2153, 2021, https://doi.org/10.24963/ijcai.2021/296.
[41] J. Parker-Holder, V. Nguyen, and S. J. Roberts, "Provably efficient online hyperparameter optimization with population-based bandits," *Adv. Neural Inf. Process. Syst.*, vol. 33, pp. 17200-17211, 2020, https://proceedings.neurips.cc/paper/2020/hash/c7af0926b294e47e52e46cfebe173f20-Abstract.html.
[42] T. Yu and H. Zhu, "Hyper-Parameter Optimization: A Review of Algorithms and Applications," *arXiv preprint arXiv:2003.05689*, 2020, https://doi.org/10.48550/arXiv.2003.05689.

[43] Y. Qin, J. Chen, L. Jin, R. Yao, and Z. Gong, "Task offloading optimization in mobile edge computing based on a deep reinforcement learning algorithm using density clustering and ensemble learning," *Sci. Rep.*, vol. 15, no. 1, pp. 1–25, 2025, https://doi.org/10.1038/s41598-024-84038-3.
[44] H. Ke, J. Wang, L. Deng, Y. Ge, and H. Wang, "Deep Reinforcement Learning-Based Adaptive Computation Offloading for MEC in Heterogeneous Vehicular Networks," *IEEE Trans. Veh. Technol.*, vol. 69, no. 7, pp. 7916–7929, 2020, https://doi.org/10.1109/TVT.2020.2993849.
[45] Y. He, J. Ren, G. Yu, and Y. Cai, "D2D communications meet mobile edge computing for enhanced computation capacity in cellular networks," *IEEE Trans. Wirel. Commun.*, vol. 18, no. 3, pp. 1750–1763, 2019, https://doi.org/10.1109/TWC.2019.2896999.
[46] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4005–4018, 2019, https://doi.org/10.1109/JIOT.2018.2876279.
[47] A. Hazra and T. Amgoth, "CeCO: Cost-Efficient Computation Offloading of IoT Applications in Green Industrial Fog Networks," *IEEE Trans. Ind. INFORMATICS*, vol. 18, no. 9, pp. 6255–6263, 2022, https://doi.org/10.1109/TII.2021.3130255.
[48] J. Peng, H. Qiu, J. Cai, W. Xu, and J. Wang, "D2D-Assisted Multi-User Cooperative Partial Offloading, Transmission Scheduling and Computation Allocating for MEC," *IEEE Trans. Wirel. Commun.*, vol. 20, no. 8, pp. 4858–4873, 2021, https://doi.org/10.1109/TWC.2021.3062616.
[49] X. Jiang, F. R. Yu, T. Song, and V. C. M. Leung, "A Survey on Multi-Access Edge Computing Applied to Video Streaming: Some Research Issues and Challenges," *IEEE Commun. Surv. Tutorials*, vol. 23, no. 2, pp. 871–903, 2021, https://doi.org/10.1109/COMST.2021.3065237.
[50] R. Chai, J. Lin, M. Chen, and Q. Chen, "Task execution cost minimization-based joint computation offloading and resource allocation for cellular D2D MEC systems," *IEEE Syst. J.*, vol. 13, no. 4, pp. 4110–4121, 2019, https://doi.org/10.1109/JSYST.2019.2921115.
[51] M. E. Yüksel, "Power consumption analysis of a Wi-Fi-based IoT device," *Electrica*, vol. 20, no. 1, pp. 62–70, 2020, https://doi.org/10.5152/ELECTRICA.2020.19081.
[52] L. P. Qian, Y. Wu, F. L. Jiang, N. N. Yu, W. D. Lu, and B. Lin, "NOMA Assisted Multi-Task Multi-Access Mobile Edge Computing via Deep Reinforcement Learning for Industrial Internet of Things," *IEEE Trans. Ind. INFORMATICS*, vol. 17, no. 8, pp. 5688–5698, 2021, https://doi.org/10.1109/TII.2020.3001355.
[53] X. Jin, S. Zhang, Y. Ding, and Z. Wang, "Task offloading for multi-server edge computing in industrial Internet with joint load balance and fuzzy security," *Sci. Rep.*, vol. 14, no. 1, pp. 1–26, 2024, https://doi.org/10.1038/s41598-024-79464-2.
[54] Z. Aghapour, S. Sharifian, and H. Taheri, "Task offloading and resource allocation algorithm based on deep reinforcement learning for distributed AI execution tasks in IoT edge computing environments," *Comput. Networks*, vol. 223, no. January, p. 109577, 2023, https://doi.org/10.1016/j.comnet.2023.109577.
[55] J. Chi, X. Zhou, F. Xiao, Y. Lim, and T. Qiu, "Task Offloading via Prioritized Experience-Based Double Dueling DQN in Edge-Assisted IIoT," *IEEE Trans. Mob. Comput.*, vol. 23, no. 12, pp. 14575–14591, 2024, https://doi.org/10.1109/TMC.2024.3452502.
[56] S. M. Shithil and M. A. Adnan, "A Prediction Based Replica Selection Strategy for Reducing Tail Latency in Geo-Distributed Systems," *IEEE Trans. Cloud Comput.*, vol. 11, no. 3, pp. 2954–2965, 2023, https://doi.org/10.1109/TCC.2023.3244203.
[57] Z. Zhou, L. Pan, and S. Liu, "Online Traffic Allocation for Video Service Providers in Cloud-Edge Cooperative Systems," *IEEE Trans. Serv. Comput.*, vol. 18, no. 3, pp. 1618–1626, 2025, https://doi.org/10.1109/TSC.2025.3565363.

AUTHOR BIOGRAPHY

Hayder Salman Dawood, received a bachelor's degree in Computer Engineering from the University of Basrah, Iraq, in 2003, and received a master's degree in Communications and Information Systems Engineering from Huazhong University of Science and Technology (HUST), China, in 2013. Now he is pursuing the Ph.D. with the Faculty of Computing at Universiti Teknologi Malaysia (UTM). He has more than 12 years of experience in the Oil and Gas Industrial Sector. His research interests include Industrial Internet of Things (IIoT), Edge Computing, Task Offloading, Machine Learning (particularly Reinforcement Learning), and Optimization Algorithms.
Email: dawood-20@graduate.utm.my
Orcid: https://orcid.org/0009-0006-4452-4473

Ismail Fauzi Isnin, is an Associate Professor at the Faculty of Computing, Universiti Teknologi Malaysia (UTM). He holds a Ph.D. in Wireless Digital Communications from the University of Plymouth, United Kingdom. His research interest spans several key areas, including wireless sensor networks, vehicular ad-hoc networks (VANETs), network security, intrusion detection, and quantum emulation. Dr. Isnin has published in journals and international conferences, contributing to the fields of network protocols, secure communications, and machine learning-based security models.
Email: ismailfauzi@utm.my
Orcid: https://orcid.org/0000-0002-9765-3491

Muhalim bin Mohamed Amin, is a Senior Lecturer at the Faculty of Computing, Universiti Teknologi Malaysia (UTM). He holds a Ph.D. in Computer Science from UTM, and an earlier degree in Computer Science (Computing System Design) from the University of Manchester Institute of Science and Technology (UMIST), United Kingdom, following his foundational studies in Computer Science at UTM. His research interests include computer networks, cybersecurity, cryptography, digital fraud analysis, and steganography. He has also developed an emerging interest in Post-Quantum Cryptography (PQC) in line with current global trends in cryptographic research. Dr. Muhalim is an active member of the Information Assurance and Security Research Group (IASRG) at UTM and has contributed numerous publications across key areas of information security. He currently serves as the Programme Coordinator for the Bachelor of Computer Science (Computer Network and Security) programme at the Faculty of Computing. Since 2000, he has been teaching various courses related to network security, system software, and computer networking,
Email: muhalim@utm.my
Orcid: https://orcid.org/0000-0002-7277-9901

Ahmad Najmi bin Amerhaider Nuar, received the PhD degree in Information Systems from Universiti Teknologi Malaysia (UTM), Johor, Malaysia, in 2022. He is currently a senior lecturer with the Faculty of Computing, UTM, and also serves as the Chief Executive Officer of NexScholar Sdn. Bhd. His research interests include artificial intelligence, big data analytics, academic information systems, and design science research methodologies.
Email: ahmadnajmi.an@utm.my
Orcid: https://orcid.org/0000-0003-3103-664X