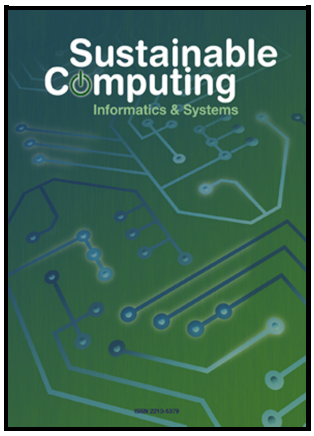
Journal Pre-proof

QTE-IoT: Q-Learning-Based Task Scheduling Scheme to Enhance Energy Consumption and QoS in IoT Environments



Ali Ghaffari, Vesal Firoozi, Ali Maleki, Mohammad Sadegh Sirjani, Maedeh Abedini Bagha

PII: S2210-5379(25)00168-4

DOI: https://doi.org/10.1016/j.suscom.2025.101247

Reference: SUSCOM101247

To appear in: *Sustainable Computing: Informatics and Systems*

Received date: 21 May 2024
Revised date: 29 August 2025
Accepted date: 4 November 2025

Please cite this article as: Ali Ghaffari, Vesal Firoozi, Ali Maleki, Mohammad Sadegh Sirjani and Maedeh Abedini Bagha, QTE-IoT: Q-Learning-Based Task Scheduling Scheme to Enhance Energy Consumption and QoS in IoT Environments, *Sustainable Computing: Informatics and Systems,* (2025) doi:https://doi.org/10.1016/j.suscom.2025.101247

QTE-IoT: Q-Learning-Based Task Scheduling Scheme to Enhance Energy Consumption and QoS in IoT Environments

Ali Ghaffari[1,2,3,*], Vesal Firoozi[4], Ali Maleki[5], Mohammad Sadegh Sirjani[6], Maedeh Abedini Bagha[1,7]

[1] Department of Computer Engineering, Ta.C., Islamic Azad University, Tabriz, Iran.
[2]Department of Computer Engineering, Faculty of Engineering and Natural Science, Istinye University, Istanbul, Türkiye.
[3]Department of Computer Science, Khazar University, Baku, Azerbaijan.
[4] Department of Computer Engineering, Ma.C., Islamic Azad University, Mashhad, Iran.
[5]Department of Computer Engineering, Shiraz Branch, Islamic Azad University, Shiraz, Iran.
[6] Department of Electronics, Information and Bioengineering, Politecnico di Milano, Milan, Italy.
[7] Roshdiyeh Higher Education Institute, Tabriz, East Azerbaijan Province, Iran.
***Correspond author email: A.ghaffari@iaut.ac.ir**

Abstract

As the proliferation of Internet of Things (IoT) devices continues unabated, the demand for efficient task scheduling mechanisms becomes increasingly critical. Task scheduling in the IoT is pivotal for optimizing resource utilization, minimizing latency, and enhancing the overall system's performance. This research proposes a novel method called QTE-IoT, standing for a Q-learning-based task scheduling scheme to enhance energy consumption and QoS in IoT environments. QTE-IoT commences by categorizing tasks into three classes: time-sensitive tasks, security tasks, and normal tasks. This classification is achieved using a multi-layer perceptron artificial neural network. Subsequently, time-sensitive tasks are offloaded to the fog layer and scheduled using the proposed African Vulture Algorithm combined with Q-learning, which we designate as QAVA. Security tasks are offloaded to the private cloud, while normal tasks are offloaded to the public cloud. For task scheduling in private and public cloud environments, QTE-IoT employs a proposed enhanced version of Artificial Rabbits Optimization integrated with the Q-learning algorithm, known as QARO. Additionally, the QTE-IoT method incorporates a monitoring agent to oversee resource workload, thereby preventing congestion and delays. Simulation results on instances of the HCSP benchmark dataset demonstrate that QTE-IoT outperforms other state-of-the-art methods in various performance metrics. QTE-IoT achieves significant improvements compared to other methods and algorithms, including a 6% to 12% reduction in energy consumption. Furthermore, QTE-IoT exhibits substantial improvements in load imbalance (42% to 79%), response time (25% to 40%), and deadline satisfaction (6% to 39%) compared to existing approaches.

Keywords: Internet of things, Metaheuristic, Reinforcement learning, Task scheduling.

1. Introduction

The Internet of Things (IoT) has revolutionized how we interact with technology, enabling seamless connectivity and communication between devices [1]. Because of the IoT's capacity to monitor the external environment and facilitate rapid and precise decision-making, this technology finds application in various fields, such as healthcare [2], smart city initiatives, video surveillance [3], and emergency response systems [4], Autonomous Vehicles [5], Space-Air-Ground Networks [6], the agricultural information system [7]. However, IoT devices are constrained by their limited computing and storage capacities. Consequently, handling and analyzing the data, particularly large volumes of data, within these devices becomes challenging. With technological progress, cloud computing, equipped with extensive storage and processing capabilities, has been employed to handle the storage and analysis of the data of IoT devices [8]. Due to the high computational demands, IoT devices transmit data to cloud-distributed systems for storage, processing, analysis, and decision-making [9]. Nevertheless, cloud computing servers are often

situated at a considerable physical distance from IoT devices, leading to significant latency that hinders the effective support of real-time IoT applications. In response to these challenges, edge and fog computing have gained prominence as alternative computing paradigms within the IoT domain [10]. Fog computing, on the other hand, facilitates data processing and analysis, bringing cloud-like services in close proximity to IoT applications. This proximity serves to decrease latency, conserve energy, and optimize bandwidth usage for IoT applications.

As shown in Figure 1, after significant advancements in IoT network architecture, we now have a system with a three-layer architecture of IoT, fog, and cloud layers, known as FCIoT [11]. The lowest layer is the IoT layer, which consists of smart IoT devices. These devices usually serve to collect data from the environment and make resource requests, which are sent to higher layers. The fog layer has resources, such as edge servers, routers, gateways, switches, and cloudlets, known as fog nodes (FNs). The top layer is the cloud layer, where there are public and private clouds. Clouds comprise data centers and virtual machines (VMs).

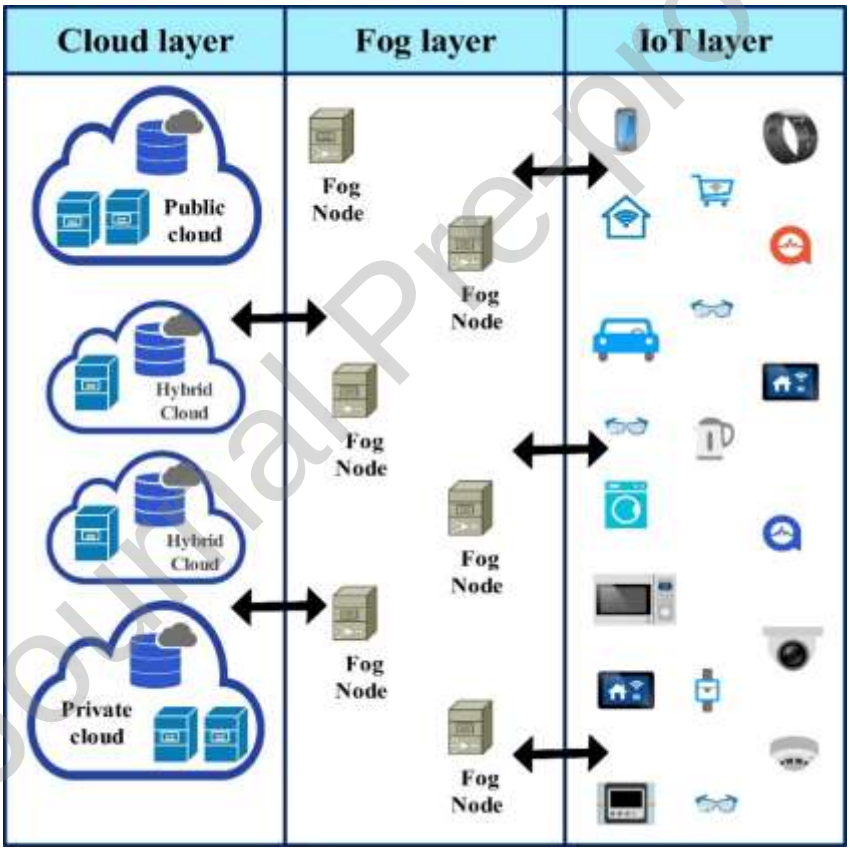


Fig. 1. The FCIoT architecture.

Public cloud services are typically hosted in data centers operated by third-party providers and are accessible over the Internet. These clouds offer high scalability and flexibility, allowing users to easily scale resources up or down based on demand. Performance in a public cloud can be affected by factors such as network latency, shared resources, and the location of data centers. Public clouds are well-suited for applications with variable workloads and global reach.

Private clouds are dedicated environments operated solely for a single organization, either on-premises or hosted by a third-party provider. These clouds offer greater control over resources, security, and performance than public clouds. Performance in a private cloud can be more consistent and predictable

since resources are not shared with other organizations. Private clouds are often used for applications with strict security and compliance requirements.

1.1 Problem Definition

This study considers the task scheduling problem within the three-layer FCIoT architecture, comprising IoT devices, fog nodes, and cloud servers. The objective is to determine an optimal mapping of a set of heterogeneous tasks generated by IoT devices to available computational resources across fog and cloud layers, such that key system performance metrics are optimized.

More formally, let $\boldsymbol{T} = \{T_1, T_2, \ldots, T_n\}$ be a set of n independent tasks, each characterized by parameters such as task size, deadline, input and output file size. Let $\boldsymbol{R} = \{R_1, R_2, \ldots, R_m\}$ denote the set of available computational resources, including fog nodes with limited capacity and cloud servers with higher but distant capacity.

The scheduling problem aims to assign each task $T_i \in T$ to a resource $R_j \in R$ in a manner that:

- Minimizes the overall energy consumption across fog and cloud resources.
- Minimizes response time and latency to guarantee timely execution, especially for time-sensitive tasks.
- Ensures load balancing among fog and cloud nodes to avoid overloads and bottlenecks.
- Maximizes Quality of Service (QoS) by meeting task deadlines and security constraints.

This allocation must observe the following constraints:

- Resource capacity constraints: Each resource node can process a limited number of tasks concurrently without degrading performance.
- Task-specific constraints: Tasks with strict deadlines or security requirements must be prioritized or assigned to suitable resources.
- Communication constraints: Network delays and bandwidth limitations between IoT devices, fog nodes, and cloud servers impact scheduling decisions.

The problem is formally an NP-hard combinatorial optimization challenge due to its multidimensional objectives and constraints, dynamic task arrivals, and the heterogeneity of the FCIoT environment [12]. Conventional exact algorithms are computationally infeasible for large-scale IoT systems, necessitating heuristic or machine learning-based scheduling strategies [13].

The overarching goal of this work is to design a task scheduling scheme that effectively addresses these challenges by optimizing energy efficiency and QoS metrics while managing resource trade-offs within the FCIoT infrastructure.

1.2. Study Motivation

Task scheduling (TS) plays a crucial role in enhancing system performance, effectively managing the load to address network overhead, optimizing resource utilization, and minimizing energy consumption [15]. The primary objective of TS involves mapping tasks to suitable resources, ensuring that task execution is completed while meeting the QoS requirements [16]. Despite the considerable advantages of cloud-fog computing, TS encounters challenges due to its dynamic nature, task setup, and resource demands[17]. These factors influence QoS optimization, which necessitates adjusting parameters and selecting the appropriate FCIoT resources. The goal of TS is optimizing metrics encompass various factors such as delay [18], energy consumption [19], load balancing, response time, cost, task deadline meeting, and more [20].

Latency is crucial in IoT task scheduling for several reasons. Many applications, like healthcare monitoring and autonomous vehicles, require real-time data processing; high latency can delay decision-making, negatively impacting outcomes [21]. Low latency is vital for maintaining high QoS in applications such as video streaming and online gaming [22]. In fog computing, efficient TS with low latency optimizes resource utilization, allowing tasks to be processed quickly and resources to be freed for others [23]. Reducing latency also lowers energy consumption, enabling devices to enter low-power states sooner and improving network efficiency, especially in environments with numerous connected devices [22].

Load balancing is essential in both cloud and fog computing, distributing workloads evenly across servers to prevent overloads [24]. This optimization enhances resource utilization, saves energy, and improves user performance through faster response times and reliable service [25]. Balanced loads reduce the risk of server failures due to excessive strain, increasing overall system reliability [26].

Energy efficiency is vital for achieving resource efficiency, cost savings, and environmental sustainability in FCIoT [27]. High energy consumption affects costs and QoS while contributing to carbon emissions. Reducing energy use lowers operational costs and ecological footprints. Effective task scheduling ensures timely processing of IoT requests while optimizing energy consumption, enhancing QoS and overall system performance [28].

Therefore, the provision and management of resources are imperative for maximizing the capabilities of IoT applications. Poor TS in FCIoT can lead to extensive response time compared to cloud computing. Hence, effective scheduling strategies are essential before the comprehensive use of fog infrastructure. Notably, resource scheduling poses an NP-hard problem with no apparent algorithm or perfect solution currently available. Addressing this challenge requires efficient and optimized methods to mitigate latency issues and ensure the optimal utilization of resources in FCIoT [29].

1.3. Gaps in the Literature

A solution is essential for the efficient allocation of resources to IoT tasks, aiming to enhance energy usage while adhering to QoS standards. Significant research has been conducted on TS, resulting in a variety of proposed solutions. However, this problem is classified as NP-hard, which complicates the development of precise algorithms and optimal solutions, especially when managing numerous resources and tasks. Machine learning techniques have demonstrated that the NP-hard nature of the problem often leads to time-consuming and impractical solutions for large-scale tasks [25, 30, 31]. To tackle NP-hard issues, metaheuristic approaches are frequently advocated [18, 32-34]. Some researchers [35-37] have proposed metaheuristic solutions, while others [11, 38, 39] have investigated hybrid strategies. Nonetheless, only a limited number of solutions address task scheduling across both fog and cloud layers while also considering critical factors such as energy consumption, load balancing, response time, and adherence to deadlines.

Numerous well-known metaheuristic methods are employed for task scheduling, including Particle Swarm Optimization (PSO), Genetic Algorithms (GA), Simulated Annealing (SA), and Ant Colony Optimization (ACO) [40]. However, these algorithms exhibit diverse search processes, issues with randomness, limited global search abilities, and reduced convergence in later iterations, which can lead to local optimum solutions [41]. Additionally, there is often an imbalance between global and local search strategies [20].

1.4. Proposed Solution

We employed proposed Q-learning based Artificial Rabbits Optimization (QARO) for TS in both private and public cloud environments. Artificial Rabbits Optimization (ARO) algorithm [42] offers several advantages, including its simplicity, ease of implementation, and effectiveness in tackling complex optimization problems. By emulating the natural foraging behavior of rabbits, ARO's population-based approach promotes diverse solution exploration, facilitating convergence towards optimal solutions. However, ARO also exhibits certain limitations, such as a susceptibility to premature convergence to local optima, especially in high-dimensional spaces. Additionally, its performance can be sensitive to parameter settings, and, like many nature-inspired algorithms, it may require numerous iterations to achieve satisfactory results, potentially leading to high computational costs in some applications. To address these limitations, we proposed QARO. This enhancement integrates reinforcement learning principles to adaptively adjust the exploration and exploitation strategies of the algorithm. By viewing the search space as an environment where each potential solution represents a state, QARO leverages Q-learning to evaluate the quality of solutions based on their fitness values, which serve as rewards. This allows QARO to learn optimal paths through the solution space over time, dynamically adapting its search behavior based on past experiences. As a result, this hybrid approach can improve convergence rates, mitigate the risk of premature

convergence to local optima, and enable more effective exploration of complex solution landscapes, ultimately leading to better optimization outcomes.

The AVA, introduced by Abdollahzadeh in 2021 [43], exhibits a strong Exploitation Phase, making it suitable for TS in the fog layer. Subsequent studies have explored enhancements to AVA, including employing diverse fitness functions for image segmentation [44], incorporating elite mutation techniques, dynamic opposition learning, and chaotic map-based population initialization [45], as well as integrating opposition-based learning with a sine-cosine approach to balance exploration and exploitation for improved population variance, applied to bidding strategies in China's two-settlement market [46]. This work presents a novel enhancement of AVA by incorporating the Q-learning algorithm for the first time. We propose Q-learning based AVA (QAVA), a hybrid approach that leverages the strengths of both methods, creating a more adaptive and efficient optimization algorithm. By combining AVA's exploitation prowess with Q-learning's adaptive learning capabilities, QAVA has the potential to achieve improved solution quality and faster convergence in complex optimization tasks.

Building on the impressive success of neural networks, reinforcement learning (RL), and meta-heuristic algorithms in real-world environmental datasets [47], this study introduces a hybrid and innovative approach to TS. It integrates neural networks, metaheuristic algorithms, RL, and a greedy algorithm, proposing a novel method called the Q-learning-based Task scheduling scheme to enhance Energy consumption and QoS in IoT Environments (QTE-IoT). Combining RL algorithms with metaheuristic algorithms enhances optimization by enabling dynamic adaptation to changing conditions, improving exploration strategies, and allowing for learning from past experiences. This integration helps optimize decision-making policies, balances exploration and exploitation more effectively, and increases robustness in uncertain environments. Additionally, RL can facilitate real-time learning in dynamic scenarios and assist in multi-objective optimization by identifying trade-offs between competing objectives. Overall, this synergy leads to more informed search strategies, potentially reducing computational costs while improving solution quality in complex, high-dimensional problems. The primary goals of QTE-IoT are to enhance load balancing, energy efficiency, and QoS by minimizing response times and reducing deadline violations. Initially, tasks are classified into three categories: time-sensitive, security, and normal classes using a Multi-Layer Perceptron Artificial Neural Network (MLP-ANN). Subsequently, metaheuristic algorithms enhanced by RL are employed for TS within the fog and cloud layers. We used ARO algorithm [42] and improved it with Q-learning, designated as QARO. Then use proposed QARO to task scheduling in private and public clouds. Also we used AVA [43] and improved it by Q-learning and proposed Q-learning based AVA (QAVA) and used it to task scheduling in fog layer. The innovation of this paper relies on the combination of two algorithms, Q-learning and African Vultures and combination of two algorithms, Q-learning and ARO. Additionally, a monitoring agent is proposed to oversee resource workloads. If the workload exceeds a predetermined threshold, tasks are offloaded using the proposed greedy algorithm in the fog layer and rescheduled in the cloud layer.

1.5. Contributions

The contributions stated in this research are presented as follows:

- Task classification: This research introduces a novel approach for classifying tasks into time-sensitive, security, and normal categories. This classification is achieved using an MLP-ANN.
- Cloud layer scheduling: An enhanced ARO algorithm [42] integrated with Q-learning, designated as QARO, has been developed for TS in private and public cloud environments. This enhanced algorithm effectively optimizes TS, leading to improved performance.
- Fog layer scheduling: An enhanced AVA [43] combined with Q-learning (QAVA) has been designed specifically for TS in the fog layer. QAVA is tailored to the unique characteristics of fog computing environments and demonstrates superior performance in this context.
- Workload monitoring and optimization: Monitoring agents are deployed in the cloud gateway to oversee the workload of VMs at the cloud level, as well as in the broker to track the workload of

FNs at the fog level. In the cloud layer, when the workload exceeds a predefined threshold, tasks are rescheduled. Similarly, in the fog layer, tasks are offloaded using a proposed greedy algorithm. This strategy effectively improves load balancing and reduces response times, resulting in a more efficient and responsive system.

The rest of the paper is organized as follows: Section 2 provides an overview of existing literature. Section 3 showcases the process of modeling the system and articulates the problem formulation. Section 4 delineates the specifics of the suggested approach. Section 5 explains the assessment framework and compares the outcomes. Lastly, Section 6 presents final remarks and potential future research directions.

2. Related works

To review previous research, we categorize it into three main groups: studies that utilize RL, those that employ metaheuristic algorithms, and those that combine different approaches.

Due to the proper efficiency of RL algorithms, they have been used for TS in some studies. The authors in [30] focus on the problem of TS in cloud-based applications to minimize computational costs under resource and deadline constraints. They propose a clipped double deep Q-learning algorithm (CDDQLS) that uses an RL approach to leverage the target network and experience relay techniques. They aim to address the challenges posed by the increasing amount of data generated by cloud users and IoT devices, highlighting the importance of efficient resource allocation to ensure high QoS and adherence to SLAs.

Shahidani et al. [25] propose an RL fog scheduling algorithm to address congestion and delays in cloud data centers caused by IoT device requests. They highlight the importance of fog computing in offloading tasks to edge devices to reduce delays for real-time requests. The algorithm aims to optimize TS in fog computing environments, improving load balance and reducing response times compared to existing algorithms. This work contributes by leveraging RL techniques to enhance performance and reliability in edge-fog-cloud architectures. But they ignore energy consumption and task deadlines. Vijayasekaran and Duraipandian [31], discuss the integration of cloud computing and IoT to enhance data management processes and offer a new approach to resource scheduling for edge computing to integrate cloud and IoT. Their proposed approach incorporates hybrid data clustering and deep learning-based resource scheduling to simplify computations. The evaluation shows that this integrated approach outperforms traditional cloud IoT systems in terms of reduced latency, increased efficiency, and improved computational time.

Metaheuristics usually performs well in optimization problems and TS, so some works have used metaheuristic algorithms for TS. Kanbar and Faraj [35] introduced the RADISH (Region Aware DynamIcScHeduling) model to address the load balancing issue in cloud computing, which can impact computing resource performance. The model consists of five processes aimed at improving scheduling efficiency and reducing latency. The first process involves classifying incoming tasks as sensitive or non-sensitive using a task nature-based bi-class neural network. The second process focuses on scheduling the classified tasks using a multi-criteria-based improved moth flame optimization algorithm (QoS-aware AMFO). The third process involves load balancing through the SAC algorithm with potential field clustering. Task allocation is addressed in the fourth process by introducing a VM state-aware Hopcroft–Karp algorithm, which aims to reduce allocation latency, enhance QoS, and ultimately achieve high SLA and QoS in an IoT fog multi-cloud setting. However, they did not consider data security and energy consumption.

Authors in [36] discuss the challenges posed by the massive data volumes generated by the Industrial Internet of Things (IIoT) on cloud computing infrastructure. They introduced a novel approach called opposition-based simulated annealing particle swarm optimizer (OSAPSO) to address the premature convergence issue of particle swarm optimizer (PSO) when dealing with complex problems like task-scheduling-in-cloud-computing (TSCC). OSAPSO combines opposition-based learning, evolution strategy, simulated annealing, and swarm intelligence to enhance PSO's performance in handling high-dimensional tasks. The algorithm employs various techniques such as diversity guarantee through

opposition-based learning, novel evolutionary operators, and survivor probabilistic selection of simulated annealing to improve exploration capacity. Experimental results demonstrated that OSAPSO outperformed other competitors in terms of power consumption, monetary cost, service makespan, and system throughput in IIoT TS within heterogeneous cloud computing environments. However, it was not tested in the fog layer and was designed for TS only in the cloud layer. Furthermore, it does not consider propagation delay in sending tasks from the IoT device to the cloud layer.

Saif et al. [20] addressed the challenges in cloud-fog computing related to task assignment and resource allocation to optimize energy efficiency and service quality. They proposed a Multi-Objectives Grey Wolf Optimizer (MGWO) algorithm to minimize delay and energy consumption in fog computing environments. The algorithm was designed to be implemented in the fog broker, which is responsible for distributing tasks to appropriate resources based on task requirements. The authors conducted simulations to demonstrate the effectiveness of the MGWO algorithm in reducing delay and energy consumption compared to existing state-of-the-art algorithms. But they overlooked transmission costs, computing resources, and load balancing. Authors in [37] proposed a novel architecture for offloading tasks and allocating resources in fog computing environments. Their approach utilizes a subtask pool to facilitate task offloading. For resource allocation in the fog layer, they combined the Moth-Flame Optimization (MFO) algorithm with Opposition-based Learning (OBL), resulting in a hybrid algorithm known as OBLMFO. The OBLMFO model achieved promising results in reducing both task completion delays and energy consumption within the fog layer. However, they did not consider QoS parameters, and their work only considered the fog layer.

Table 1. An overview of relevant studies.

Ref.	Year	Algorithm Type	Environment	Name	Advantages	Disadvantages
[30]	2021	RL	Cloud	CDDQLS	Improves QoS	Energy consumption, deadline of tasks
[25]	2023	RL	Fog	RLFS	Improves delay, load balancing, and response time	Energy consumption, deadline of tasks
[31]	2022	RL	Edge	-	Improves latency, resource utilization, and computational time	High computational complexity
[35]	2022	Metaheuristic	Cloud	RADISH	Improves reducing latency, load balancing, and QoS	It doesn't consider data security and energy consumption.
[36]	2023	Metaheuristic	Cloud	OSAPSO	Improves power consumption, monetary cost, makespan, and throughput	It doesn't consider Propagation delay.
[20]	2023	Metaheuristic	cloud-fog	MGWO	Reduces delay and energy consumption	It doesn't consider transmission costs, computing resources, and load balancing.
[37]	2023	Metaheuristic	Fog	OBLMFO	Reduces task completion delay and energy consumption	It doesn't consider QoS parameters.
[38]	2022	Hybrid	Cloud	MOABCQ	Improves load balancing, makespan, cost, and resource utilization	It doesn't consider propagation delay.
[11]	2023	Hybrid	Fog	WCLA+GA	Improves response time, energy consumption, and meeting task deadlines	High computational complexity
[39]	2024	Hybrid	Fog	ETFC	Improves energy consumption, costs, response time, and meeting task deadlines	It focusses only on fog layer.

Some works have focused on hybridizing metaheuristic and RL methods for TS. Kruekaew et al. [38] proposed an independent TS approach in cloud computing using a multi-objective task scheduling optimization based on the Artificial Bee Colony (ABC) algorithm with a Q-learning algorithm, termed the MOABCQ method. This method aims to optimize scheduling and resource utilization, maximize VM

throughput, and achieve load balancing between VMs based on makespan, cost, and resource utilization. However, their work focused only on TS in cloud computing and did not consider propagation delay from IoT devices to VMs. Authors in [11] proposed a novel approach named WCLA+GA to optimize TS in fog environments. The primary objective was to minimize energy usage while ensuring the QoS criteria for IoT tasks, such as response time and deadline adherence. The WCLA+GA algorithm, an enhanced iteration of WCLA incorporating Genetic Algorithm (GA), was employed for TS in fog environments. The simulation results demonstrated that WCLA+GA surpassed alternative techniques to enhance response time, decrease energy usage, and meet task deadlines. However, the computational time was prolonged due to using an exact algorithm to address the NP-hard problem, limiting its scalability for solving large-scale issues. Pakmehr et al. [39] focused on optimizing TS in fog computing environments to minimize energy consumption, reduce costs, and enhance QoS by improving response time and meeting task deadlines for IoT tasks. The proposed Energy-efficient and Deadline-Aware TS in Fog Computing (ETFC) method utilizes the support vector machine to predict FN traffic, categorizes nodes into low-traffic and high-traffic groups, and employs different scheduling algorithms for each group. The low-traffic tasks are scheduled using an RL algorithm based on ICLA-SOA, while high-traffic tasks are scheduled with a metaheuristic algorithm based on Non-Dominated Sorting Genetic Algorithm (NSGA-III). However, they only focused on fog layer and did not have any idea about sending tasks to cloud layer. They did not consider data security either.

Table 1 shows the summary of related works. Typically, previous works have failed to consider all factors of energy consumption, load balancing, response time, and deadlines simultaneously. However, QTE-IoT tackles all these aspects concurrently. Previous studies have often focused on scheduling within a single layer, be it fog or cloud. In contrast, our approach takes into account both fog and cloud layers, enabling the algorithm to efficiently schedule tasks across both environments. We focus on TS in FCIoT and propose a hybrid method based on neural networks, metaheuristics, and RL to improve energy consumption, load balancing, and QoS requirements.

3. Proposed System Model

As depicted in Figure 1, FCIoT comprises three layers. The IoT layer encompasses a wide area of diverse IoT devices spread across different geographical locations. These devices typically generate time-sensitive data, necessitating real-time processing. Also, some data includes sensitive information, such as account information and passwords. The fog layer consists of dispersed FNs and fog controllers known as brokers. These brokers manage fog resources, and tasks are orchestrated by a TS algorithm within them. FNs are typically characterized by their computation resources, storage, and networking capabilities. The cloud layer is composed of multiple cloud data centers, both public and private, where several VMs can define on a physical machine using virtualization technology [48]. Table 2 presents the symbols and notations used in our formulation.

Table 2. The symbols and notations.

	Symbol	Describe
IoT	$\boldsymbol{T} = \{T_1, T_2, \ldots, T_n\}$	Set of n tasks.
	T_k	Task k with features: - T^s_k: Task size (MI) - T^d_k: Deadline (ms) - T^{in}_k: Input size (KB) - T^{out}_k: Output size (KB)
	$\boldsymbol{ST}=\{ST_1, ST_2, \ldots, ST_n\}$	Set of n security tasks.
	$\boldsymbol{TST}=\{TST_1, TST_2, \ldots, TST_n\}$	Set of n time sensitive tasks.
	$\boldsymbol{NT}=\{NT_1, NT_2, \ldots, NT_n\}$	Set of n normal tasks.
Fog	R_{Fq}	Fog resource q.
Cloud	R_{PCo}	Private cloud resource o.
	R_{Cp}	Public cloud resource p.
FCIoT	e_{ij}	Link between resources i and j with features:

		- e^{P}_{ij}: Propagation delay of link e_{ij} (ms). - e^{b}_{ij}: Bandwidth of link e_{ij} (Mbps).
	$G = (R, L)$	Network graph where R is the set of resources and L is the set of links.
	$\boldsymbol{L} = \{e_{ij} \mid Ri \leftrightarrow Rj\ i,j \in 1,2,\ldots,m,\ i \neq j\}$	Set of connecting links between resources.
	M	Individual fog or cloud resources.
	$\boldsymbol{R} = \{R_{F1}, R_{F2}, \ldots, R_{Fq}, R_{PC1}, R_{PC2}, \ldots, R_{PCo}, R_{C1}, R_{C2}, \ldots, R_{Cp}\}$	A collection of FCIoT resources.
	R_i	Resource I with features: R^{C}_{i}: CPU processing capability (MIPS) of resource i. R_i^{act}: Active-mode energy consumption (joules) of resource i. R_i^{inact}: Inactive-mode energy consumption (joules) of resource i.
QARO	$\mathcal{BU}_{ir}(t)$	The burrow selected by the rabbit in the hiding phase.
	C	Vector used to select individuals during the search process.
	D	The burrows created in the rabbit's vicinity.
	Dim	Dimensions of the search space.
	EA	The energy factor of rabbit.
	g_1, g_2, g_3, g_4, g_5	Parameter associated with the normal distribution function.
	H	The hiding function.
	K	Number of burrows generated by a rabbit in each iteration.
	p_v	The probability of moving vulture to either the first- or second-best vulture.
	$Randperm$	Random permutation function in the range from 1 to the problem dimension.
	RL	Parameter representing the running distance.
	α	The random variable in (0,1).
	ρ	Operator simulating rabbit locomotion during the foraging phase.
	$\vec{\Delta}_i(t+1)$	Updated ith candidate at time $t + 1$.
MLP-ANN	b_u^{ly}	Bias term for neuron u in layer ly.
	$data$	Input values vector from the prior layer.
	Ly	Layer index.
	P	True label (one-hot encoded) for class ly.
	U	Neuron index
	$w_u^{ly^T}$	Transposed weight vector of neuron u in layer ly.
	Wsz_u^{ly}	Weighted sum output of neuron u in layer ly.
Q-Learning	$Q(s_t,a_t)$	Q-value for taking action at in state st .
	a_t	Action taken at time step t.
	Rt	Reward received after taking action a_t in state s_t.
	s_t	Current state at time step t.
	B	Learning rate.
	$\boldsymbol{\Gamma}$	Discount factor ($0 < \gamma < 1$).
QAVA	$A1, A2$	Attraction forces toward BestVul1 and BestVul2.
	$B(v)$	Selected best vulture for the v-th solution.
	$BestVul_1$	Best solution in the first group.
	$BestVul_2$	Second-best solution in the second group.
	$D(t)$	Distance between the best and current vulture.
	$D(v)$	Distance between the current vulture and the selected best vulture.
	Dim	the dimensionality of the problem
	Fu	Adaptive coefficient balancing exploration/exploitation.
	$Hrand$	Random number ∈ [-2, 2].
	$Infp$	Parameter influencing hunger/satiation.
	J	Random coefficient (typically ∈1).
	$Levy(dim)$	Levy flight distribution for aggressive jumps.
	$Pos(v)$	Position of the v-th vulture.
	$Prob_1, Prob_2$	Probabilities (parameters) determining attraction to $BestVul_1$ or $BestVul_2$.
	$rand_1, rand_2, rand_3, rand_4, rand_5, rand_6, rand_7$	Random actual numbers actual that ranges from 0 to 1.
	$S1, S2$	Spiral motion components for rotational movement.
	ub, lb	Upper and lower bounds of the search space.
	W	Constant controlling sine/cosine oscillations (default: 1).
	$\Gamma(\cdot)$	Gamma function (generalized factorial).
	Δ	Scaling factor derived from the gamma function (Γ).
	Θ	Constant set to 1.5 (controls tail heaviness in Levy distribution).
Decision variables	s_k	Binary variable: 1 if T_k meets its deadline.
	X	The task assignment matrix.
	x_{ik}	Binary variable: Element of matrix X.
	y^k_{ij}	Binary variable: 1 if link e_{ij} is used for T_k .
	Z_{lK}	Binary variable: 1 if T_k has priority on link l .
Variables	$\Psi: T \rightarrow R$	Task allocation mapping function assigning tasks (T) to resources (R).
	A_i	The active time for R_i.
	$Cost$	Overall cost of services for the system.

d^{tx}_k	The transmission delay of T_k .
d^{prop}_k	The propagation delay of T_k.
d^{exe}_k	The execution time of T_k .
d^{wt}_k	The waiting delay of T_k in the queue.
DLI	The Degree of Load Imbalance.
E_i	The energy consumption of R_i.
E_{Total}	The system's overall energy consumption.
l_i	The inactive time for R_i.
Iter	Current iteration value (QAVA, QARO).
The load of resource Ri.	Ld_{Ri}
LI_{Avq}	The average load imbalance.
LI_{Worst}	The worst-case load imbalance.
M	The system's makespan.
$max(Ld_R)$	The maximum load among all resources.
MaxIter	Total number of iterations (QAVA, QARO).
Pop	The population size (QAVA, QARO).
The minimum load of all resources.	$min(Ld_R)$
N_s	The number of tasks meeting their deadlines.
RT_k	Response time of T_k.
V_k	The deadline violation time for T_k.
V^{Total}	The total deadline violation time of system.
w1,w2, w3, w4	Weights of cost.

FCIoT is a network consisting of various interconnected resources – VMs and FNs – with a mesh topology within the fog or cloud layer. This network is represented as a graph $G = (R, L)$, where $\boldsymbol{R} = \{R_{F1}, R_{F2}, \ldots, R_{Fq}, R_{PC1}, R_{PC2}, \ldots, R_{PCo}, R_{C1}, R_{C2}, \ldots, R_{Cp}\}$ represents a collection of m individual fog or cloud resources. This set includes q Fog resources (R_{Fq}), o private cloud resources (R_{PCo}), and p public cloud resources (R_{Cp}). The set $\boldsymbol{L} = \{e_{ij} | R_i \leftrightarrow R_j , i,j \in 1,2,\ldots,m, i \neq j)\}$ denotes the set of communication links that exist between resource nodes R_i and R_j within the system. The resources are characterized by:

- The CPU's processing capability, measured in millions of instructions per second (MIPS), is represented by R^C_i.
- Energy consumption in the active mode measured in joules and referred to as R^{act}_i.
- Energy consumption in the inactive state measured in joules and denoted by R^{inact}_i.

Furthermore, each link $e_{ij} \in \boldsymbol{L}$ has two primary attributes:

- Propagation delay in milliseconds (ms) labeled as e^P_{ij}.
- Bandwidth measured in megabits per second (Mbps) and denoted by e^b_{ij}.

$\boldsymbol{T} = \{T_1, T_2, \ldots, T_n\}$ represents a set of n autonomous tasks from IoT devices. Each task k (T_k) *for k=1,2,..n* is characterized by four features described as $T_k = \{T^s_k, T^d_k, T^{in}_k, T^{out}_k\}$, where T^s_k indicates the task size in million instructions (MI), primarily reflecting the computational workload or resource demands of task k; T^d_k signifies the deadline in ms, a crucial factor in TS to ensure timely completion; T^{in}_k represents the input file size in Kilo Bytes (KB), denoting the data quantity required by a task for processing; and T^{out}_k denotes the output file size in KB, representing the data generated from task execution.

The input and output file sizes significantly impact task execution efficiency and performance. As per [13] and [49], it is estimated that these parameter values are known for the broker or can be calculated by smart gateways or IoT devices. IoT tasks are typically presumed to be independent, allowing for parallel and distributed computation within the fog or cloud layer. We define constraints as follows:

Constraint (1) dictates that each task can be assigned to only one resource at a time and cannot be assigned to multiple resources simultaneously. But resources have buffers and may have more than one task assigned to them simultaneously. The binary variable x_{ik} determines whether task k (T_k) is executed on resource i (R_i) as per Eq. (4). Eq. (2) represents a crucial constraint wherein RT_k signifies the response time of task T_k with a deadline d, underscoring the significance of meeting task deadlines (T_k^d). Successful meeting of task deadlines is indicated when RT_k is lesser than T_k^d. RT_k is computed using Eq. (12). The binary decision variables $x_{ik}. y^k_{ij}. Z_{lk}.$ and S_k are detailed in Section 3.1. Constraint Eq. (3) stipulates that

these variables have values in the range {0,1}. y^k_{ij} addresses link utilization per Eq. (5), Z_{lk} relates to task priority according to Eq. (6), and S_k pertains to deadline adherence based on Eq. (7). In circumstances where meeting task deadlines becomes impractical, tasks may need to be declined, deferred, or attempted despite deadline violations, as explored in prior studies [39, 49]. Here, the latter approach is adopted, wherein efforts are made to minimize the duration of deadline violations should they occur.

$$\sum_{i=1}^{m} x_{ik} = 1 \qquad \forall\, k \in \{1, \dots, n\}, \forall\, i \in \{1,2, \dots, m\} \tag{1}$$

$$RT_k \leq T_k^d \qquad \forall\, k \in \{1, \dots, n\} \tag{2}$$

$$x_{ik}, y_{ij}^k, Z_{lk}, S_k \in \{0.1\} \qquad \forall i, j \in \{1,2, \dots, m\}, \forall\, k, l \in \{1, \dots, n\} \tag{3}$$

The following assumptions are essential considerations in addressing the TS problem:

1. Each resource (VM/ FN) is capable of handling one task at a time.
2. Tasks in progress within a resource cannot be transferred to another node mid-execution.
3. Resources are assumed to be stable without experiencing failures or downtime during the TS process.
4. Communication channels maintain reliability without link failures, ensuring task integrity during data transfer.
5. IoT devices are aware of the surrounding resources for effective task allocation and execution coordination.

These stipulated assumptions and constraints serve as foundational factors acknowledged prior to problem analysis. They are accepted as inherent conditions guiding subsequent analysis and conclusions without necessitating explicit validation in the research context.

3.1. Decision variables

The task assignment matrix, denoted as X, is a binary matrix with dimensions' n × m. Each element $x_{i\times k}$ in the matrix takes a value of either 0 or 1, indicating whether T_k is assigned to resource R_i. This matrix representation allows for a concise and efficient method of specifying task assignments [49].

$$x_{ik} = \begin{cases} 1 & if\ T_k\ is\ assigned\ to\ R_i \\ 0 & Otherwise \end{cases} \tag{4}$$

For each task T_k, a binary variable $y^k_{ij} \in \{0, 1\}$ is defined, where R_i and R_j and $e_{ij} \in L$. This variable equals 1 if the link e_{ij} (connecting resources R_i and R_j) is selected for routing task T_k, as governed by Eq. (5) [49]. The variable y^k_{ij} contributes to the response time calculation by incorporating the propagation delay and transmission delay of the chosen links.

$$y_{ij}^k = \begin{cases} 1 & if\ link\ e_{ij}\ is\ chosen\ for\ routing\ T_K \\ 0 & Otherwise \end{cases} \tag{5}$$

A two-state variable $z_{kl} \in \{0, 1\}$ is used to compare the priorities of two tasks, T_k and T_l. If task T_l has a higher priority than task T_k, then z_{kl} is set to 1; otherwise, it is set to 0. Eq. (6) formalizes the relationship between task priorities and their deadlines [49].

$$Z_{kl} = \begin{cases} 1 & if\ T_l^d < T_k^d \\ 0 & Otherwise \end{cases} \tag{6}$$

A two-state variable $S_k \in \{0, 1\}$ is used to indicate whether the deadline for completing task T_k has been met [49].

$$S_k = \begin{cases} 1 & if\ RT_k \leq T_k^d \\ 0 & Otherwise \end{cases} \tag{7}$$

3.2. Parameters

The challenge at hand involves the efficient allocation of *n* tasks among *m* resources, denoted by $\Psi : \boldsymbol{T} \rightarrow \boldsymbol{R}$, to optimize total energy consumption while meeting task deadlines. Thus, we formally define the parameters in this section.

3.2.1. Response Time

For each task T_k in the FCIoT, the response time encompasses four components: propagation delay($d^{prop}{}_k$), transmission delay ($d^{tx}{}_k$), execution time ($d^{exe}{}_k$), and queue waiting time ($d^{wt}{}_k$). $d^{prop}{}_k$ refers to the time a flow or data takes to traverse a communication channel from one point to another. It is influenced by factors like the medium, signal speed, and distance. $d^{prop}{}_k$ for task T_k is calculated by Eq. (8) [49]. This delay is multiplied by 2 to account for the round-trip communication.

$$d_k^{prop} = \sum_{e_{ij}} (2 \times e_{ij}^p \times y_{ij}^k) \tag{8}$$

$d^{tx}{}_k$ is the time required to transmit all bits of task T_k over the output link. It is determined by the task size ($T^{in}{}_k$ and $T^{out}{}_k$.) and the link bandwidth. The transmission delay for task T_k in a multi-hub network is calculated using Eq. (9) [49].

$$d_k^{tx} = \sum_{e_{ij}} \frac{T_k^{in} + T_k^{out}}{e_{ij}^b} \times y_{ij}^k \tag{9}$$

CPU's execution time ($d^{exe}{}_k$) is the time required to execute task T_k on the allocated processor. It is calculated by dividing the task size by the processing power of the processor, as shown in Eq. (10) [49].

$$d_k^{exe} = \sum_{i \in R} \frac{T_k^S}{R_i^c} \times x_{ik} \tag{10}$$

Each FN or VM as a resource can only handle one task at a time, and tasks cannot be preempted or shared. Therefore, a task assigned to a resource must wait in a queue until higher-priority tasks are completed. The waiting time for task T_k is calculated using Eq. (11) [49].

$$d_k^{wt} = \sum_{l \in T} \sum_{i \in F} (d_l^{exe} \times Z_{kl} \times x_{ik} \times y_{il}) \tag{11}$$

Finally, the response time to T_k (RT_k) is calculated by Eq.(12) [49].

$$RT_k = d_k^{prop} + d_k^{tx} + d_k^{exe} + d_k^{wt} \tag{12}$$

3.2.2. Tasks Deadline Compliance

To assess the effectiveness of TS, it's crucial to analyze the total instances of deadline breaches. This requires calculating the percentage of IoT tasks that are completed on time relative to the total time of violations for a set of n within a given timeframe. The number of tasks (Ns) that meet their predetermined deadlines is used to calculate the deadline meet ratio, as shown in Eq. (13) [49]. The deadline violation time for each task T_k can be calculated by Eq. (14) [49]. This metric quantifies the extent to which a task's execution exceeds its deadline. Eq. (15) calculates the total deadline violation time (V_{total}), which represents the cumulative amount of time by which all tasks exceed their deadlines [49]. This metric provides an overall measure of the system's ability to meet task deadlines.

$$N_S = \sum_{k \in T} S_k \tag{13}$$

$$V_k = \max(0, RT_k - T_k^d) \tag{14}$$

$$V_{Total} = \sum_{k \in T} V_k \tag{15}$$

3.2.3. Energy Consumption

Our focus is on the energy resources consume to execute all n tasks. The energy expenditure of a resource $R_i \in \boldsymbol{R}$ during this period is calculated by summing the energy used in both active and inactive modes of resources.

The active time for R_i (A_i) is determined by the time required to process all allocated tasks, as given by Eq. (16). To calculate the inactive time for R_i (l_i), we first determine the system's makespan (M), which is the maximum execution time of R_i. The makespan is calculated using Eq. (17), and the inactive time of R_i is then calculated using Eq. (18) [11].

$$A_i = \sum_{k \in T}(d_k^{exe} \times x_{ik}) \quad (16)$$

$$M=\max(A_i) \quad (17)$$

$$l_i=(M-A_i) \quad (18)$$

The energy consumption of R_i is estimated based on its energy consumption specifications and the duration spent in each mode. Eq. (19) provides the formula for calculating the energy usage of R_i [49].

$$E_i = (A_i \times R_i^{act} + l_i \times R_i^{inact}) \quad (19)$$

The system's overall energy consumption is obtained by summing the total energy consumption of each R_i, as shown in Eq. (20) [11].

$$E_{Total} = \sum_{i \in R} E_i \quad (20)$$

3.2.4. Load Imbalance

Load imbalance refers to the uneven distribution of tasks across resources in a network. When tasks are assigned to the nearest FN or VM, some resources may become overloaded while others remain underutilized. The load imbalance between two FNs or two VMs is calculated as the difference in the number of tasks assigned to them. The load of resource i is denoted as Ld_{Ri}, and the load of resource j is denoted as Ld_{Rj}. The maximum load among all resources is represented by $max(Ld_R)$, and the minimum load is represented by $min(Ld_R)$. The worst-case load imbalance (LI_{Worst}) metric identifies the maximum load imbalance between any two resources by Eq. (21). The average load imbalance (LI_{Avg}) metric represents the average load imbalance across all resources in Eq. (22). Eq. (23) calculates the Degree of Load Imbalance (DLI) by dividing the worst-case load imbalance by the average load imbalance. This metric represents the percentage extent of load imbalance in the system.

$$LI_{Worst}=\max(Ld_R)\_\min(Ld_R) \quad (21)$$

$$LI_{Avg}=\frac{1}{|R|.(|R|-1)}\sum_{Ri,Rj \in R} |Ld_{Ri} - Ld_{Rj}| \quad (22)$$

$$DLI=\frac{LI_{Worst}}{LI_{Avg}} \times 100 \quad (23)$$

3.2.5. Cost

Finally, the cost is defined in Eq. (24). To normalize the cost, which involves parameters with different units of measurement, each metric is divided by its respective maximum value. These maximum values, determined by the network operator, represent the physical limitations of the network devices involved.

$$cost = w1(\frac{RT_k}{\max(RT_k)}) + w2(\frac{V_{Total}}{\max(V_{Total})}) + w3(\frac{E_{Total}}{\max(E_{Total})}) + w4(\frac{LI_{Avg}}{\max(LI_{Avg})}) \quad (24)$$

with w1–w4 being the balance coefficients among response time, deadline violation time, energy consumption and load imbalance parameters.

4. Proposed Method

The proposed method (QTE-IoT) uses the neural network and greedy methods. In addition, a combination of metaheuristic and RL algorithms has been proposed. Figure 2 displays a flowchart of the proposed method. First, the tasks are divided into three groups using the MLP-ANN. The security tasks are sent to the private cloud, where they are executed to maintain security. Also, the tasks for which there is plenty of time to be executed in the cloud layer and which contain no security data are sent to the public cloud for execution. The tasks are scheduled in public and private clouds using a proposed ARO algorithm improved by Q-learning (QARO).

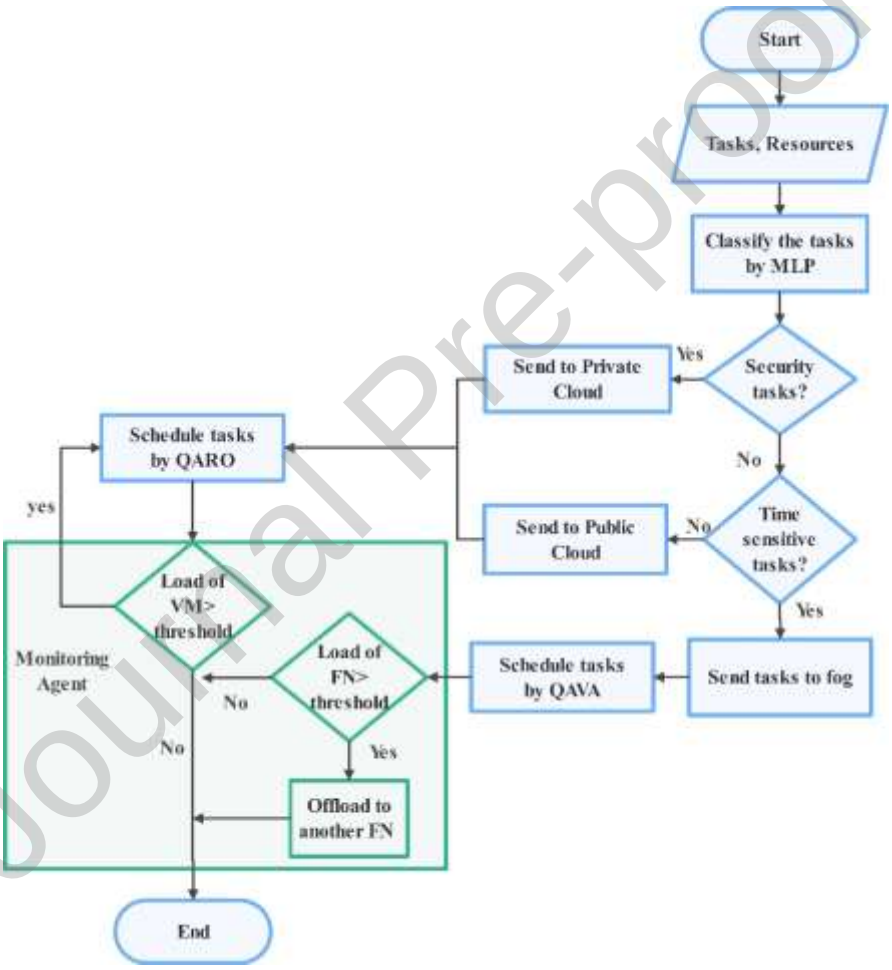


Fig. 2. The flowchart of QTE-IoT.

The time-sensitive tasks that must be executed promptly are sent to the fog, where they are scheduled by the proposed African Vulture algorithm improved by Q-learning (QAVA). QTE-IoT proposed QARO and QAVA, Q-Learning enhances the exploration and exploitation stages of these metaheuristic algorithms. Additionally, metaheuristic algorithms retain information about the search space during iterations, which is essential for Q-Learning. Consequently, this research introduces two hybrid algorithms that merge reinforcement learning with metaheuristic techniques, leveraging the strengths of both while addressing and reducing their weaknesses. The monitoring agent has been defined in the proposed method, which is

responsible for monitoring the loads of VMs in the cloud layer and the load of FNs in fog layer. If the load of VMs exceed a predefined threshold, TS with QARO is repeated. In order to take care of the task deadlines and do them promptly, the monitoring agent supervises loads of FNs in the fog layer, too. If it exceeds a given threshold, the tasks are offloaded to another FN based on the proposed greedy algorithm.

4.1. Fitness Function

The QTE-IoT aims to minimize the fitness function, represented by cost as defined in Eq. (24), by simultaneously addressing multiple optimization objectives. These objectives include optimizing task deadlines (V_{Total}) to ensure timely completion, minimizing response time (RT_k) to reduce the time taken for task execution, decreasing energy consumption (E_{Total}) to enhance system efficiency, and balancing the workload across resources to prevent both overutilization and underutilization, thereby reducing load imbalance. By achieving these objectives, the proposed QARO and proposed QAVA algorithms aim to enhance the overall performance and efficiency of task scheduling in cloud and fog layers, respectively.

4.2. Task Classifications by MLP-ANN

To classify tasks in the QTE-IoT framework, we employed a Multi-Layer Perceptron Artificial Neural Network (MLP-ANN) [50]. The MLP-ANN was selected for its ability to perform multiple classification tasks concurrently, which effectively minimizes latency in task classification. An MLP-ANN is a type of feedforward artificial neural network that serves as a foundational architecture for deep learning or deep neural networks (DNNs). One advantage of deep learning methods over traditional machine learning is the elimination of a separate feature extraction step before classification in deep learning. This reduces reliance on human resources and fosters a more automated setting [51, 52].

The MLP-ANN architecture is commonly used in neural networks, featuring an input layer for input features, an output layer for classes or labels, and one or more hidden layers in between. MLP-ANN forms a fully connected structure of these layers, with each hidden layer capable of accommodating varying numbers of neurons [2]. It is typically employed for supervised classification tasks, where the model learns associations between input features and output labels through training on a designated dataset. Model accuracy is assessed by testing it on a separate test dataset. The learning process of MLP-ANN involves two main functions for classification: forward propagation and backward propagation.

Forward propagation denotes the process of inputting data into the MLP-ANN's initial layer. This initiation enables data propagation through subsequent hidden layers. Within each neuron, the input data undergoes multiplication by a weight factor and the addition of a bias term. Consequently, each neuron (excluding those in the input layer) evaluates the weighted sum *Wsz* using Eq. (25):

$$Wsz_u^{ly} = w_u^{ly^T} . data + b_u^{ly} \qquad (25)$$

in which *w*, *data*, *b*, and *Wsz* correspond to the weight matrix of the preceding layer's neurons, the input values vector of the prior layer's neurons, the bias vector of the previous layer, and the resultant output, respectively. Additionally, *ly* represents the layer index, whereas *u* signifies the neuron's position within that layer. Upon *Wsz* computation, the output is input to an activation function to constrain the numerical values within a specific range.

$$L(Wsz, p) = -\sum_{u=1}^{c}(p_{ly} \times \log(Wsz_c)) \qquad (26)$$

These actions persist throughout the neural network until reaching the output layer. Subsequently, the forecasted label is juxtaposed with the actual label to compute the related loss and gauge the effectiveness of the weights and biases, facilitated by categorical cross-entropy loss. For classification tasks, cross-entropy is a popular choice due to its effectiveness in quantifying the performance of a classification model.

Categorical cross-entropy is essential in multi-class classification, where a model must classify an instance into one of several classes. The categorical cross-entropy formula is expressed by Eq. (26).
The categorical cross-entropy loss, denoted as *L(Wsz, p)*, measures the discrepancy between the true labels *Wsz* and the predicted probabilities *p* for each class. Here, p_{ly} represents the predicted probability of class *ly* (either 0 or 1 in a one-hot encoded target vector), The number of classes is denoted by *c*. This loss function utilizes the logarithm to heavily penalize incorrect predictions. A high loss value indicates confident but inaccurate predictions, while a low loss value signifies predictions that closely align with the true labels. This encourages the model to make accurate and confident predictions.

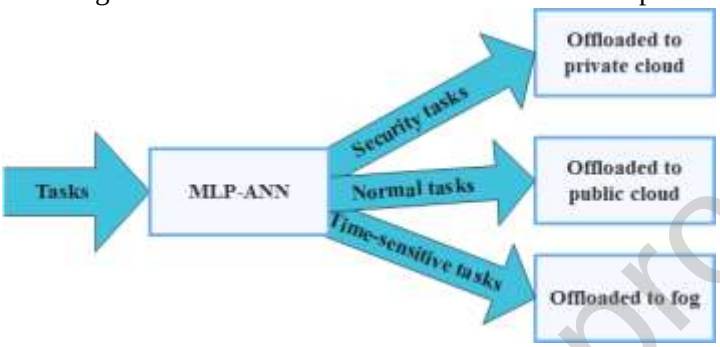


Fig. 3. The classification of tasks by MLP-ANN.

In this work, an MLP-ANN was utilized. The network architecture comprised an input layer with 18 neurons, representing the features of tasks, followed by a single hidden layer with 10 neurons. The output layer consisted of two neurons, capable of representing three classes: time-sensitive tasks, security tasks, and normal tasks (as depicted in Fig. 3). A single hidden layer was chosen as a compromise between training efficiency and accuracy, achieving near-perfect accuracy (close to 100%) during the testing phase. Further tuning, through adjustments to the number of neurons in the hidden layer, ultimately validated the initial choice of 10 neurons as optimal. The classification process relies on several key features, including login credentials, resource requirements (computational power, memory usage, network bandwidth), authentication and authorization, and QoS metrics such as latency and availability. Let the incoming tasks be denoted as $\boldsymbol{T} = \{T_1, T_2, \ldots, T_n\}$, which are then organized into these three sets $\boldsymbol{ST}=\{ST_1, ST_2, \ldots, ST_n\}$ for security tasks, $\boldsymbol{TST}=\{TST_1, TST_2, \ldots, TST_n\}$ for time sensitive tasks and, $\boldsymbol{NT}=\{NT_1, NT_2, \ldots, NT_n\}$ for normal tasks. After the tasks are classified by MLP-ANN, the security tasks are offloaded to the private cloud, the time-sensitive tasks are offloaded to the fog, and the normal tasks are offloaded to the public cloud.

4.3. Task Scheduling in Cloud Layer

Previous research has successfully used optimization techniques to address various engineering scheduling problems. These studies indicate the potential to develop even better optimization methods tailored to specific challenges. However, as the No-Free-Lunch theorem emphasizes, no single optimization algorithm can solve all problems equally well. Recognizing this, we propose a novel optimization approach called Q-learning based Artificial Rabbits Optimization (QARO). QARO builds upon the successful Artificial Rabbits Optimization (ARO) algorithm, known for its effectiveness in diverse mathematical and engineering contexts. ARO's strengths, including its ability to effectively explore the solution space and efficiently converge to optimal solutions, make it an ideal foundation for our proposed QARO algorithm.
The QTE-IoT for scheduling ***ST*** in private and ***NT*** in public cloud environments utilizes an enhanced version of ARO. To enhance ARO's capabilities, we integrated it with the Q-learning algorithm, resulting in a new approach called QARO. QARO assists in boosting and decision-making to optimize *TS*.
ARO [42], a bio-inspired metaheuristic algorithm, emulates the survival strategies of rabbits, including detour foraging and random hiding, to solve optimization problems. By balancing exploration and exploitation, ARO efficiently explores the search space, escapes local optima, and enhances convergence. In the context of scheduling, ARO effectively schedules tasks by mimicking the foraging behavior of

rabbits. Each rabbit (representing a task) searches for food (optimal resource allocation) in distant locations (alternative resource options) rather than consuming nearby food (local optima). This approach improves task distribution, resource utilization, and overall system performance. This foraging behavior is mathematically expressed in the subsequent equations, as detailed in [42]:

$$\vec{\Delta}_i(t+1) = \vec{z}_j(t) + \rho \cdot \left(\vec{z}_i(t) - \vec{z}_j(t)\right) + round\left(0.5 \cdot (0.05 + g_1)\right) \cdot n_1$$

$$i, j = 1,2 \dots, RabPop \text{ and } i \neq j \tag{27}$$

$$\rho = RL \cdot c \tag{28}$$

$$RL = (e - e^{\left(\frac{1-t}{MaxIter}\right)^2}) \cdot \sin(2\pi g_2) \tag{29}$$

$$\text{c(K)}=\begin{cases} 1 & if\ K = h(u) \\ 0 & else \end{cases}, K = 1 \dots, dim, u = 1,2 \dots, \lceil g_3 \cdot dim \rceil \tag{30}$$

$$h=randperm(dim) \tag{31}$$

$$n_1 \sim N(0,1) \tag{32}$$

where $\vec{z}_i(t)$ represents the current position of the *i*th candidate at time t (defined by Equation (44) in our QTE-IoT method), $\vec{\Delta}_i(t+1)$ updated *i*th candidate at time $t + 1$, *Pop* is the population size, *MaxIter* is the total number of iterations, *dim* denotes the problem dimensions, *round* shows rounding to the nearest integer, *randperm* is random permutation function in the range from 1 to the problem dimension, and *RL* is the running length during foraging. The parameters g_1, g_2, and g_3 are associated with the normal distribution function and uniform random numbers within the [0, 1] interval, respectively. Perturbation, as defined in Eq. (27), helps prevent stagnation at local optima and facilitates thorough exploration of the search space. The parameter *RL*, representing the running distance, enables longer traversals during the initial iterations for exploration and shorter ones later on for exploitation. The operator ρ simulates rabbit locomotion, while the vector *c* assists in selecting individuals during the search process. This foraging phase enhances the ARO algorithm's exploration and global search capabilities. During exploitation, rabbits adopt a random hiding strategy to evade predators. They create multiple hideouts near their burrows to deceive potential threats. In each ARO iteration, a rabbit generates a specific number of burrows across all dimensions and randomly selects one to hide in.

$$\overrightarrow{BU_{ij}}(t) = \overrightarrow{z_i}(t) + H \cdot h \cdot \overrightarrow{z_i}(t), i = 1,2, \dots, RabPop\ , j = 1,2, \dots, dim \tag{33}$$

$$\text{H}=\frac{1-t+MaxIter}{MaxIter}\ g_4 \tag{34}$$

$$n_2 \sim N(0.1) \tag{35}$$

$$h(K) = \begin{cases} 1 & if\ K = j \\ 0 & else \end{cases}, K = 1, \dots, dim \tag{36}$$

Initially, burrows are established within a larger territory around the rabbit. As the number of iterations increases, the size of this surrounding area decreases. The random hiding approach is detailed in Eq. (37) to (39) [42], in which *H* represents the hiding function and *d* is the burrows created in the rabbit's vicinity.

$$\vec{\Delta}_i(t+1) = \vec{z}_j(t) + \rho \cdot \left(g_4 \cdot \overrightarrow{BU_{ir}}(t) - \vec{z}_j(t)\right)$$

$$i, j = 1,2, \dots, RabPop \tag{37}$$

$$h_r(K) = \begin{cases} 1 & if\ K = [g_5 \cdot d] \\ 0 & else \end{cases}, K = 1, \dots, dim \quad (38)$$

$$\overrightarrow{BU_{ir}}(t) = \overrightarrow{z_i}(t) + H \cdot h \cdot \overrightarrow{z_i}(t) \quad (39)$$

In the random hiding strategy, BU_{ir} (t) is the burrow selected by the rabbit in the hiding phase, and g_4, g_5 are random numbers within the range [0, 1]. The position of the ith rabbit is adjusted following either a detour foraging mode or a random hiding procedure.

$$\overrightarrow{z_s}(t+1) = \begin{cases} \overrightarrow{z_s}(t) & if\ f(\overrightarrow{z_s}(t) \le f(\vec{\Delta}_s(t+1)) \\ \vec{\Delta}_s(t+1) & else \end{cases} \quad (40)$$

In the continue, the fitness is checked and the movement is made. If the fitness of the sth rabbit's candidate surpasses the current position's fitness, the rabbit relocates from its current spot to the new candidate location specified by either Eq. (27) or Eq. (37). As the iterations advance, the rabbits' energy dwindles, facilitating the shift from an exploratory mode to an exploitative mode, as described by Eq. (41):

$$EA(t) = 4\left(1 - \frac{1}{MaxIter}\right)\ln(\frac{1}{\alpha}) \quad (41)$$

where α is the random variable in (0,1). If $EA\ (t)$ exceeds 1, the algorithm engages in a global search for the solution (exploration); if $EA\ (t) \le 1$, the algorithm focuses on a local search for the solution (exploitation).

4.3.1 Q-Learning Model in QARO

Q-learning is a value-based reinforcement learning algorithm well-suited for integration with metaheuristic optimization methods [53]. RL enables an agent to learn an optimal policy through trial-and-error interactions with a stochastic environment, modeled as a Markov decision process (MDP). The agent observes the environment's state, selects actions to influence future states, and receives rewards as feedback. The goal is to maximize the cumulative expected reward over time.

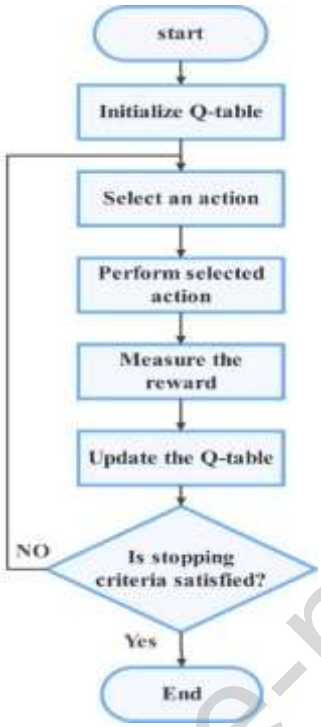


Fig. 4. The flowchart of Q-learning.

In Q-learning, the agent maintains a Q-table $Q(s,a)$ representing the expected utility of taking action a in state s. The Q-values are iteratively updated according to Eq. (42).

$$Q(s_t, a_t) = (1-\beta)Q(s_t, a_t) + \beta[r_t + \gamma max_{a+1} Q(s_{t+1}, a_{t+1})] \quad (42)$$

where β is the learning rate, γ $(0 < \gamma < 1)$ is the discount factor regulating future rewards, and r_t+1 is the reward obtained after taking action a_t in state s_t [38]. Figure 4 illustrates the flowchart of this process.

Figure 5 presents the flowchart of QARO, illustrating the interaction between the Q-learning agent and the metaheuristic optimization algorithm ARO within the QTE-IoT framework.

The Q-learning model is formally defined by State Space, Action space, reward function and Integration and Learning Process in QARO.

State Space (*S*):

Each state $s_i(t) \in S$ encodes the current condition of ARO's population and cloud computing environment at iteration t. It is represented by a vector of performance metrics by Eq. (43).

$$s_i(t) = [\frac{f_{best}}{\text{median}(P.fitness)}, \text{LI}_{worst}, \frac{iter}{\text{MaxIter}}] \quad (43)$$

where f_{best} is the best fitness value in the population, *median(P.fitness)* represents the median population fitness, LI_{worst} evaluates the load imbalance across cloud resources, and *iter*/*MaxIter* normalizes the current iteration number to reflect the progress of the optimization process. The fitness value used in these calculations is obtained from the multi-objective fitness function defined by Eq. (24).

Action Space (*A*):

The action space *A* consists of ARO search operators and comprises two types of actions. The first is exploration, referred to as detour foraging, which involves selecting a rabbit (solution) to explore new areas of the search space more globally. The second is exploitation, known as random hiding, where a burrow (hiding position) is selected to intensify the search locally around promising solutions. At each iteration and given state $s_i(t)$, the Q-learning agent selects an action $a_t \in$ {detour_foraging,random_hiding} guided by the policy derived from the Q-values.

Reward Function (r_t):

The reward quantifies the immediate improvement in task scheduling quality after executing action a_t and updating solution positions. It is computed based on the change in the fitness value, which integrates multiple objectives such as reductions in total cost, energy consumption, response time, deadline violations, and load imbalance. Formally, the reward expressed as the difference between the previous and current fitness values $r_t=f_{old}-f_{new}$, where a positive reward indicates an improvement in the scheduling solution. Positive rewards encourage beneficial exploratory or exploitative moves, while stagnant or detrimental changes yield lower or negative rewards. This feedback drives the Q-learning agent to adaptively guide the search towards more optimal task schedules.

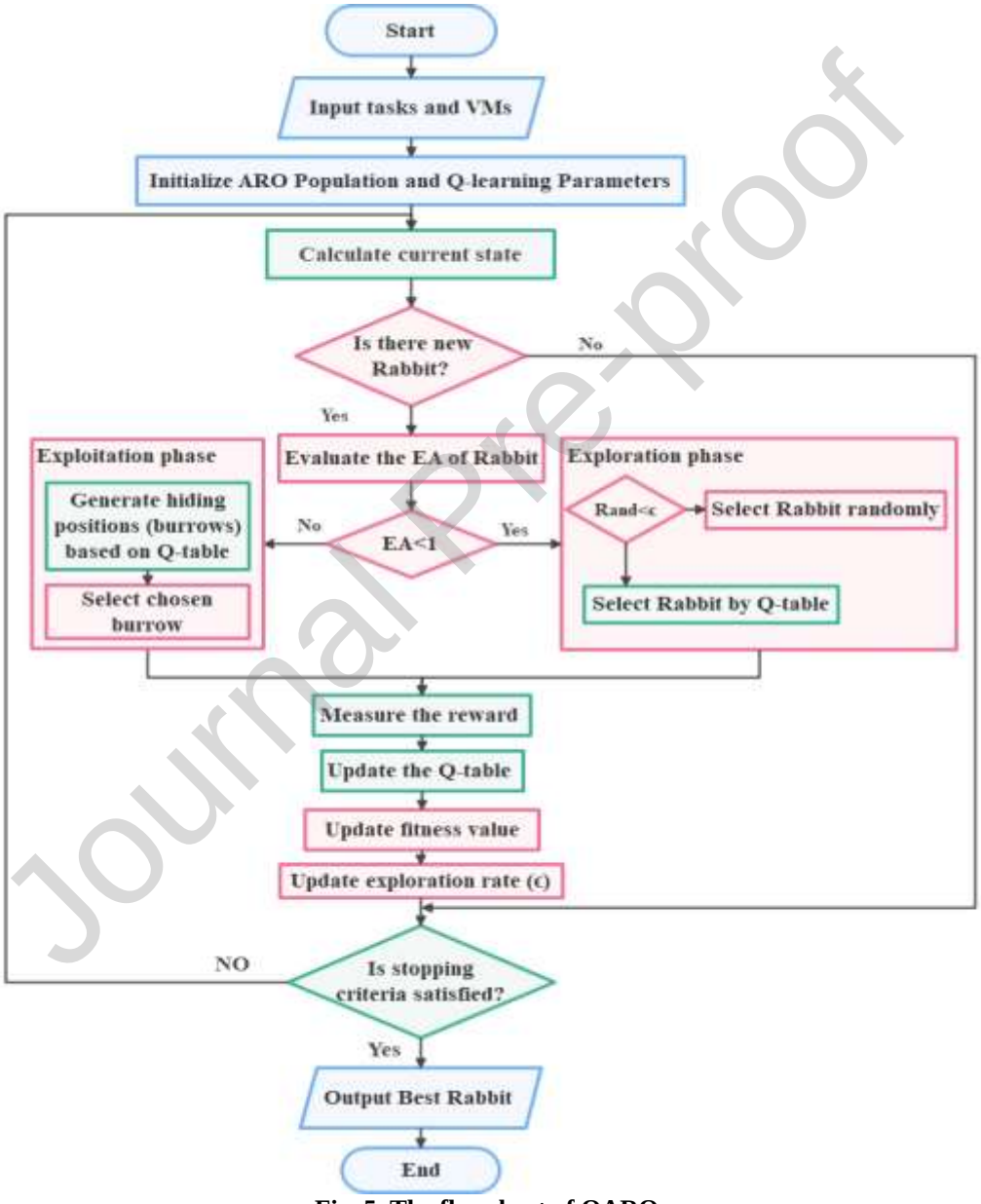


Fig. 5. The flowchart of QARO.

Integration and Learning Process:

At each iteration, Q-learning observes the current state derived from ARO's population metrics and selects an action to guide ARO's search operators. The search process updates rabbit positions and fitness

evaluations, which feedback as rewards, enabling temporal difference learning to update Q-values by Eq. (44).

$$Q_{t+1}(s_t, a_t) = Q(s_t, a_t) + \beta[r_{t+1} + \gamma max_{at}[Q_t(\delta(s_{t+1}, a_t), a_t)] - Q_t(s_t, a_t)] \quad (44)$$

Where $\max_a Q(s_{t+1}, a)$ denotes an approximation of the optimal future value. Additionally, the ARO's energy factor $EA(t)$ is leveraged to switch between exploration and exploitation dynamically, enhancing responsiveness to the environment. The exploration rate ε decays over time to balance exploration and exploitation adaptively.

Algorithm 1: Procedure of QARO

Input: Tasks' set (***ST***) or (***NT***), VMs' set of private cloud(R_{pc}) or public cloud (R_c)
Output: Best solution

```
Initialize
   Initialize ARO population Pop, Q-table Q with zeros, exploration rate ε, learning rate β, discount
factor γ, maximum number of iterations MaxIter, best_Rabbit ← NULL.
for iter in range (MaxIter):
{
        current_state ← calculate current state by Eq. (44);
        for each rabbit in Pop
                {
                 Evaluate the Energy Factor (EA) of rabbit by Eq. (41);
                 If (EA<1)    #Q-guided exploration
                #   Q-learning action selection
                        {
                           if random() < ε then
                              selected_rabbit ← random_choice (Pop);
                            Else
                              {
                              selected_rabbit ← argmax(Q[current_state, :]); # Q→ARO: Policy guidance
                              ρ ← calculate_step_size(selected_rabbit); # ARO Eq. (28)
                              }
                            new_position ← detour_foraging(selected_rabbit, ρ);  # ARO Eq. (29)
                           action_type ← EXPLORATION
                        }
                       Else  # Q-guided exploitation
                           {
                            burrows ← generate_hiding_positions(Q[current_state, :]); // ARO Eq. (29)
                            chosen_burrow ← select_burrow(burrows, Q[current_state, :]); // Q→ARO:
                            Value-based selection
                            new_position ← random_hiding(chosen_burrow); // ARO Eq. (27)
                            action_type ← EXPLOITATION;
                           }
                  # Reward calculation from ARO fitness
                  reward ← calculate reward #  ARO→Q: Multi-objective feedback
                  # Q-table update with ARO state transition
                  Q[current_state, action] ← calculate Q-value by Eq. (43);
                   update_rabbit_position(rabbit, new_position); # ARO Eq. (40)
                   update_fitness(rabbit); # Evaluate Eq. (24)
                  }
          best_Rabbit ← find_min_cost(P); # ARO→Q: New state basis
          ε ← ε × 0.98 // Exploration decay;
}
```

42	***return best_Rabbit;***

In proposed QARO, Q-learning is integrated with the ARO algorithm to dynamically balance exploration and exploitation phases during task scheduling in public and private clouds. A possible action in cloud computing involves scheduling the tasks in the VM, which allocates tasks in the VM and can be described in Algorithm 1. In the public cloud, the ***NT*** set is used as input, while the ***ST*** set serves as input in the private cloud for the algorithm. Initially, the algorithm initializes the rabbit population and Q-learning parameters. In each iteration, it calculates the current system state and, for each rabbit in the population, evaluates an energy factor to decide between exploration and exploitation. During exploration, Q-learning guides the selection of rabbits and their movements using a detour foraging strategy, while in exploitation, it selects promising hiding positions (burrows) and updates rabbit positions accordingly. Rewards based on multi-objective fitness evaluations are used to update the Q-table iteratively, enabling the agent to learn and adapt its action-selection policy. This process continues until convergence or the maximum iteration count is reached, with the algorithm returning the best rabbit solution that optimally maps tasks to cloud resources. The convergence of QARO to an optimal policy is theoretically supported by the Q-learning update rule combined with ARO's population-based search and adaptive strategy. This synergy improves convergence speed and solution quality in cloud-layer task scheduling.

The integration of Q-learning with ARO in QARO provides a robust framework for optimization problems. Theoretically, QARO ensures convergence to an optimal policy by leveraging Q-learning's temporal difference learning and ARO's population-based search. The Q-learning component guarantees policy improvement through the update rule Eq. (44).

This update rule, combined with ARO's adaptive exploration-exploitation strategy, ensures that QARO explores the solution space efficiently while converging to optimal solutions.

Moreover, the use of ARO's energy factor (EA) to switch between exploration and exploitation phases enhances the algorithm's adaptability and responsiveness to changing environmental conditions. This dual mechanism provides a theoretical foundation for QARO's superior performance over traditional optimization methods.

The set of ***ST*** schedules by QARO in private cloud and the set of ***NT*** schedules by QARO in public cloud. After scheduling security and normal tasks with QARO in the public and private clouds, the QTE-IoT monitors the load of VMs with the help of monitoring agents. If the VM’s load exceeds the threshold, the schedule is repeated with QARO.

4.4. Task Scheduling in Fog Layer

QTE-IoT schedules time-sensitive tasks in the fog with the proposed QAVA. The original AVA is a new algorithm that demonstrates impressive results. However, in AVA, the exploration phase is less effective than the exploitation phase during the solution search process. This imbalance diminishes the quality of the final output in AVA [54]. To address this issue, we propose combining AVA with Q-learning in this paper, enhancing its exploration capabilities while maintaining its strong exploitation performance. The Q-learning mechanism is used to find fast optimal solutions in AVA. Also, Q-learning can enhance the exploration-exploitation balance of AVA, leading to improved optimization performance. Inspired by African vultures’ social behavior and foraging patterns, AVA is a metaheuristic algorithm that employs two phases: exploration and exploitation.

The algorithm begins by generating an initial set of potential solutions. Each solution is then evaluated based on a fitness function, which measures its quality. The best solution is designated as the " $BestVul_1$" of the first group, and the second-best solution becomes the " $BestVul_2$" of the second group. The remaining solutions are then adjusted using Eq. (45) to move closer to these top performers.

$$B(v) = \begin{cases} BestVul_1 & if\ p_v = Prob_1 \\ BestVul_2 & if\ p_v = Prob_2 \end{cases} \quad (45)$$

In Eq. (45), the symbol p_v represents the probability that a given vulture (solution) will move closer to either the first or second best solution within its group during the optimization process. This probability controls the algorithm's balance between exploration and exploitation by determining the direction and strength of attraction toward the leading solutions. The value of p_v is computed based on two parameters, $Prob_1$ and $Prob_2$, which quantify the relative influence of the first and second best vultures, respectively. These parameters are set between 0 and 1, ensuring their sum equals 1.

In exploration phase, AVA adopts two methods as described in Eqs. (46) and (47):

Eq. (45): Roulette wheel selection to determine one of the two best vultures (B(v)), followed by the calculation of D(v) using Eq. (47). Subsequently, *Fu* is computed using Eq. (48), where Pos(v) represents the position of the current vulture vector (a random number between 0 and 1).

$$Pos(v+1) = B(v) - D(v) \times Fu \quad (46)$$

$$D(v) = |J \times B(v) - Pos(v)| \quad (47)$$

Eq. (48): Generation of a random number *hrand* ranging from -2 to 2 and then calculating *Fu* using sine and cosine functions. *Iter* denotes the current iteration value, *MaxIter* represents the total number of iterations, *w* is set to a constant value, *infp* is parameter influencing hunger/satiation and $rand_1$ is a random number between 0 and 1.

Eq. (49): Calculation of a new position for the vulture vector within the problem's upper (*ub*) and lower (*lb*) bounds using a random number $rand_2$ between 0 and 1.

$$Fu = (2 \times rand_1 + 1) \times infp\left(1 - \frac{Iter}{MaxIter}\right) + hrand\left(\sin^w \times \left(\frac{\pi}{2} \times \frac{Iter}{MaxIter}\right) + \cos\left(\frac{\pi}{2} \times \frac{Iter}{MaxIter}\right) - 1\right) \quad (48)$$

$$Pos(v+1) = B(v) - Fu + rand_2 \times ((ub - lb) \times rand_2 + lb) \quad (49)$$

For optimization operations in the exploitation phase, AVA utilizes four mechanisms:

I. Food Competition: Emulating the competition among vultures for food (Eq. (50)). D(v) is calculated using Eq. (51), *Fu* is calculated using Eq. (52), $rand_3$ is a random number between 0 and 1, and *D(t)* is computed using Eq. (50).

$$Pos(v+1) = D(v) \times (Fu + rand_3) - D(t) \quad (50)$$

$$D(t) = B(v) - Pos(v) \quad (51)$$

II. Vulture Rotation: Eq. (52) introduces $rand_4$ and $rand_5$ as random numbers between 0 and 1. It incorporates sine and cosine functions and calculates the new vulture position using Eq. (55).

$$S_1 = B(v) \times \left(\frac{rand_4 \times Pos(v)}{2\pi}\right) \times \cos\big(Pos(v)\big)$$

$$S_2 = B(v) \times \left(\frac{rand_5 \times Pos(v)}{2\pi}\right) \times \sin\big(Pos(v)\big) \quad (52)$$

$$Pos(v+1) = B(v) - (S_1 + S_2) \quad (53)$$

III. Accumulation of Different Vultures on Food Source: Eq. (54) and (55) model the behavior of multiple vultures congregating around a food source. $BestVul_1$ and $BestVul_2$ represent the top two vultures in the population. The new vulture position is computed using Eq. (56).

$$MA_1 = BestVul_1(v) - \frac{BestVul_1(v) \times Pos(v)}{BestVul_1(v) - Pos(v)^2} \times Fu$$

$$MA_2 = BestVul_2(v) - \frac{BestVul_2(v) \times Pos(v)}{BestVul_2(v) - Pos(v)^2} \times Fu \quad (54)$$

$$Pos(v+1) = \frac{MA_1 + MA_2}{2} \quad (55)$$

I. Aggressive Food Competition: Eq. (56) simulates aggressive competition among vultures for food. *Levy(dim)* represents a random number drawn from the Levy flight distribution [55] using Eq. (57),where *dim* represents the dimensionality of the problem being solved. $Rand_6$ and $rand_7$ are randomly generated values between 0 and 1 and θ is a constant number set to 1.5 in Eq. (58).

$$Pos(v+1) = B(v) - |D(t)| \times Fu \times Levy(dim) \quad (56)$$

$$LF(x) = 0.01 \times \frac{rand_6 \times \delta}{|rand_7|^{\frac{1}{\beta}}} \quad (57)$$

$$\delta = \left\{ \frac{\Gamma(1+\theta)\sin(\pi\theta/2)}{\Gamma(1+2\theta) \times \theta \times 2^{(\frac{\theta-1}{2})}} \right\}^{1/\theta} \quad (58)$$

4.4.1 Q-Learning Model in QAVA

The Q-learning mechanism in proposed QAVA provides adaptive decision-making by learning optimal action policies based on feedback from the environment, guiding the vultures' movement toward high-quality solutions more efficiently and avoiding random moves used in standard AVA.

Figure 6 presents the flowchart of QAVA, illustrating the interaction between the Q-learning agent and the metaheuristic optimization algorithm AVA within the QTE-IoT framework.

State Space (*S*):

Each state $s_t \in S$ represents the current condition of the optimization process during iteration *t*. In QAVA, states are defined based on the vultures' population fitness distribution, social behaviors, and the iteration count, reflecting the algorithm's progress and the quality/diversity of solutions.

Similar QARO definitions, the state vector s_t for QAVA formally expressed as a collection of performance metrics related to the current population and system conditions by Eq. (43). Where $f_{best}(t)$ is the best fitness value among all vultures (solutions) at iteration *t*. $LI_{worst}(t)$ quantifies the load imbalance across fog nodes at iteration *t*.

Action Space (*A*):

The action space *A* consists of AVA search operators and comprises two types of actions. The first is exploration, which involves moving a vulture toward one of the best vultures using the roulette wheel selection method to explore new areas of the search space more globally (Eqs. 45-49). The second is exploitation, which includes behaviors such as food competition (Eq. 50), vulture rotation (Eq. 53), social accumulation (Eqs. 55–56), and aggressive competition modeled with Levy flights (Eq. 56). At each iteration and given state $s_i(t)$, the Q-learning agent selects an action $a_t \in$ {exploration, exploitation} guided by the policy derived from the Q-values.

Reward Function (r_t):

In the proposed QAVA algorithm, the reward function quantifies the immediate improvement in the fitness metric after executing an action, reflecting multiple objectives including reductions in task completion time,

deadline violations, energy consumption, and load imbalance, as formulated in Eq. (24). The reward is computed as the difference between the fitness values before and after taking the action, where positive rewards signal enhanced scheduling performance and encourage the Q-learning agent to select similar beneficial actions in future iterations. Conversely, lower or negative rewards discourage ineffective moves. This reward feedback mechanism facilitates a balanced exploration-exploitation strategy and contributes significantly to the improved convergence and solution quality of the fog-layer task scheduling process.

Integration and Influence on QAVA:

The Q-learning agent dynamically selects among the available vulture behaviors by estimating action values (Q-values) during each iteration. Unlike standard AVA, where decisions in exploration and exploitation phases are random or probabilistic, QAVA's Q-learning-driven policy chooses actions that have historically led to better solutions, based on cumulative rewards.

During the exploration phase (triggered when the hunger factor $Fu \geq 1$ as per Eq. 48), Q-learning guides vultures either toward promising areas (Eq. 46) or towards generating new candidates within the search space boundaries (Eq. 49).

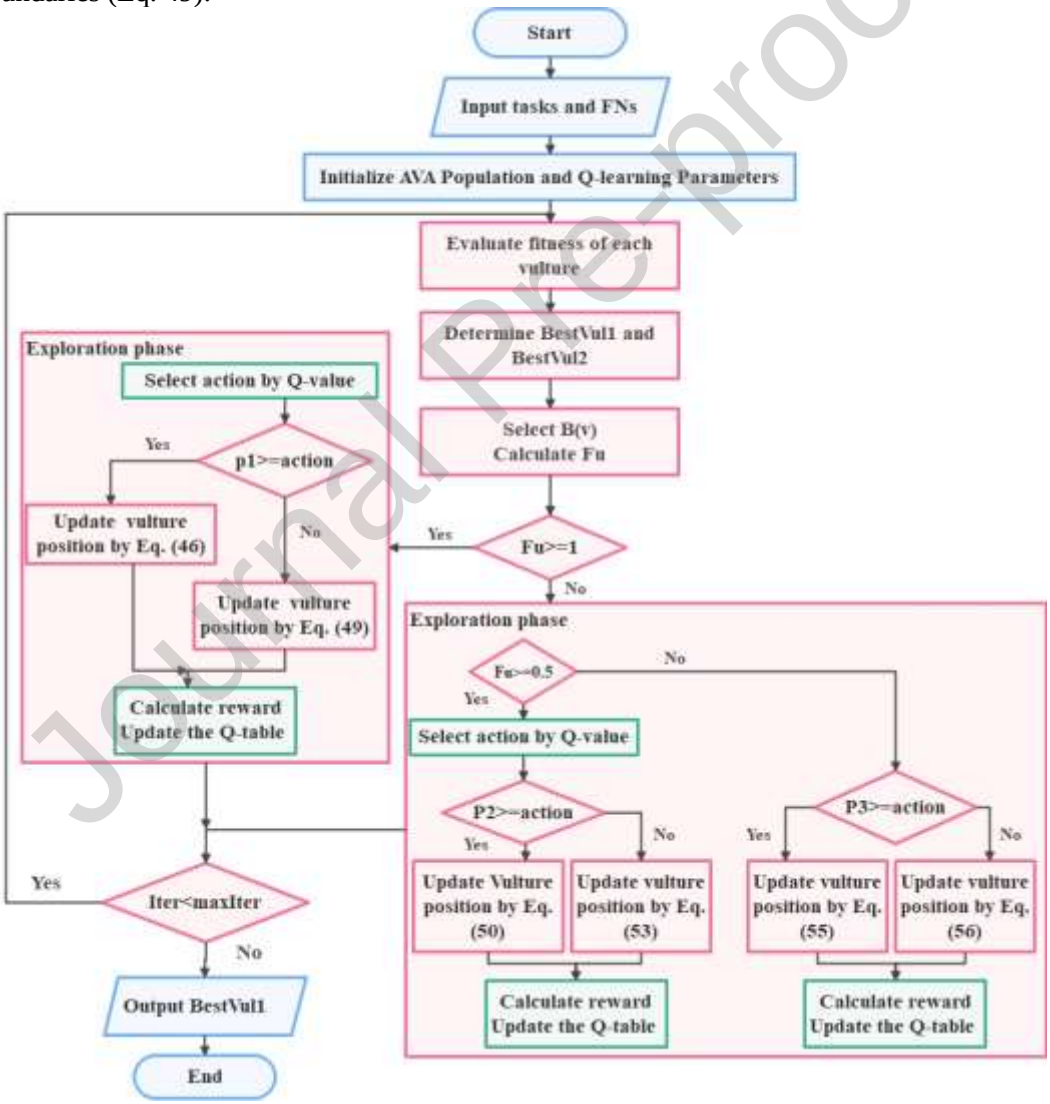


Fig. 6. The Flowchart of QAVA.

In the exploitation phase ($0.5 \leq Fu < 1$), Q-learning selects among food competition, rotation, accumulation, or aggressive competition behaviors based on learned policies, thereby improving local search efficiency and avoiding premature convergence.

The Q-table is updated iteratively using the Q-learning update rule (Eq. 44), incorporating rewards computed from fitness improvements after applying actions. This continuous feedback loop enables QAVA to refine its policy dynamically, effectively balancing exploration and exploitation in a problem-adaptive manner.

Algorithm 2 presents the pseudocode for the proposed QAVA. In the fog layer, the ***TST*** set serves as input for the QAVA algorithm. The process begins with the initialization of parameters and the initial population, which consists of various combinations of FNs for task execution. Once initialized, the algorithm evaluates the fitness function of the vultures using Eq. (24) (line 9). The first-best, and second-best vultures are identified based on their fitness values. Subsequently, actions are selected to guide other vultures toward these top performers. The exploration or exploitation phase is determined based on Eq. (48). During the exploration phase, while the standard AVA randomly selects actions, the QAVA utilizes Q-learning algorithm to make more informed decisions, which is a key advantage of QAVA over the standard AVA.

Algorithm 2: Procedure of QAVA

Input: Tasks' set (***TST***), FNs' set ($\boldsymbol{R_F}$)
Output: Best solution

```
1  Initialize
2  Pop, MaxIter, P1(exploitation probeblity), P2 (Exploration probeblity), P3(social probability),
3  initialize_population with FNs.
4  for Iter in MaxIter
5          {
6          for vulture in Pop
7                  {
8   # Evaluate fitness of the vulture
9                          fitness = evaluate_fitness(vulture) by Eq. (24);
10                         Set PBestVulture1 as the location of Vulture;   #First best location Best Vulture
11                         Set PBestVulture2 as the location of Vulture;   #Second best location Best Vulture
12  # Select an action (move) using Q-value
13                         Select B(v) by Eq. (45);
14                         Calculate Fu by Eq. (48);
15                         if (Fu ≥1)
16   # Explore phase
17                                 {
18                                 select action by Q-value; # AVA (Fu) → Q-Learning (Action Selection)
19                                 if (P1≥ action)
20                                  {
21                                      update the position of the vulture by Eq. (46);
22                                      }
23                                  else
24                                      {
25                                      update the position of the vulture by Eq. (49);
26                                    }
27                                 update the Q-table by Eq. (44);
28                                 else if (Fu≥0.5)
29  # Exploit: Choose the best action
30                                  {
31                                   select action by Q-value;  # AVA (Fu) → Q-Learning (Action Selection)
32                                   if (P2≥ action)      # Food Competition
33                                        update the position of the vulture by Eq. (50);
34                                   else                 # Vulture Rotation
35                                       update the position of the vulture by Eq. (53);
36                                   update the Q_table by Eq. (44);
37                                   select action by Q-value; # AVA (Fu) → Q-Learning (Action Selection)
38                                   if (P3≥ action)        # Accumulation of Different Vultures on Food Source
                                         update the position of the vulture by Eq. (55);
```

```
39                          else                    # Aggressive Food Competition
40                              update the position of the vulture by Eq. (56);
41                          Calculate reward;
42                          update the Q_table by Eq. (44); #AVA (Vulture Position) → Q-Learning (Q-Table
43  Update)
44                          }
    return PBestVulture1;
```

In this phase, an action is selected using Q-learning (line 18), and vultures search for food in their vicinity, at a random distance from one of the best vultures in the two groups, as per Eq. (46). Alternatively, a random generation of solutions may be created using Eq. (49). After completing the exploration phase, the Q-table is updated (line 27). In the exploitation phase, while the standard AVA again opts for random action selection, QAVA determines actions using the Q-learning algorithm (line 31). Based on these determined actions, vultures either complete food acquisition or rotate positions, followed by an update to the Q-table for the selected action. Additionally, Q-learning is employed once more to determine actions based on social probability in QAVA, contrasting with standard AVA's random selection (line 37). This facilitates an accumulation of various vultures around food sources and promotes aggressive food competition. The Q-table is then updated for the determined action. The algorithm iteratively updates the positions of subsequent vultures and ultimately identifies the best vulture as the optimal solution.

The QAVA algorithm introduces significant modifications to the AVA by integrating Q-learning. These changes primarily affect the decision-making process for exploration and exploitation phases, as well as the mechanism for updating vulture positions. The integration of Q-learning allows QAVA to make informed decisions during both exploration and exploitation phases, unlike the standard AVA, which relies on random action selection.

In QAVA, actions are selected using Q-values rather than random probabilities, and the Q-table is dynamically updated to improve decision-making over iterations. During the exploration phase, actions are chosen based on Q-values compared against the exploitation probability *P1*, influencing whether vultures move randomly or toward the best vultures. This approach contrasts with the standard AVA, where vultures explore randomly. The Q-table is updated after each action to refine future decisions, enhancing the algorithm's ability to adapt and converge efficiently.

In the exploitation phase, QAVA selects actions using Q-values compared against the probabilities $P2$ and $P3$, dictating specific behaviors such as food competition, vulture rotation, accumulation, or aggressive competition. This contrasts with the standard AVA, which exploits using random moves. The Q-table is updated after each exploitation move, ensuring that the algorithm learns from past iterations and identifies optimal solutions more effectively.

The parameters $P1$, $P2$, $P3$ serve as thresholds for comparing Q-values, guiding the balance between exploration and exploitation phases more intelligently than in AVA. This integration of reinforcement learning enables QAVA to leverage adaptive action selection, which optimizes fitness outcomes and helps avoid premature convergence to suboptimal solutions. By diversifying search directions during exploration, QAVA improves its ability to avoid local optima. Additionally, the dynamic updating of the Q-table ensures quicker identification of optimal solutions, enhancing convergence speed.

Furthermore, QAVA better models complex interactions among vultures by using Q-learning for social probability-based actions, improving solution quality in task scheduling scenarios. These modifications make QAVA a hybrid algorithm that combines the strengths of AVA's metaheuristic approach with reinforcement learning's adaptive capabilities, addressing limitations in convergence speed and solution quality inherent in standard AVA.

4.5. Workload Monitoring

Monitoring agents are strategically deployed within the cloud gateway to oversee the workload of VMs at the cloud layer, while also being embedded in the broker to track the workload of FNs at the fog layer. Following the task scheduling process in both layers, these agents engage in continuous monitoring of resource workloads to ensure optimal performance. If the number of tasks in a resource's buffer exceeds a

predefined threshold or if critical performance metrics—such as CPU usage, memory utilization, and network bandwidth—consistently approach 100%, this signals an overload condition.

In scenarios where the monitoring agent detects an overload at the cloud layer—encompassing both public and private clouds—the task rescheduling process is managed by the QARO. QARO intelligently reallocates tasks based on current resource availability and performance metrics, ensuring that system efficiency is maintained while minimizing downtime and maximizing resource utilization.

When the monitoring agent in the fog layer detects an overload, a task offloading process is promptly initiated. This is due to the time-sensitive nature of tasks in this layer, where rescheduling may not be feasible. The overloaded FN then offloads tasks to a selected destination FN. The QTE-IoT employs a greedy algorithm to determine the most suitable destination FN for offloading tasks. As detailed in Algorithm 3, this process begins by identifying neighboring FNs with workloads below a specified threshold, designating them as candidate FNs within the Candidate set. The algorithm then assesses these candidates based on their response times and current workloads. If an FN with a shorter response time is identified, it is selected as the destination FN, and one of the tasks in the buffer of FN is migrated accordingly. This iterative process continues until the workload of the original FN is reduced to, or falls below, the defined threshold.

Algorithm 3: Destination fog nod selection

Input: Threshold, set of FNs ($\boldsymbol{R_F}$)
Output: Destination fog nod

```
1  Candidate[]=0;   //Array of Candidate FN
2  minTime = +∞ // Initialize with positive infinity
3  DestinationFN = null // Initialize the selected node as null
   ---------------------------------------------------------------------------
4  for each FN in the fog layer
5      if (Ld < threshold)
6                  {
7                      Candidate [i] = FN;
8                       i++;
9                  }
10 for each FN in Candidate []
11                 {
12                     if (FN.responime < minTime)
13                                 {
14                                     minTime = FN.responseTime;
15                                     DestinationFN = FN;
16                                 }
17                 }
18 return DestinationFN;
```

By implementing monitoring agents across both layers, the system not only enhances responsiveness to overload situations but also optimizes resource allocation, ultimately improving overall performance and load balancing in cloud and fog layers.

4.6. Computational Complexity

Computational complexity is crucial for evaluating the efficiency of algorithms addressing optimization problems, such as those in the QTE-IoT method, which consists of three main components. The first part involves classifying tasks into three categories using an MLP-ANN, with a time complexity of O(N). The second part includes two algorithms: the QAVA algorithm schedules time-sensitive tasks in the fog layer, while the QARO algorithm manages security tasks in the private cloud and normal tasks in the public cloud. The computational complexity of QARO is expressed as $O(T \cdot N \cdot d + T \cdot N + N)$, and for QAVA, it is $O(N \cdot T \cdot d)$, where N is the number of vultures/rabbits, T is the maximum number of iterations, and d is the dimensionality of the problems. The final part features a monitoring agent that uses a greedy algorithm to

oversee resource workloads, with a time complexity of O(N·lg N). Therefore, the overall complexity of the QTE-IoT method can be summarized as O(N.lg N + T·N·d + T·N + N)= O(Nlg N + T·N·d).

5. Evaluation

The effectiveness of QTE-IoT was evaluated and compared against a range of existing task scheduling algorithms, including standard ARO [42], AVA [43], MGWO [20], OSAPSO [36], as well as more complex methods like ETFC [39] and WCLA+GA [11]. These comparisons were conducted across various criteria such as network load distribution, energy consumption, response time, and the ability to meet deadlines for IoT tasks. Standard ARO and AVA algorithms were chosen as benchmarks to demonstrate QTE-IoT's improvements. MGWO was selected for comparison due to its focus on energy efficiency and QoS within a three-layer FCIoT architecture. Additionally, OSAPSO was included due to its strong performance in both QoS and energy consumption.

The evaluation employed instances of the HCSP (Heterogeneous Computing Scheduling Problem) benchmark dataset [56]. This dataset, generated using the Expected Time to Compute (ETC) model, accurately simulates a diverse computing environment by incorporating data related to scheduling independent tasks on heterogeneous processors. The HCSP dataset included diverse resource types and tasks, each with their own Computation Best Time (CBT). To enhance the dataset and facilitate task classification, parameters such as login credentials, email IDs, passwords, and service types were randomly assigned to each the tasks. This resulted in a dataset with 18 features per task. Tasks were then labeled based on their features. For example, tasks containing passwords were classified as security tasks. Time-sensitive tasks were defined as those with a computational budget (CBT) of less than 30 minutes, sizes ranging from 100 to 372 MB, and deadlines evenly distributed between 100 and 500 milliseconds. Normal tasks, on the other hand, had sizes between 100 and 500 MB and deadlines ranging from 500 to 2500 milliseconds. This process yielded a labeled dataset of 1000 tasks, with a distribution of 40% Normal tasks, 50% Time-sensitive tasks, and 10% Security tasks. Tasks were distributed among the FNs and VMs for simulation, with the volume of IoT tasks ranging from 200 to 600. Each FN and VM were given a CPU processing capacity between 2000 and 6000 MIPS to introduce diversity, and the energy usage in the active state was randomly distributed from 80 to 200 Joules. A 1-3 ms propagation delay was considered between communication link bandwidth aligned with HCSP dataset specifications. Packet arrival times were randomly generated between 1 and 20. The simulations were conducted using MATLAB R2019b on a Windows 11 computer system with an Intel® Core i7 processor and 8GB of RAM. Each experiment was repeated 10 times for reliability, and the reported results were based on the average values. Additional simulation details can be found in Table 3.

Table 3. The simulation parameters.

Parameter	Value
Initial population (MGWO, OSAPSO, QARO, QAVA)	30
Iterations	50
Number of FNs	50
Number of private cloud VMs	50
Number of public cloud VMs	50
β(Q-learning)	0.1
γ(Q-learning)	0.15
L1, L2 (QAVA)	0.8, 0.2
w (QAVA)	2.5
P1, P2, P3 (QAVA)	0.6, 0.4, 0.6

5.1. MLP-ANN Performance

The MLP-ANN designed for task classification features 18 neurons in the input layer, which correspond to task attributes, and two neurons in the output layer, categorizing tasks into three classes: time-sensitive tasks, security tasks, and normal tasks. Seventy percent of the tasks were allocated to the training set, while

the remaining 30% served as the test set for classification. To evaluate the performance of the MLP-ANN in task classification, we assessed accuracy, precision, recall, and F1-score. Accuracy represents the proportion of correctly predicted classes, precision measures the proportion of true positives within predicted positives, recall measures the proportion of true positives within actual positives, and the F1-score is the harmonic mean of precision and recall. Our model achieved an average accuracy of 98.88%, a precision of 98.73%, a recall of 98%, and an F1-score of 98.5%, demonstrating strong performance across these metrics as shown in Table 4.

Table 4. Performance metrics of the MLP-ANN model.

Metric	value
Accuracy	98.88%
Precision	98.73%
Recall	98%
F1-score	98.5%

5.2. Benchmark Comparison

To evaluate the effectiveness of the QARO algorithm, we conducted a comprehensive benchmark comparison against standard Q-learning (QL) and ARO alone. The results are summarized in Table 5. These results demonstrate that QARO outperforms both standalone QL and ARO in terms of convergence speed, energy efficiency, and deadline adherence. A total of 400 tasks, including both security tasks (*ST*) and normal tasks (*NT*), were utilized to assess the algorithm's versatility. A mix of 25 private cloud (R_{pc}) and 25 public cloud (R_c) resources was employed to mimic real-world deployment scenarios. The evaluation was based on makespan, energy consumption, deadline violations, and load imbalance to provide a comprehensive assessment. Fifty independent runs were conducted for each algorithm to ensure statistical reliability.

Table 5. Benchmark Comparison of QARO

Metric	QARO	QL-Only	ARO-Only	Improvement
Makespan (s)	142	217	195	**34% faster**
Energy Consumption (J)	18200	24700	22100	**26% better**
Deadline Violations (%)	4.1	11.3	8.2	**54% reduction**
Load Imbalance	0.11	0.28	0.19	**61% improvement**

The simulation settings were designed to mimic real-world cloud computing scenarios, with tasks having varying priorities and resource requirements. The use of both private and public cloud resources allowed for the evaluation of the algorithm's adaptability across different deployment environments.

The benchmark comparison highlights QARO's ability to balance multiple objectives effectively, achieving significant improvements in makespan, energy consumption, and load balancing. This suggests that the hybrid approach is well-suited for complex cloud task scheduling problems where multiple performance metrics need to be optimized simultaneously.

Table 6. Benchmark Comparison of QAVA

Metric	QAVA	QL-Only	AVA-Only	Improvement
Makespan (s)	120	180	150	**20% faster**
Energy Consumption (J)	15500	22500	20000	**22% better**
Deadline Violations (%)	3.5	10.5	7.0	**50% reduction**

Load Imbalance	0.08	0.25	0.15	**47% improvement**

To evaluate the effectiveness of the QAVA algorithm, a comprehensive benchmark comparison was conducted against standard Q-learning (QL) and the African Vulture Algorithm (AVA) alone. The results are summarized in Table 6, demonstrating QAVA's superior performance in convergence speed, energy efficiency, and deadline adherence.

A total of 400 tasks, including time-sensitive tasks, were used to assess the algorithm's versatility. Fifty fog resources were employed to mimic real-world deployment scenarios. The evaluation focused on makespan, energy consumption, deadline violations, and load imbalance to provide a comprehensive assessment. Fifty independent runs were conducted for each algorithm to ensure statistical reliability.

QAVA outperforms both standalone QL and AVA across all metrics. It achieves a makespan of 120 seconds, which is 20% faster than AVA-Only and significantly quicker than Q-learning-Only. Energy consumption is reduced to 15,500 J, representing a 22% improvement over AVA-Only. Deadline violations are substantially reduced to 3.5%, marking a 50% decrease compared to AVA-Only. Additionally, QAVA achieves a load imbalance of 0.08, which is a 47% improvement over AVA-Only.

Overall, these results highlight the benefits of integrating Q-learning with AVA in QAVA. By combining these approaches, QAVA enhances efficiency, reliability, and scalability, making it a robust solution for optimizing task scheduling in fog computing environments. The synergy achieved through this integration is evident in the significant improvements across all evaluated metrics.

5.3. Energy Consumption

Figure 7 illustrates the energy utilization of cloud and fog resources in relation to TS with varying task quantities ranging from 200 to 600 and including 50 public VMs, 50 private VMs in the cloud layer, and 50 FNs in the fog layer. The energy usage is calculated using Eq. (20), which accounts for the overall energy expended in both active and inactive states. In the simulation, energy consumption values are set at 0.1 during inactive periods and 0.6 when active. One of the primary goals of the proposed approach is to optimize resource energy consumption by effectively allocating tasks across available resources. The chart demonstrates that energy consumption increases as the number of tasks rises from 200 to 600 across all methods. A comparison between the QTE-IoT and other scheduling techniques highlights the effectiveness of the QTE-IoT in minimizing energy consumption. In 200 tasks, the results of QTE-IoT is near to AVA and ETFC. However, as shown in Table 7, the QTE-IoT performs better in terms of energy consumption to other compare methods and algorithms.

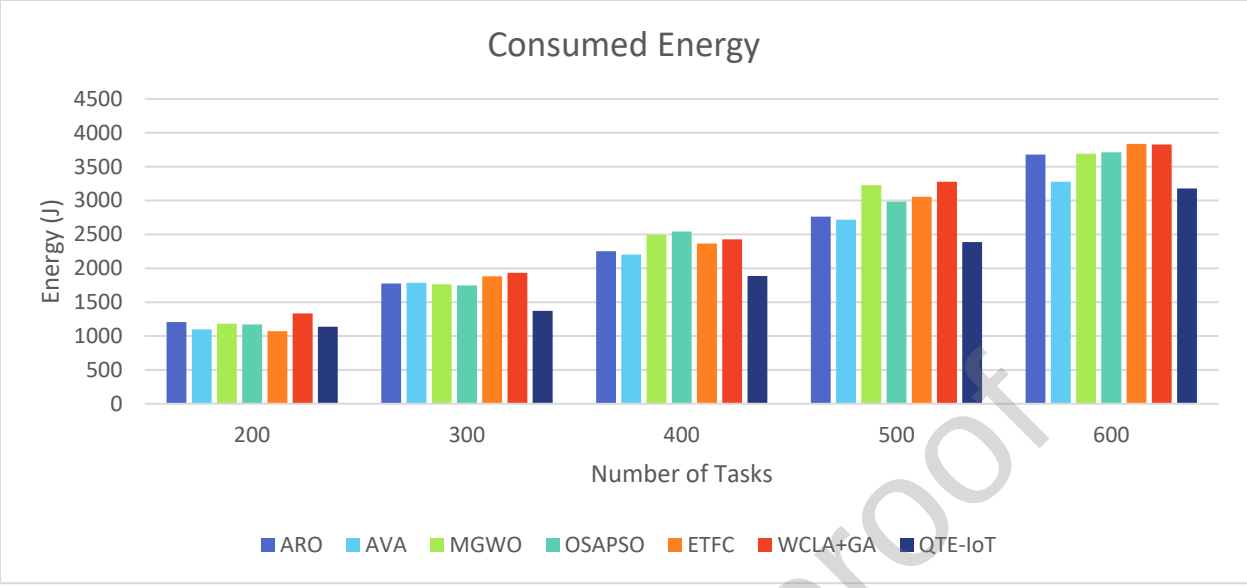


Fig. 7. The total energy consumption of fog and cloud layers.

Table 7 presents the energy usage of methods with different tasks across 30 to 50 resources. It also calculates the average energy usage and measures the percentage improvement of the QTE-IoT method compared to the standard algorithms and other methods. Specifically, as shown in Table 7, the QTE-IoT method has achieved approximately a 12% improvement in energy consumption compared to the WCLA+GA and a 11% improvement compared to the ETFC. The proposed method, which integrates metaheuristics with Q-learning, has the potential to significantly enhance the efficiency of traditional metaheuristic approaches. By incorporating the adaptive learning capabilities of Q-learning, the method can achieve a faster convergence rate, enabling QTE-IoT to find optimal or near-optimal solutions in a shorter timeframe.

Table 7. The energy consumption analysis.

Method	Number of Resources									Average	Improvement of QTE-IoT (%)
	30			40			50				
Number of tasks	200	400	600	200	400	600	200	400	600		
MGWO	1287.97	2901.84	3943.01	1234.28	2629.34	3657.63	1205.12	2252.63	3679.47	2532.37	7.19
OSAPSO	1443.64	2878.36	4117.55	1472.53	2370.74	3865.08	1097.10	2202.79	3275.98	2524.86	6.92
ARO	1258.07	2737.67	4390.06	1140.77	2597.23	3371.82	1181.97	2490.91	3691.51	2540.00	7.47
AVA	1251.74	2732.53	3866.33	1355.34	2457.22	3428.20	1170.63	2543.59	3712.03	2501.96	6.06
WCLA+GA	1411.62	3041.91	4213.77	1478.95	2548.73	3937.32	1071.70	2364.65	3835.87	2656.06	11.51
ETFC	1468.29	2798.69	4093.52	1389.08	2643.20	3915.07	1332.82	2425.94	3826.78	2654.82	11.47
QTE-IoT	1155.54	2333.32	4102.74	1413.08	2366.07	3578.27	1137.97	1886.43	3178.95	2350.26	0.00

5.4. Load balancing

To assess load balancing at the resource level, we examine the degree of imbalance among three types of resources in our scenario: FNs, public cloud VMs, and private cloud VMs. We calculate the DLI separately for FNs, public cloud, and private cloud using Eq. (23) and then determine the maximum among them using Eq. (59). Here, DLI_{fog} represents the DLI in the fog layer, $DLI_{PrivateCloud}$ indicates the DLI in the private cloud, and $DLI_{PublicCloud}$ reflects the DLI in the public cloud.

$$DLI_{sys}=\max(DLI_{fog}, DLI_{PrivateCloud}, DLI_{PublicCloud}) \quad (59)$$

Figure 8 illustrates the system's DLI across various tasks for the compared methods in the system, including 50 public VMs, 50 private VMs in the cloud layer, and 50 FNs in the fog layer. The analysis reveals that OSAPSO exhibits subpar performance in load balancing specially when the task number is over than 300, compared to the other two methods. Notably, in 300, 500 and 600 tasks, there are a significant increase in the DLI for OSAPSO, possibly due to getting trapped in local optima. On the other hand, ARO, AVA, ETFC and WCLA+GA have poor performance in 200 tasks and they couldn't find optimal even near optimal solution in this situation. While MGWO and OSAPSO demonstrate similar performance levels, with QTE-IoT showcasing superior load balancing capabilities in 200 tasks. This improvement can be attributed to the monitoring agent's influence in QTE-IoT, which detects resource overload and enhances system load balancing by task rescheduling or utilizing the proposed greedy algorithm for task offloading. Table 8 presents an analysis of load imbalance for various compared methods, utilizing 30, 40, and 50 VMs and FNs across different task scenarios. The average load imbalance results highlight the relatively poor performance of MGWO compared to other approaches. OSAPSO, ARO, and AVA exhibit similar performance levels. QTE-IoT consistently outperforms the other six algorithms and methods, demonstrating a significant improvement in average results. It achieves approximately 42% to 79% reduction in load imbalance compared to the other approaches. This improvement is attributed to the monitoring phase inherent in the QTE-IoT method. This phase continuously monitors resource load and, when exceeding a predefined threshold, utilizes the proposed greedy algorithm to offload tasks to another node in the fog layer or re-schedule tasks in the cloud layer. This proactive approach effectively avoids overload and enhances overall system load balance.

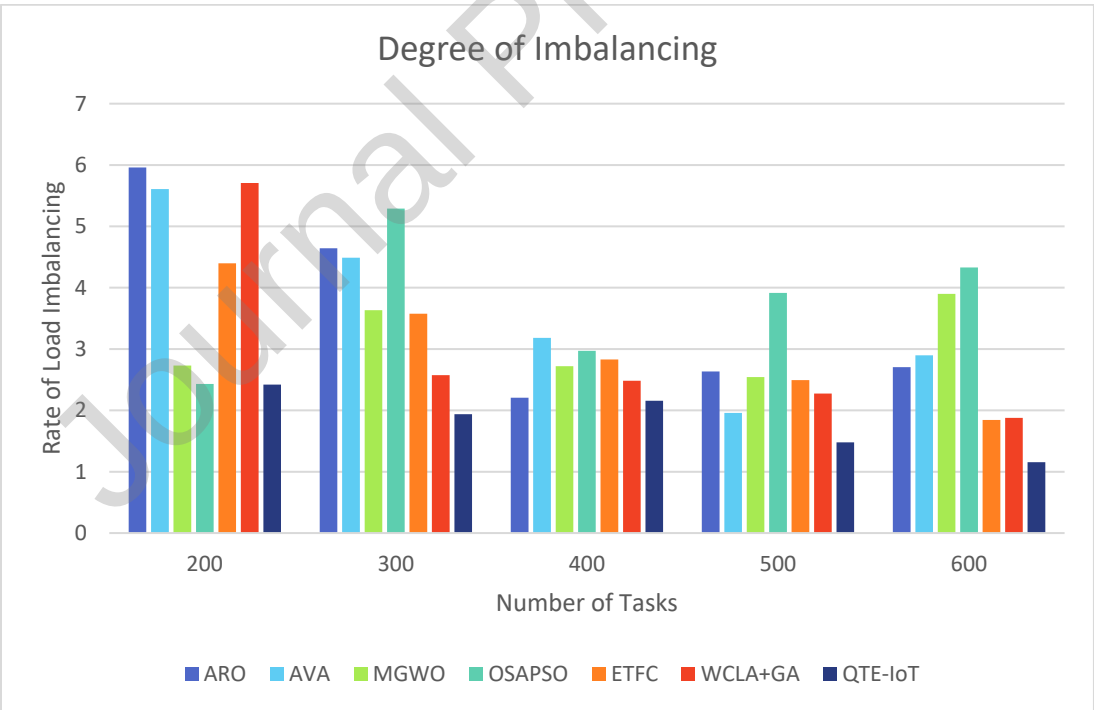


Fig. 8. The system's degree of load imbalance.

Table 8. The load imbalancing analysis.

Method	Number of Resources			Average	Improvement of QTE-IoT (%)
	30	40	50		

Number of tasks	200	400	600	200	400	600	200	400	600		
MGWO	2.93	1.96	1.96	30.00	10.71	2.20	5.96	2.21	2.70	7.27	**79.14**
OSAPSO	3.92	1.78	1.83	5.08	1.92	1.74	5.61	3.18	2.90	3.16	**51.97**
ARO	3.08	2.18	1.93	6.93	3.28	2.08	2.73	2.72	3.90	3.17	**52.10**
AVA	4.00	2.04	1.96	5.44	2.43	1.56	2.43	2.97	4.33	3.08	**50.70**
WCLA+GA	4.65	1.82	1.10	3.64	2.05	1.67	4.40	2.83	1.84	2.65	**42.88**
ETFC	3.58	2.24	1.59	3.40	3.31	1.11	5.71	2.48	1.88	2.87	**47.21**
QTE-IoT	1.26	1.02	1.20	1.54	1.77	2.20	2.42	2.16	1.16	1.52	**0.00**

5.5. Response Time

To assess the QoS, we examine the response time metric, which refers to the time a system takes to respond to a task. Response time is a crucial factor in the QTE-IoT method. The effectiveness of QTE-IoT in minimizing response time is clearly depicted in Figure 9, which showcases significant improvements compared to the alternative methods when using 50 FNs, 50 public cloud VMs, and 50 private cloud VMs. As the volume of tasks decreases, so does the response time. By categorizing tasks and allocating resources accordingly, the system can enhance efficiency and prioritize time-sensitive tasks for quicker completion. Ultimately, reducing response time is vital for optimizing system performance, QoS, and user satisfaction. Interestingly, when the volume of tasks increases to 600, most methods exhibit a decrease in response times compared to scenarios with 500 tasks. This is likely due to the variation in task sizes. Generally, tasks in the 500-task scenarios are larger than those in the 600-task scenarios. It is also noteworthy that the response time of the ARO and AVA algorithms increases, unlike other methods. This behavior might be attributed to these algorithms potentially falling into local optima traps, preventing them from fully optimizing their performance in these scenarios.

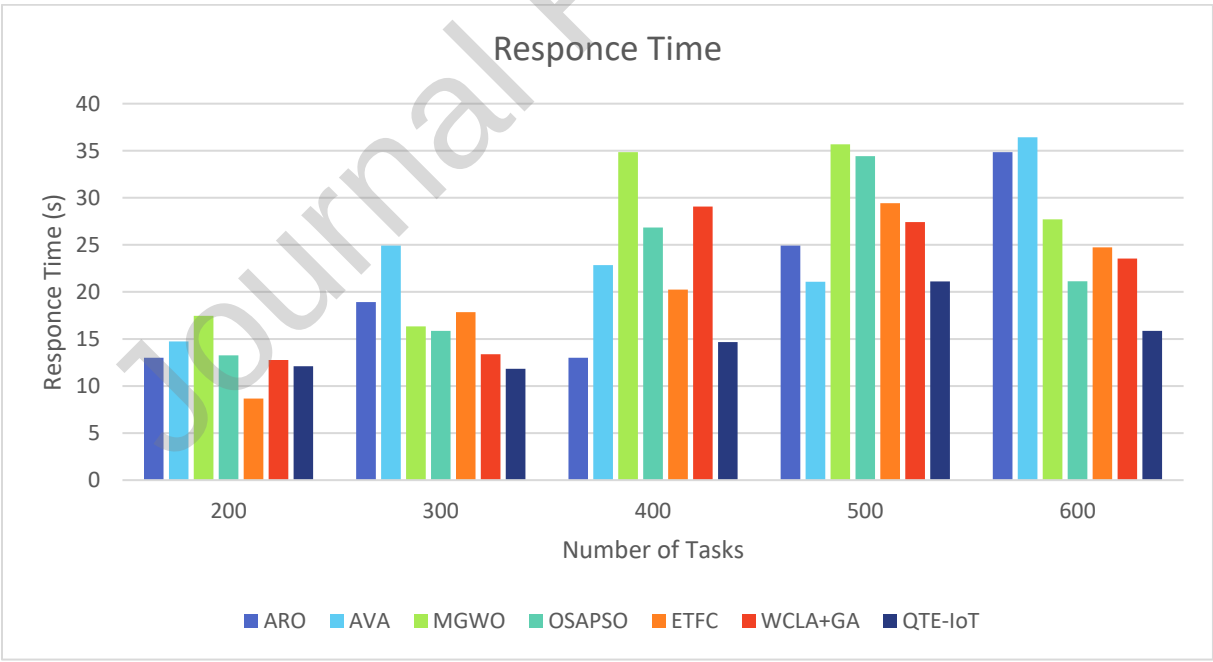


Fig. 9. The response time of the system.

Table 9 presents an analysis of response times for methods utilizing 30, 40, and 50 VMs and FNs across various tasks. The results indicate that OSAPSO generally demonstrates poorer performance in response time compared to the other methods. Notably, QTE-IoT outperforms its counterparts, achieving better average results with approximately 25% to 40% improvements in response time relative to other algorithms and methods. By integrating Q-learning with standard metaheuristics (ARO and AVA), QTE-IoT

accelerates the solution-finding process more effectively than traditional metaheuristics. Additionally, when compared to other methods, QTE-IoT exhibits a lower response time and superior overall performance.

Table 9. The response time analysis.

Method	Number of Resources									Average	Improvement of QTE-IoT (%)
	30			40			50				
Number of tasks	200	400	600	200	400	600	200	400	600		
MGWO	20.63	25.55	38.56	14.63	21.07	58.20	13.00	13.00	34.86	26.61	37.49
OSAPSO	20.29	24.79	35.69	11.92	16.42	44.75	14.73	22.85	36.43	25.32	34.30
ARO	12.17	21.30	42.71	19.36	45.95	29.50	17.46	34.85	27.72	27.89	40.36
AVA	15.58	21.11	35.16	15.50	23.13	27.00	13.26	26.84	21.13	22.08	24.66
WCLA+GA	19.11	20.36	24.36	14.33	24.29	27.50	8.67	20.25	24.73	20.40	18.46
ETFC	14.17	24.06	28.58	13.13	29.83	28.93	12.77	29.08	23.54	22.68	26.64
QTE-IoT	5.57	18.75	29.40	11.89	23.02	18.46	12.10	14.66	15.85	16.63	0.00

5.6.Tasks Deadline Compliance

Meeting task deadlines is another important parameter in QoS evaluation. In our evaluation test, we analyze both the time of deadline violations and the percentage of deadlines met. Figure 10 illustrates the deadline violation times for various tasks in the system, which includes 50 public VMs, 50 private VMs in the cloud layer, and 50 FNs in the fog layer. As depicted in the figure, when there are 200 tasks, deadline violation times are minimal for all algorithms and methods. However, as the volume of tasks increases, resource limitations lead to an increase in deadline violation time.

Figure 11 displays the proportion of tasks that are completed on time. QTE-IoT stands out with over 98% deadline satisfaction, showcasing strong performance compared to the other algorithms and methods. While QTE-IoT exhibits slightly lower performance than OSAPSO and MGWO in terms of deadline violation time, it consistently outperforms other methods in terms of deadline satisfaction percentage. This discrepancy can be attributed to QTE-IoT's prioritization of maximizing the number of tasks that meet their deadlines, even if this means potentially increasing the violation time for a subset of tasks. This approach reflects a deliberate trade-off between minimizing overall deadline violations and ensuring a high percentage of task deadlines are met.

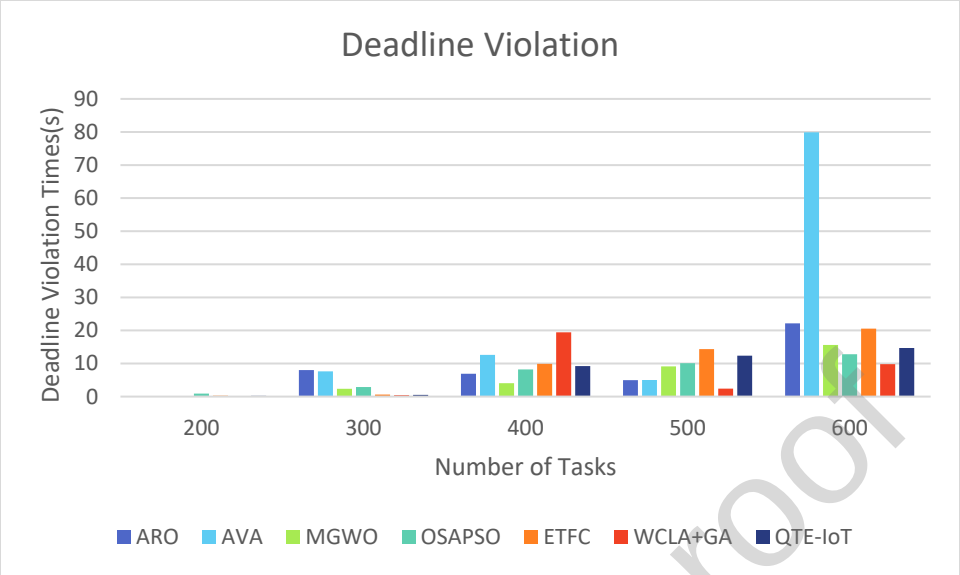


Fig. 10. The total deadline violation time of the tasks.

Table 10 analyzes deadline satisfaction percentages for various methods, utilizing 30, 40, and 50 VMs and FNs across different task scenarios. The results indicate that MGWO generally exhibits inferior performance in meeting task deadlines compared to other approaches. Notably, QTE-IoT consistently outperforms the other algorithms and methods, demonstrating superior average results and achieving approximately 6% to 39% improvement compared to the other algorithms and methods. QTE-IoT's strong performance can be attributed to its advanced task scheduling mechanisms. It not only excels in classifying and prioritizing time-sensitive tasks but also incorporates a monitoring agent that actively tracks resource loads. Through effective load balancing, QTE-IoT can enhance deadline adherence and overall system performance.

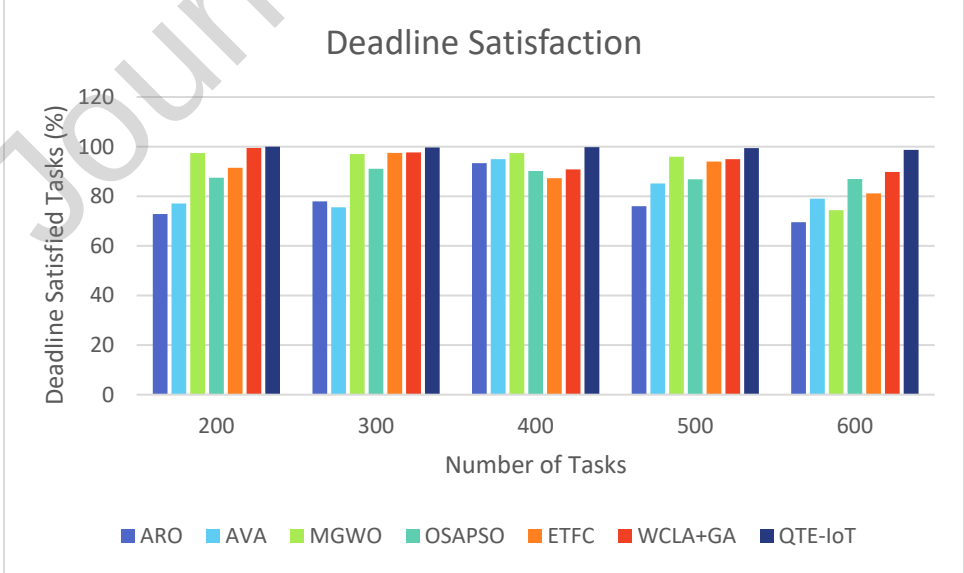


Fig. 11. Percent of deadline satisfaction

Table 10. Analysis of the percentage of deadline satisfaction.

Method	Number of Resources									Average	Improvement of QTE-IoT (%)
	30			40			50				
Number of tasks	200	400	600	200	400	600	200	400	600		
MGWO	85.30	66.77	51.52	85.30	66.77	51.52	72.84	93.31	69.54	71.43	**39.14**
OSAPSO	85.46	83.64	88.70	85.46	83.64	88.70	77.10	94.92	79.04	85.18	**16.67**
ARO	84.59	84.04	60.14	66.97	72.84	67.78	97.37	97.37	74.35	78.38	**26.80**
AVA	73.33	83.68	93.58	97.97	77.38	91.72	87.46	90.17	86.95	86.92	**14.35**
WCLA+GA	94.90	91.81	89.19	81.00	93.82	92.49	91.47	87.24	81.11	89.22	**11.39**
ETFC	96.23	96.57	97.42	94.80	89.82	89.77	99.44	90.79	89.75	93.84	**5.91**
QTE	99.03	99.24	98.82	100.00	99.78	99.18	100.00	**99.76**	**98.66**	99.39	**0.00**

6. Conclusion and Future Work

TS is crucial for optimizing system performance by effectively managing workload, optimizing resource utilization, and minimizing energy consumption. The primary goal of TS is to assign tasks to the most suitable resources while adhering to QoS requirements. This study introduces a novel approach to TS in IoT environments, known as QTE-IoT, which leverages a unique combination of neural networks, metaheuristic algorithms, reinforcement learning (RL), and a greedy algorithm. The primary objectives of QTE-IoT are to enhance energy efficiency and improve QoS by reducing response times and minimizing deadline violations. QTE-IoT employs an MLP-ANN for task classification. Subsequently, it utilizes the proposed QARO algorithm for TS in both private and public cloud environments. Additionally, it employs the proposed QAVA algorithm for TS in the fog layer. This approach combines the strengths of metaheuristics and RL, creating a powerful solution for tackling complex optimization problems. This synergistic approach harnesses the exploration capabilities of metaheuristics, the adaptive learning of RL, and the ability to handle complex constraints and multiple objectives, resulting in more efficient, robust, and effective solutions compared to using either technique alone. Finally, a monitoring agent continuously monitors resource loads and, when necessary, utilizes the proposed greedy algorithm for offloading tasks. Simulation results demonstrate that QTE-IoT outperforms existing methods in various performance metrics, including energy consumption, load imbalance, response time, and deadline satisfaction. Specifically, QTE-IoT shows approximately 6% to 12% improvement in energy consumption compared to existing methods and algorithms. Additionally, it surpasses existing approaches with approximately 42% to 79% improvement in load imbalance, 25% to 40% enhancements in response time, and 6% to 39% improvement in the percentage of tasks meeting their deadlines. These results highlight the significant advantages of QTE-IoT in optimizing task scheduling for IoT environments. Further research and real-world implementation of QTE-IoT could lead to significant advancements in IoT task scheduling for efficient resource utilization and improved system performance.

Future work will focus on expanding the QTE-IoT framework to address more complex challenges. We will incorporate security considerations and dynamic transmission costs into the optimization objectives, aiming to address real-world issues like cybersecurity threats and uncertainties in task arrival and execution. Furthermore, we plan to explore the potential of containerization technologies, which allow for the execution of multiple tasks concurrently within a single resource, to further enhance resource utilization and efficiency. This shift will require adapting the framework to account for this increased resource capacity. Our research will also investigate the application of a wider range of optimization algorithms, including AI-driven approaches, to further improve performance. We plan to conduct comprehensive simulations on large-scale systems and perform real-world deployments to validate the framework's practical effectiveness in various IoT applications. These efforts will contribute to the development of a robust and adaptable task scheduling system that can effectively manage the evolving landscape of interconnected devices.

CRediT authorship contribution statement

A. Ghaffari: Conceptualization, Methodology, Supervision. **V. Firoozi:** Software, Validation. **A. Maleki:** Resources, Conceptualization, Methodology, Validation, Editing. **M. S. Sirjani:** Writing - Review & Editing. **M. Abedini Bagha:** Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing - Original Draft, Data Curation, Visualization, Writing - Review & Editing.

Author Statement

We declare that this manuscript is original, has not been published before and is not currently being considered for publication elsewhere. We confirm that the manuscript has been read and approved by all named authors.

Funding Information

No funding is provided for the preparation of the manuscript.

Declaration of Competing Interest

No conflict of Interest

Data availability

No data was used for the research described in the article.

References

[1] M. Nematollahi, A. Ghaffari, and A. Mirzaei, "Task offloading in Internet of Things based on the improved multi-objective aquila optimizer," *Signal, Image and Video Processing*, vol. 18, no. 1, pp. 545-552, 2024.

[2] F. Dashti, A. Ghaffari, A. Seyfollahi, and B. Arasteh, "A self-predictive diagnosis system of liver failure based on multilayer neural networks," *Multimedia Tools and Applications*, pp. 1-20, 2024.

[3] M. Heidary, E. S. Pour, A. Noori, and M. Abedini Bagha, "Optimisation of energy consumption in cloud video surveillance centre based on monitoring and placement of virtual machines," *International Journal of Computer Applications in Technology*, vol. 70, no. 2, pp. 94-103, 2022.

[4] D. Wang, J. Zhou, M. Masdari, S. N. Qasem, and B. T. Sayed, "Security in Wireless Body Area Networks via Anonymous Authentication: Comprehensive Literature Review, Scheme Classification, and Future Challenges," *Ad Hoc Networks*, p. 103332, 2023.

[5] X. Peng, S. Song, X. Zhang, M. Dong, and K. Ota, "Task Offloading for IoAV under Extreme Weather Conditions Using Dynamic Price Driven Double Broad Reinforcement Learning," *IEEE Internet of Things Journal*, 2024.

[6] Y. Gong, H. Yao, Z. Xiong, C. P. Chen, and D. Niyato, "Blockchain-Aided Digital Twin Offloading Mechanism in Space-Air-Ground Networks," *IEEE Transactions on Mobile Computing*, 2024.

[7] J. Luo, C. Zhao, Q. Chen, and G. Li, "Using deep belief network to construct the agricultural information system based on Internet of Things," *The Journal of Supercomputing*, vol. 7 ,8no. 1, pp. 379-405, 2022.

[8] M. Abdel-Basset, R. NourMoustafa, O. Elkomy, and M. Abouhawwash, "Multi-Objective Task Scheduling Approachfor Fog Computing," *IEEE Access, Piscataway, NY, USA*, 2021.

[9] P. Choppara and S. Mangalampalli, "An Effective analysis on various task scheduling algorithms in Fog computing," *EAI Endorsed Transactions on Internet of Things*, vol. 10, 2024.

[10] Z. Wang, M. Goudarzi, M. Gong, and R. Buyya, "Deep Reinforcement Learning-based scheduling for optimizing system load and response time in edge and fog computing environments," *Future Generation Computer Systems*, vol. 152, pp. 55-69, 2024.

[11] S. J. Jassbi and S. Teymori, "The improvement of wavefront cellular learning automata for task scheduling in fog computing," *Transactions on Emerging Telecommunications Technologies,* vol. 34, no. 8, p. e4803, 2023.
[12] S. Singh and I. Chana, "A survey on resource scheduling in cloud computing: Issues and challenges," *Journal of grid computing,* vol. 14, pp. 217-264, 2016.
[13] S. Ghanavati, J. Abawajy, and D. Izadi, "An energy aware task scheduling model using ant-mating optimization in fog computing environment," *IEEE Transactions on Services Computing,* vol. 15, no. 4, pp. 2007-2017, 2020.
[14] A. K. Budati, S. R. Vulapula, S. B. H. Shah, A. Al-Tirawi, and A. Carie, "Secure multi-level privacy-protection scheme for securing private data over 5G-enabled hybrid cloud IoT networks," *Electronics,* vol. 12, no. 7, p. 1638, 2023.
[15] L. Chen, K. Guo, G. Fan, C. Wang, and S. Song, "Resource constrained profit optimization method for task scheduling in edge cloud," *IEEE Access,* vol. 8, pp. 118638-118652, 2020.
[16] P. Varshney and Y. Simmhan, "Characterizing application scheduling on edge, fog, and cloud computing resources," *Software: Practice and Experience,* vol. 50, no. 5, pp. 558-595, 2020.
[17] M. H. Shirvani and M. Masdari, "A survey study on trust-based security in Internet of Things: Challenges and issues," *Internet of Things,* vol. 21, p. 100640, 2023.
[18] M. Abedini Bagha, K. Majidzadeh ,M. Masdari, and Y. Farhang, "Improving delay in SDNs by metaheuristic controller placement," *International Journal of Industrial Electronics Control & Optimization,* vol. 5, no. 3, 2022.
[19] V. M. Raee, A. Ebrahimzadeh, R. H. Glitho, M. El Barachi, and F .Belqasmi, "E2dne: Energy efficient dynamic network embedding in virtualized wireless sensor networks," *IEEE Transactions on Green Communications and Networking,* 2023.
[20] F. A. Saif, R. Latip, Z. M. Hanapi, and K. Shafinah, "Multi-objective grey wolf optimizer algorithm for task scheduling in cloud-fog computing," *IEEE Access,* vol. 11, pp. 20635-20646, 2023.
[21] S. Shukla, M. F. Hassan, D. C. Tran, R. Akbar, I. V. Paputungan, and M. K. Khan, "Improving latency in Internet-of-Things and cloud computing for real-time data transmission: a systematic literature review (SLR)," *Cluster Computing,* pp. 1-24, 2023.
[22] S. Sheng, P. Chen, Z. Chen, L. Wu, and Y. Yao, "Deep reinforcement learning-based task scheduling in iot edge computing," *Sensors,* vol. 21, no ,5 .p. 1666, 2021.
[23] T. Bu *et al.*, "Task scheduling in the internet of things: challenges, solutions, and future trends," *Cluster Computing,* vol. 27, no. 1, pp. 1017-1046, 2024.
[24] V. M. Raee, A. Ebrahimzadeh, M. Rayani, R. H. Glitho, M. El Barachi ,and F. Belqasmi, "Energy efficient virtual network embedding in virtualized wireless sensor networks," in *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, 2022, pp. 187-192: IEEE.
[25] F. Ramezani Shahidani, A. Ghasemi, A. Toroghi Haghighat, and A. Keshavarzi, "Task scheduling in edge-fog-cloud architecture: a multi-objective load balancing approach using reinforcement learning algorithm," *Computing,* vol. 105, no. 6, pp. 1337-1359, 2023.
[26] M. Kaur and R. Aron, "A systematic study of load balancing approaches in the fog computing environment," *The Journal of supercomputing,* vol. 77, no. 8, pp. 9202-9247, 2021.
[27] T. Salehnia *et al.*, "An optimal task scheduling method in IoT-Fog-Cloud network using multi-objective moth-flame algorithm," *Multimedia Tools and Applications,* pp. 1-22, 2023.
[28] V. M. Raee, D. Naboulsi, and R. Glitho, "Energy efficient task assignment in virtualized wireless sensor networks," in *2018 IEEE Symposium on Computers and Communications (ISCC)*, 2018 ,pp. 00976-00979: IEEE.

[29] H. Rafique, M. A. Shah, S. U. Islam, T. Maqsood, S. Khan, and C. Maple, "A novel bio-inspired hybrid algorithm (NBIHA) for efficient resource management in fog computing," *IEEE Access*, vol. 7, pp. 115760-115773, 2019.

[30] S. Swarup, E. M. Shakshuki, and A. Yasar, "Task scheduling in cloud using deep reinforcement learning," *Procedia Computer Science*, vol. 184, pp. 42-51, 2021.

[31] G. Vijayasekaran and M. Duraipandian, "An efficient clustering and deep learning based resource scheduling for edge computing to integrate cloud-IoT," *Wireless Personal Communications*, vol. 124, no. 3, pp. 2029-2044, 2022.

[32] K. Rezvani, A. Gaffari, and M. R. E. Dishabi, "The bedbug meta-heuristic algorithm to solve optimization problems," *Journal of Bionic Engineering*, vol. 20, no. 5, pp. 2465-2485, 2023.

[33] M. Khojand, K. Majidzadeh, M. Masdari, and Y. Farhang, "Controller placement in SDN using game theory and a discrete hybrid metaheuristic algorithm," *The Journal of Supercomputing*, vol. 80 ,no. 5, pp. 6552-6600, 2024.

[34] M. Abedini Bagha, K. Majidzadeh, M. Masdari, and Y. Farhang, "ELA-RCP: An energy-efficient and load balanced algorithm for reliable controller placement in software-defined networks," *Journal of Network and Computer Applications*, vol. 225, p. 103855, 2024.

[35] A. B. Kanbar and K. Faraj, "Region aware dynamic task scheduling and resource virtualization for load balancing in IoT–fog multi-cloud environment," *Future Generation Computer Systems*, vol. 137, pp. 70-86, 2022.

[36] A .G. Gad, E. H. Houssein, M. Zhou, P. N. Suganthan, and Y. M. Wazery, "Damping-assisted evolutionary swarm intelligence for industrial iot task scheduling in cloud computing," *IEEE Internet of Things Journal*, 2023.

[37] M. Nematollahi, A. Ghaffari, and A .Mirzaei, "Task and resource allocation in the internet of things based on an improved version of the moth-flame optimization algorithm," *Cluster Computing*, pp. 1-23, 2023.

[38] B. Kruekaew and W. Kimpan, "Multi-objective task scheduling optimization for load balancing in cloud computing environment using hybrid artificial bee colony algorithm with reinforcement learning," *IEEE Access*, vol. 10, pp. 17803-17818, 2022.

[39] A. Pakmehr, M. Gholipour, and E. Zeinali, "ETFC: Energy-Efficient and Deadline-Aware Task Scheduling in Fog Computing," *Sustainable Computing: Informatics and Systems*, p. 100988, 2024.

[40] F. A. Saif, R. Latip, M. Derahman, and A. A. Alwan, "Hybrid meta-heuristic genetic algorithm: Differential evolution algorithms for scientific workflow scheduling in heterogeneous cloud environment," in *Proceedings of the future technologies conference*, 2022, pp. 16-43: Springer.

[41] Z. Deng, Z. Yan, H. Huang, and H. Shen, "Energy-aware task scheduling on heterogeneous computing systems with time constraint," *IEEE Access*, vol. 8, pp. 23936-23950, 2020.

[42] L. Wang, Q. Cao, Z. Zhang, S. Mirjalili, and W. Zhao, "Artificial rabbits optimization: A new bio-inspired meta-heuristic algorithm for solving engineering optimization problems," *Engineering Applications of Artificial Intelligence*, vol. 114, p. 105082, 2022.

[43] B. Abdollahzadeh, F. S. Gharehchopogh, and S. Mirjalili, "African vultures optimization algorithm: A new nature-inspired metaheuristic algorithm for global optimization problems," *Computers & Industrial Engineering*, vol. 158, p. 107408, 2021.

[44] F. S. Gharehchopogh and T. Ibrikci, "An improved African vultures optimization algorithm using different fitness functions for multi-level thresholding image segmentation," *Multimedia Tools and Applications*, vol. 83, no. 6, pp. 16929-16975, 2024.

[45] D. Selvaraj, S. Nattuthurai, A. Uehara, and T. Senjyu, "Optimal energy management strategy for electric vehicles and electricity distribution system based on quantile deep learning with enhanced African vulture optimization," *Energy Reports*, vol. 12, pp. 631-655, 2024.

[46] W. Cao, X. Chen, Z. Cao, and B. Badami, "An improved african vulture optimization for biding strategy of two-settlement market in china," *Journal of Electrical Engineering & Technology,* vol. 18, no. 2, pp. 751-764, 2023.
[47] M. A. Mohammed, M. A. Ahmed, and A. V. Hacimahmud, "Data-Driven Sustainability: Leveraging Big Data and Machine Learning to Build a Greener Future," *Babylonian Journal of Artificial Intelligence,* vol. 2023 ,pp. 17-23, 2023.
[48] G. Sun, D. Liao, D. Zhao, Z. Xu, and H. Yu, "Live migration for multiple correlated virtual machines in cloud-based data centers," *IEEE Transactions on Services Computing,* vol. 11, no. 2, pp. 279-291, 2015.
[49] S. Azizi, M. Shojafar, J. Abawajy, and R. Buyya, "Deadline-aware and energy-efficient IoT task scheduling in fog computing systems: A semi-greedy approach," *Journal of network and computer applications,* vol. 201, p. 103333, 2022.
[50] Y. Javed and N. Rajabi, "Multi-layer perceptron artificial neural network based IoT botnet traffic classification," in *Proceedings of the Future Technologies Conference (FTC) 2019: Volume 1*, 2020, pp. 973-984: Springer.
[51] A. Ghaffari, N. Jelodari, S. pouralish, N. derakhshanfard, and B. Arasteh, "Securing internet of things using machine and deep learning methods: a survey," *Cluster Computing,* pp. 1-25, 2024.
[52] H. Asgharzadeh, A. Ghaffari, M. Masdari, and F. S. Gharehchopogh, "An Intrusion Detection System on The Internet of Things Using Deep Learning and Multi-objective Enhanced Gorilla Troops Optimizer," *Journal of Bionic Engineering,* vol. 21, no. 5, pp. 2658-2684, 2024.
[53] E.-G. Talbi, "Machine learning into metaheuristics: A survey and taxonomy," *ACM Computing Surveys (CSUR),* vol. 5 ,4no. 6, pp. 1-32, 2021.
[54] Q. Liu, H. Kosarirad, S. Meisami, K. A. Alnowibet, and A. N. Hoshyar, "An optimal scheduling method in IoT-fog-cloud network using combination of Aquila optimizer and African vultures optimization," *Processes,* vol. 11, no. 4 ,p. 1162, 2023.
[55] X.-S. Yang, *Nature-inspired metaheuristic algorithms*. Luniver press, 2010.
[56] T. F. Oliveira, S. Xavier-de-Souza, and L. F. Silveira, "Improving energy efficiency on SDN control-plane using multi-core controllers," *Energies,* vol. 1 ,4no. 11, p. 3161, 2021.

CRediT author statement

Ali Ghaffari: Conceptualization, Methodology, Supervision.

Vesal Firoozi: Software, Validation.

Ali Maleki: Resources, Conceptualization, Methodology**,** Validation**,** Editing.

Mohammad Sadegh Sirjani: Writing - Review & Editing.

Maedeh Abedini Bagha: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing - Original Draft**,** Data Curation**,** Visualization, Writing - Review & Editing.

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐ The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Highlights

- This research introduces a novel approach for classifying tasks into time-sensitive, security, and normal categories. This classification is achieved using an MLP-ANN.
- An enhanced Artificial Rabbits Optimization (ARO) algorithm integrated with Q-learning, designated as QARO, has been developed for task scheduling in private and public cloud environments. This enhanced algorithm effectively optimizes task scheduling, leading to improved performance.
- A novel African Vulture Algorithm combined with Q-learning (QAVA) has been designed specifically for task scheduling in the fog layer. QAVA is tailored to the unique characteristics of fog computing environments and demonstrates superior performance in this context.
- A monitoring agent has been implemented to track the workload of fog nodes and virtual machines. When the workload exceeds a predefined threshold, tasks are rescheduled in the cloud layer and offloaded in the fog layer using a proposed greedy algorithm. This strategy effectively improves load balancing and reduces response times, resulting in a more efficient and responsive system.