Latency-Aware and Energy-Efficient Task Offloading in IoT and Cloud Systems with DQN Learning

Amina Benaboura [*] , Rachid Bechar , Walid Kadri , Tu Dac Ho [*] , Zhenni Pan , Shaaban Sahmoud

Posted Date: 4 June 2025

doi: 10.20944/preprints202506.0360.v1

Keywords: task offloading; deep Q-networks (DQN); internet of things (IoT); energy consumption; Latency

Article

Latency-Aware and Energy-Efficient Task Offloading in IoT and Cloud Systems With DQN Learning

Amina Benaboura [1,*], Rachid Bechar [2], Walid Kadri [2], Tu Dac Ho [3,4,*], Zhenni PAN [5], and Shaaban Sahmoud [6,7]

[1] Laboratory of Applied Mathematics, Department of Computer Science, FSEI, Hassiba Ben Bouali University of Chlef, Chlef, Algeria
[2] LIA laboratory, Department of Computer Science, FSEI, Hassiba Ben Bouali University of Chlef, Chlef, Algeria
[3] Faculty of Information Technology and Electrical Engineering, NTNU- Norwegian University of Science and Technology, Trondheim 7491, Norway
[4] Faculty of Engineering Science and Technology, UiT- The Arctic University of Norway, Narvik 8514, Norway
[5] Faculty of Science and Engineering, Global Center for Science and Engineering, Waseda University, Tokyo 169-0072, Japan
[6] Computer Engineering Department, Fatih Sultan Mehmet Vakif University, Istanbul 34015, Turkey
[7] Data Science Application and Research Center (VEBIM), Fatih Sultan Mehmet Vakif University, Istanbul 34015, Turkey
[*] Correspondence: tu.d.ho@ntnu.no(T.D.); a.benaboura@univ-chlef.dz (A.B.).

Abstract: The exponential growth of the Internet of Things (IoT) has generated significant challenges related to computing capacity and energy consumption. IoT devices produce large amounts of combined data and demand processing tasks, often leading to high latency and redundant energy consumption. Task offloading has emerged as an applicable solution; however, existing strategies often neglect optimization of latency and energy consumption. This paper introduces a novel task-offloading approach based on deep Q-network (DQN) learning, designed to intelligently and dynamically balance these important metrics. The proposed framework continuously optimizes real-time task offloading decisions by deploying the adaptive learning capabilities of DQN, thus significantly improving latency and energy consumption. To further enhance performance, the framework integrates optical networks into IoT-fog-cloud architecture, leveraging their high-bandwidth and low-latency capabilities. This integration enables tasks to be distributed and processed more efficiently, especially in data-intensive IoT applications. In addition, we present a comparative analysis of the proposed DQN algorithm and the optimal strategy. Through extensive simulations, we demonstrate the effectiveness of the proposed DQN framework in different IoT scenarios over the BAT and DJA approaches, enhancing energy consumption and latency by 35%, 50%, 30%, and 40%, respectively. Our results further demonstrate the importance of selecting an appropriate strategy according to the specific requirements of the IoT application, particularly in terms of environmental stability and performance requirements.

Keywords: Task offloading; Deep Q-networks (DQN); Internet of things (IoT); Energy consumption; Latency.

1. Introduction

The Internet of Things (IoT) is a vital part of our lives because of its wide applications and ability to connect various devices for communication and information exchange [1]. IoT is increasingly used in several industries, including healthcare, industrial automation, smart homes, and emergency response. Data explosions are the result of several devices being connected more easily, due to IoT. The International Data Corporation (IDC) estimates that by 2025, there will be more than 41 billion connected IoT devices, producing more than 79 ZB of data annually [2]. By 2030, this number might increase to 500 billion devices. On the other hand, these mobile devices (MDs) generate a lot of data and require a platform to manage it due to the constraints of limited resources, poor battery life, and limited storage capacity [3]. Furthermore, certain services require low latency, higher equitability, and high spectral and energy efficiency, as well as interactive real-time gaming, virtual reality (VR), and augmented reality (AR).

Cloud computing is a cost-effective and efficient technique for executing and storing massive volumes of data for MDs [4]. The cloud platform provides on-demand resource provisioning for common services, eliminating the need for additional hardware or software. In addition, applications and algorithms can be hosted in the cloud to perform computationally demanding tasks. However, delays and poor service performance may result from distances between the cloud and data sources, particularly for time-sensitive applications such as online gaming and video streaming [5]. A new computing paradigm is required to handle applications that are latency-sensitive and require real-time data with minimal delays to ensure high performance and meet the strict requirements effectively.

For services that need to handle data instantaneously, fog devices provide cloud computing processing and storage power, minimizing latency and maximizing bandwidth. The installation of a fog node (FN) to offer a feasible and imaginative solution for real-time applications has become a popular approach in the fog computing paradigm [1]. In many applications, MDs perform significant computational tasks. However, due to the limited energy required for computing and storage services, the tasks must be transferred to the FN [6]. The data path has been changed so that the data is now handled locally on devices close to data sources, rather than regularly passing through the cloud. Fog devices minimize latency and optimize bandwidth by utilizing cloud computing and storage capabilities for applications that require real-time processing and substantial computing resources [7,8]. Moreover, an optical network is a great option to allow frequent resource scheduling and information exchange between servers, which will further improve the responsiveness and efficiency of fog and cloud networks [24]. Advanced resource and scheduling techniques are necessary to manage fog devices and control task offloading [9]. It needs sophisticated resources and scheduling techniques to manage fog devices and regulate work offloading.

However, task offloading in the cloud remains common even if fog computing is less resource-intensive than remote cloud computing. Therefore, in the IoT, fog, and cloud paradigms, efficient approaches are needed to detect when tasks should be offloaded and ensure that they are accurately distributed to the appropriate computing resources [4]. Furthermore, network latency, communication protocols, capacity, cost, security, offload options, and authentication are significant issues in integrated cloud fog networks [10]. Energy efficiency, execution time, service cost, and other indicators of quality of service (QoS) are all impacted by these issues. As stated in [29–32,40], task requests can be offloaded to FN or cloud servers rather than on-premise servers to mitigate the above-mentioned problems. This will save processing time and energy.

Motivated by that, this work presents a latency-aware and energy-efficient task-offloading in fog and cloud computing networks using deep Q-networks (DQN) learning. The study concentrates on large-scale IoT devices, or FNs, connected through the optical network. , which are part of the task offloading system [11]. The performance study focuses mainly on QoS, latency, and energy efficiency [12]. The suggested approach is novel, as it includes latency and energy characteristics as preferences for task offloading.

The primary goal of this research is to provide a novel approach to task offloading for IoT-fog-cloud computing paradigms that considers energy efficiency and latency. The DQN model in MDs is utilized to make these judgments. The following paragraphs briefly describe the main contributions of this paper:

- A task offloading problem is formulated as a multi-objective optimization problem to minimize both latency and energy.
- A DQN learning-based energy-efficient and latency-aware algorithm is proposed for optimal offloading decision strategies in the IoT-fog-cloud collaboration model and provides a methodology for assigning tasks to different layers.
- We evaluated and validated the suggested model and showed how this model improves QoS metrics by comparing it with previous studies. (in particular, energy consumption and application latency).

The remainder of this paper is organized as follows. Section 2 reviews the related work. Section 3 presents the formulation of the problem, the objective function, and the system model. Section 4 describes the suggested algorithm, and a pseudocode is supplied. Section 5 evaluates the proposed algorithms and presents the results. Finally, section 6 provides a summary and recommendations for further study.

2. Related Works

Several works have explored the optimization issues of offloading problems in cloud and fog computing networks [35]. The two main problems with IoT services that have an impact on QoS metrics (quality, availability, and reliability) are energy consumption and latency. This section covers recent studies of the most significant methods discovered through literature research, highlighting their contributions, limitations, and differences from our proposed strategy.

In [16] we find an overview of current research in the cloud computing paradigm that proposes a refined whale optimization algorithm (RWOA) offloading technique for latency-sensitive, computationally demanding, and power-hungry tasks. Multiple processing locations (local, mobile edge computing (MEC), cloud) can split up the tasks [17]. In addition, a large number of users can offload their tasks to the fog and cloud server [18]. The authors of [1] suggested an intelligent framework for choosing the best resource to achieve a latency-sensitive IoT application and making secure offloading decisions. In this work, they used a metaheuristic BAT algorithm to optimize the characteristics of service cost, energy, and latency. However, the authors presented the bat-inspired evolutionary approach in a collaborative fog cloud environment and dynamically researched the optimal resource using food-searching behavior [36]. In [41], the authors proposed a joint optimization approach based on deep Q-learning for task offloading at the device level and the edge level. In their proposal, Chiang et al. [43] improved resource allocation and dynamic reconfiguration of network slices in a multi-tenant edge computing system by utilizing Q-learning-based deep learning. The goal of this work is to jointly solve the dynamic slice scaling and task offloading issues from the standpoint of service providers (SPs) profit in the multitenant edge computing (EC) system. In [15], a deep reinforcement learning (DRL) based task-offloading scheme was proposed to jointly offload tasks with dependencies. Specifically, we model the dependencies between tasks through directed graphs with discrete loops and formulate the task-offloading issue as minimizing the average cost of energy and time (CET) for users.

Table 1 summarises the objectives, constraints, and solutions introduced in the reviewed literature. Also, it provides a detailed comparison of our work with existing literature. We categorize the reviewed literature into two aspects: the first is energy-aware task offloading, and the second is latency-aware task offloading.

2.1. Energy-Aware Task Offloading

Recently, the issue of energy-efficient task offloading in the context of cloud and fog computing has received more attention. In [4], Alasmari, M. K., et al. addressed the problem of employing an intelligent allocation technique to optimize energy utilization in IoT devices. This multi-classifier for energy efficient tasks offloading (MCEETO) selects the best fog devices for task placement using an algorithm based on multi-classifiers to offload tasks efficiently. The MCEETO algorithm is an effective method for reducing energy consumption, and the study highlights the importance of energy conservation in fog computing. This article provides insights into energy optimization strategies related to fog computing and IoT devices. The authors compared the MCEETO algorithm with two existing techniques, namely edge-ward and cloud-only, in terms of various performance measures. Moreover, the edge ward has been selected because it represents a placement strategy focused on the edge only. In another hand, cloud-only placement assumes that all application modules run in cloud data centers [13]. On the other hand, the authors of [14] suggested secrecy energy efficiency (SEE) by optimizing transmission power time allocation and task partitioning while meeting secrecy and energy constraints. They investigated the secure offloading of a wireless-powered MEC system and proposed a security-enabled scheme for the physical layer.

Table 1. Comparaison of research publications

Reference	Objectives	Proposed solution	Executions locations
[1]	Minimize network latency and energy	BAT based	Local devices, fog, and cloud .
[4]	Minimize energy	MCEETO	local devices, fog, and cloud.
[13]	Minimize latency	Edge-ward	local devices, edge, fog, and cloud.
[14]	Minimize energy	SEE	local devices and MEC.
[15]	Minimize network latency and energy	MOBA-CV-SARSA	local device and edge server .
[16]	Minimize latency, energy and cost	RWOA	Local device, MEC, and cloud .
[19]	Minimize energy consumption	MSTEC and HREC	Local device, fog, and cloud.
[20]	Minimize energy consumption	DRL-based	Local device, fog, and cloud.
[21]	Minimize cost and latency	PSO	Local device, fog, and cloud.
[22]	Minimize latency	Collaborative cloud-edge scheme	Local device, edge, and cloud.
[23]	Minimize latency	federated learning-based	Local device, edge, cloud.
[24]	Minimize delay	TD-based CLB-TO and GA-based CLB-TO	Local device, edge servers connected by optical network
[32]	Minimize latency	Scheduling and queue management algorithms	Local device, edge, and cloud.
[41]	Minimize latency and energy	Deep Q-learnin	Local device and edge.
[42]	Minimize latency	LSTM and dual DQN	Local device and edge.
[43]	Minimize latency and energy	DRL-based	Local device and edge.
Our	Minimize latency, energy, and cost	DQN-based	Local device, fog, and cloud.

As with the aforementioned efforts, in [19] the authors proposed a low delay scheduling algorithm for energy-constrained fog computing workflows, called the minimal schedule time with energy constraint (MSTEC) algorithm. Furthermore, to maximize the system reliability of fog and cloud computing networks, a high reliability algorithm with an energy constraint (HREC) algorithm was presented for fog computing workflow subject to energy consumption restriction. In their contribution, Ale et al.[20] propose a DRL-based approach to specify an ideal number of resources and choose the best edge server to perform offloaded operations. Their objective is to reduce energy consumption and increase the number of tasks performed.

2.2. Latency Aware Task Offloading

This part provides an overview of recent research proposing latency-efficient task offloading methods, especially for latency-sensitive tasks. Sabireen et al.[21] proposed models that minimize processing costs and delays. However, they do not consider the security of IoT applications. The authors have proposed a lightweight modified particle swarm optimization (PSO) technique with clustering to maximize resource efficiency while minimizing latency. This approach allows for the offloading of IoT tasks to FNs in real time. The proposed technique employs a machine learning model with enhanced PSO to reinforce other key QoS factors and reduce latency between fog nodes and

IoT devices. Gupta et al. [13] proposed a centralized edge-to-edge module deployment technique for distributed systems represented as directed acyclic graphs (DAGs). Their approach begins by installing modules from the base of the fog hierarchy and works backward until it finds a node with sufficient resources. Nevertheless, the method only allows modules to extend vertically by transferring data to fog or cloud and then back to the application; it disregards horizontal connections between FN at the same level. For the same purpose, a closed-form task offloading policy is obtained by transforming the formulation of a computational and communication resource allocation problem to minimize the weighted sum of MD latency [22]. However, the goal of using a collaborative cloud and edge computing method is to reduce the delay. Tang et al. [43] presented a model-free distributed DRL-based algorithm to offload tasks in MEC. The authors incorporated long short-term memory (LSTM) and dual DQN techniques to reduce task drop rates and average response time compared to the previous algorithms.

In their contribution, Chen et al. [23] studied the computation offloading problem for latency and privacy-sensitive tasks in a hierarchical local-edge-cloud architecture using a federated learning method. The main objective of this approach is to minimize the running time of latency-sensitive tasks requested by MDs with data privacy concerns, while each task can be executed under various computing modes. Also, they formulate an optimization problem with constraints to reduce the latency that the federated offloading cooperation consumes. Also, the authors of [33] have proposed scheduling and queue management algorithms to address the issue of task offloading; they take into consideration the maximum tolerable latency. Conversely, a different study has focused on determining the best location for task offloading, ensuring that the maximum latency is achieved while minimizing the energy consumed as much as possible. Some authors proposed a metaheuristic-based approach to address the problem of task offloading. The authors of [34] introduced an approach based on the discrete jaya algorithm (DJA) to reduce latency and optimize resource utilization (RU). Additionally, they proposed a task migration algorithm to transfer the partially completed task to another server. The authors in [24] proposed two heuristic algorithms: the transmission delay-based CLB-TO and the genetic algorithm-based CLB-TO, for computing offloading balancing with task offloading. They studied how to reasonably select the target edge server for each task while minimizing the completion delay. They considered that edge servers were connected by the optical network.

It is observed from the literature review that most techniques had good performance on the objective targets, but these works need to balance contradictory goals. However, many studies desert the collaborative role of fog and cloud computing networks within the optical network or without it [14,15,24,41–43]. Meanwhile, most studies that consider the collaboration of fog and cloud have failed to consider latency, energy, and cost simultaneously [4,13,14,21–23,31,42], which are critical in task offloading.

In this regard, and differently from the above discussions, we formulate a fundamental IoT-fog-cloud architecture comprising MDs, FNs, and a cloud server. The main contribution is the design and implementation of an optimized task offloading strategy based on DQN. This intelligent offloading system allows for real-time selection of the best processing node by dynamically adjusting to task-specific features and ambient variables. Our method focuses on systems that are particularly vulnerable to latency constraints, computing requirements, and energy usage. The proposed DQN-based approach efficiently reduces total system latency and energy consumption through the use of reinforcement learning, which optimizes operating costs without sacrificing QoS. Our approach, in contrast to static or heuristic-based strategies, continuously learns from the environment, considering important parameters such as task deadlines, cloud data transfer energy limits, the available bandwidth and energy between MDs and FNs, and task queue stability across all layers.

3. System Model and Problem Formulation

This section presents a three-level model that explains the different task scenarios. First, we illustrate each part of the model in detail. Next, we introduce the task offloading model that contains an overview of the proposed framework. Finally, we present the objective function of the proposed research work.

3.1. *System model*

The three-layer collaboration model is considered in this work, namely the devices, the fog, and the cloud layer, as shown in Figure 1. IoT devices that are MDs responsible for processing tasks independently or by sending them to higher levels make up the first layer, also known as the infrastructure layer. This layer has the lowest latency and computational capacity. The fog layer, which is the second layer, is made up of numerous FNs distributed in different regions connected by the optical network, offering a range of functionalities [4]. FNs are small servers with intermediate latency and computational capacity. The cloud layer, which constitutes the third layer, has moderately high latency and very high computing capacity. Cloud servers can have huge computational capacities because they comprise numerous high-performance machines that are kept in a data center [25]. The layers are connected through wired and wireless connections. In addition, a maximum bandwidth is assigned to each level of connection.

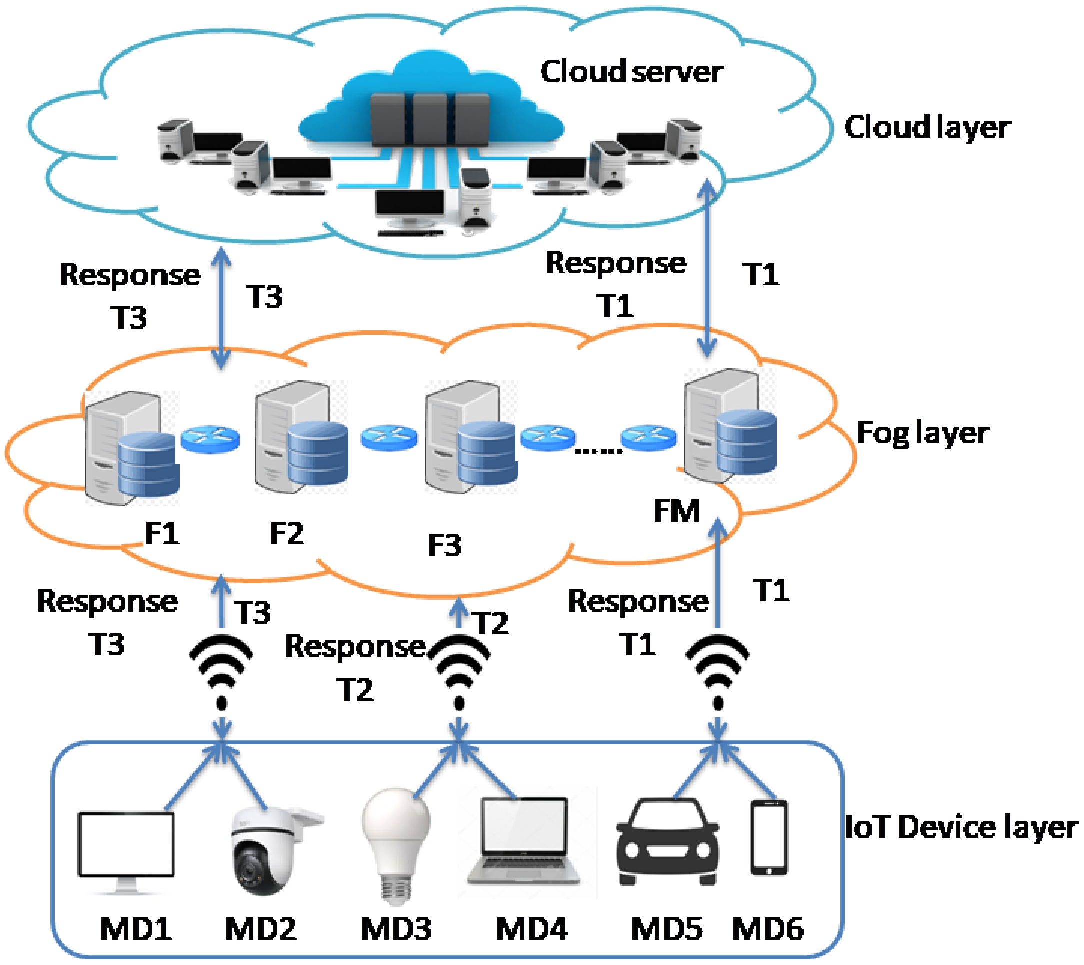


Figure 1. Overview of IoT-fog-cloud architecture.

Our proposed system involves an arrangement of MDs to collect the heterogeneous data that needs processing power. Table 2 represents a list of the several notations that are utilized in this paper.

Each MD may complete a computing task (T_i) where $T_i \epsilon$\{ T_1, T$_2$,. . . .., T$_N$\}, N is the number of tasks, and each task is owned by a user (U_j) ϵ \{ U_1, U$_2$,. . . .., U$_K$\}. The main characteristics of a computing task are as follows:

D$_{T_i}$: The data size of the task.

τ_{T_i} : The maximum acceptable delay needed to complete the task.

W$_{T_i}$: The total workload of the task.

As shown in Figure 1, there are M fog nodes (FN) F= \{F$_1$, F$_2$,. . . . . . ., F$_M$\}, and a cloud server (C).

Equation (1)represents the relationship between D$_{T_i}$, CB, and W$_{T_i}$[38].

$$W_{T_i} = D_{T_i} * CB, \tag{1}$$

where W_{T_i} represents the total workload of task (T_i), (CB) is the CPU cycles needed per bit of data, and D_{T_i} is the data size of the task (T_i).

Table 2. Abbreviations used in the proposed model

Notation	Description
MD	A set of mobile devices.
N	Number of tasks.
T	Set of tasks.
T$_i$	The task number i.
D$_{Ti}$	Task data size.
τ_{Ti}	The maximum acceptable delay to execute task T_i.
CB	CPU cycles required per bit of data.
W$_{Ti}$	Total workload of the task T_i.
F	Set of fog node.
M	Number of fog nodes.
C	Cloud server.
L_{Ti}	Latency(execution time of task T_i).
X^k_{Ti}	Decision offloading matrix of task T_i from user i in the location k.
f_d	CPU frequency of device d for a processing task.
Q	The initial latency queue.
E	Energy consumption of the task.
η	Energy efficiency factor.
$P^{loc}_{d,Ti}$	Time processing of task T_i locally.
$L^{loc}_{d,Ti}$	The total latency of processing task T_i.
$E^{loc}_{d,Ti}$	Energy consumption of task T_i that processing locally.
$C^{loc}_{d,Ti}$	Overall cost of task T_i that processing locally.
$T^f_{d,Ti}$	The transmission time of task T_i to fog layer.
$P^f_{d,Ti}$	The processing time of task T_i in fog layer.
$L^f_{d,Ti}$	The total latency of processing task T_i in fog layer.
$E^f_{d,Ti}$	The energy consumption of task T_i that processing in fog layer.
B_{u2f}	The MD and fog node communication bandwidth.
PW_{u2f}	The communication transmission power between MD and fog node.
$C^f_{d,Ti}$	Cost of processing task T_i over fog layer.
$T^c_{d,Ti}$	The transmission time of task T_i to cloud server.
$P^c_{d,Ti}$	The processing time of task T_i in cloud server.
$L^c_{d,Ti}$	The total latency of processing task T_i in cloud server.
B_{f2c}	The communication bandwidth between fog node and cloud server.
PW_{f2c}	The communication transmission power between the cloud server and fog node.
$C^c_{d,Ti}$	Cost of processing task T_i over cloud server.

3.2. *Task Offloading Model*

MD tasks can be executed locally or offloaded to the best fog node in M. Under the assumption that each MD has an indivisible task and that the delays of all tasks are equal to T [26]. Also, the decision matrix $X^{K}_{Ti}\epsilon$ {0,1} is introduced, which indicates the choice to offload task i, which will be executed on server k. In this context, it helps in selecting the most efficient and effective offloading strategy by balancing latency and energy consumption. In particular, local execution is indicated by $(X^{loc}_{Ti} = 1)$, fog node execution by $(X^{f}_{Ti} = 1)$, and cloud execution by $(X^{C}_{Ti} = 1)$.

Furthermore, a single execution location, whether local, fog, or cloud, must be assigned to every task T_{Ti}. As a result, the following equation allows us to ensure these requirements:

$$X^{loc}_{Ti} + \sum_{F=1}^{M} X^{f}_{Ti} + X^{c}_{Ti} = 1 \ \forall i\epsilon\{0.....N\}, \tag{2}$$

where X is a binary variable and loc and C represent local execution and cloud execution, respectively. However, Fϵ [1,..., M] denotes the fog node, where at most one of the following conditions must be satisfied as follows:

$$X = \begin{cases} X^{loc}_{Ti} = 1 \ \Rightarrow local\ execution \\ \sum_{F=1}^{M} X^{f}_{Ti} = 1 \ \Rightarrow fog\ execution \\ X^{c}_{Ti} = 1 \ \Rightarrow cloud\ execution. \end{cases} \tag{3}$$

3.3. *The three variant executions*

In this section, we introduce the three task offloading models: local processing, fog processing, and cloud processing.

3.3.1. Local Execution

During the offloading process, the first available decision that can be taken is to execute the task locally. When a device chooses to process the task Ti from the device (d) independently without sending it to other devices, the local processing time $(P^{loc}_{d,Ti})$ is defined as:

$$P^{loc}_{d,Ti} = \frac{W_{Ti}}{f_d}, \tag{4}$$

where W_{Ti} represents the total workload of task Ti, and the (f_d) is the CPU frequency of the device (d) for processing a task.
The total latency is defined as:

$$L^{loc}_{d,Ti} = P^{loc}_{d,Ti} + Q_{Ti}, \tag{5}$$

where Q_{Ti} represents the initial latency referring to the time tasks spend waiting in the queue.

Similarly, the energy consumption $(E^{loc}_{d,Ti})$ can be defined by equation (6) for local processing of the task:

$$E^{loc}_{d,Ti} = \eta_d * W_{Ti} * (f_d)^2, \tag{6}$$

where η_d represents the energy efficiency of device d, W_{Ti} represents the total workload of task Ti, and (f_d) is the CPU frequency of device (d) for processing a task.

Therefore, the total cost of local processing is determined by combining the weighted local processing latency $(L^{loc}_{d,Ti})$ and local power consumption $(E^{loc}_{d,Ti})$. This can be modeled as follows:

$$C^{loc}_{d,Ti} = \alpha * E^{loc}_{d,Ti} + \beta * L^{loc}_{d,Ti}, \tag{7}$$

it is derived from equations (4) and (5). Where α and β are the energy weights and latency with values between zero and one [1]. To ensure balanced importance between the two objectives, the weights are set as follows: α= 0.18 and β= 0.82. These weights are used to determine the relative importance of each factor in the total cost.

3.3.2. Fog Execution

Due to the limited capability of MD, several tasks must be offloaded to the fog nodes. The overall task latency consists of two parts, the transmission ($L^f_{d,Ti}$) and the processing time ($P^f_{d,Ti}$). The transmission time ($T^f_{d,Ti}$) is calculated by the following equation:

$$T^f_{d,Ti} = \frac{D_{Ti}}{B_{u2f}}, \tag{8}$$

where D_{Ti} is the size of the task to be processed by the fog node, and B_{u2f} is the bandwidth available for data transmission between the MD and FN.

The processing time ($P^f_{d,Ti}$) on the FN is calculated in the same way as the local processing. It is calculated by the equation (9):

$$P^f_{d,Ti} = \frac{W_{Ti}}{f_j} \ \forall j \epsilon [1.....M], \tag{9}$$

The variable f_j represents the CPU frequency of the fog node (j).

The total latency ($L^f_{d,Ti}$) of processing task (T_i) of the device d in the fog layer is defined as:

$$L^f_{d,Ti} = P^f_{d,Ti} + T^f_{d,Ti} + Q^f_{Ti}, \tag{10}$$

where Q^f_{Ti} refers to the time tasks spent waiting in the queue over the FN (j).

Moreover, the energy usage ($E^f_{d,Ti}$) across the fog node is calculated by the equation (11):

$$E^f_{d,Ti} = PW_{u2f} * L^f_{d,Ti} + \eta_j * W_{Ti} * f_j \ \forall j \epsilon [1.....M], \tag{11}$$

where PW_{u2f} is the communication transmission power between the MD and the fog node, W_{Ti} represents the total workload of task T_i, and (η_j) and (f_j) represent the energy efficiency factor and the CPU frequency of the FN (j), respectively.

Equation (12) computes the overall offloading cost over the fog node.

$$C^f_{d,Ti} = \alpha * E^f_{d,Ti} + \beta * L^f_{d,Ti}. \tag{12}$$

3.3.3. Cloud Execution

Since the fog nodes also have limited servers, they can offload the task to the cloud layer. Cloud servers are rich in resources and have efficient power to process any task. When tasks are processed in cloud data centers, the propagation time is increased due to the geographical distance between the task and the resources.

The total latency of the task is represented by ($L^c_{d,Ti}$), which is the combination of two factors called transmission ($T^c_{d,Ti}$) and processing time ($P^c_{d,Ti}$). It is explained as follows:

$$T^c_{d,Ti} = \frac{D_{Ti}}{B_{u2f}} + \frac{D_{Ti}}{B_{f2c}}, \tag{13}$$

$$P^c_{d,Ti} = \frac{W_{Ti}}{f_c} + Q^c_{Ti}, \tag{14}$$

$$L^c_{d,Ti} = T^c_{d,Ti} + P^c_{d,Ti}, \tag{15}$$

where D_{Ti} represents the data size of the task to be processed by the cloud server, B_{f2c} is the bandwidth available for data transmission between the fog node and the cloud server, $f_c$is the CPU frequency of the cloud server, Q^c_{Ti} represents the initial latency referring to the cloud server, and W_{Ti} represents the total workload of task Ti.

Equation (15) is used to calculate the energy consumption of the cloud server:

$$E^{c}_{d,Ti} = PW_{u2f} * L^{c}_{d,Ti} + PW_{f2c} * L^{c}_{d,Ti} + \eta_c * W_{Ti} * f_c, \tag{16}$$

where (PW_{f2c}) is the communication transmission power between the cloud server and fog node, (PW_{u2f}) is the communication transmission power between MD and fog node, (η_c), and (f_c) are the energy efficiency factor, and CPU frequency of cloud server.

The total offloading cost over the cloud server is calculated as follows:

$$C^{c}_{d,Ti} = \alpha * E^{c}_{d,Ti} + \beta * L^{c}_{d,Ti}, \tag{17}$$

the total cost of offloading is the sum of the execution latency and energy consumed by IoT devices and devices across fog and cloud servers. It is calculated as follows:

$$cost = \sum_{i=1}^{N} (X^{loc}_{Ti} * C^{loc}_{d,Ti} + X_(Ti)^{f} * C^{f}_{d,Ti} + X^{c}_{Ti} * C^{c}_{d,Ti}). \tag{18}$$

3.4. Problem Formulation

In this section, we formulate our problem for multi-MDs with multi-generated tasks, taking into account all of the above models. We aim to optimize the overall cost, considering several QoS factors including task data size, communication bandwidth, with special attention to the high throughput capabilities of optical networks - transmission and processing time, as well as the maximum acceptable delay to execute the task. By efficiently offloading tasks, the suggested method aims to decrease task latency and energy consumption by making a suitable offloading decision.

However, we formulate mathematically the cost minimization problem of the task offloading process as follows:

$$Min(cost) = min \sum_{i=1}^{N} (X^{loc}_{Ti} * C^{loc}_{d,Ti} + \sum_{F=1}^{M} (X^{f}_{Ti} * C^{f}_{d,Ti}) + X^{c}_{Ti} * C^{c}_{d,Ti}), \tag{19}$$

s.t.
C1 : $X^{loc}_{Ti} \epsilon \{0, 1\}$
C2 : $X^{f}_{Ti} \epsilon \{0, 1\}$
C3 : $X^{c}_{Ti} \epsilon \{0, 1\}$
C4 : $X^{loc}_{Ti} + X^{f}_{Ti} + X^{c}_{Ti} = 1 \ \ \forall i \epsilon N$
C5 : $B_{Ti} > 0 \ \ \forall i \epsilon N$
C6 : $X^{loc}_{Ti} * L^{loc}_{d,Ti} + X^{f}_{Ti} * L^{f}_{d,Ti} + X^{c}_{Ti} * L^{c}_{d,Ti} <= \tau_{Ti} \ \ \forall i \epsilon N$

The explanation of the mentioned constraints is given as follows:

- C1, C2, and C3 denote that these decision variables are guaranteed to be binary through these 3 constraints.
- C4 indicates that there should only be one location in which each work is completed, so the choice location variable will be equal to 1.
- C5 ensures that the bandwidth assigned to the task must be positive.
- C6 indicates that the task latency must not exceed the maximum tolerable delay τ_{Ti} to execute task T_i, whether in a local, fog, or cloud server.

To solve the optimization problem, it is necessary to determine the optimal offloading value and make a decision by minimizing the cost, which is a combination of latency and energy consumption.

4. Proposed Solution

This part will introduce a task offloading paradigm that uses two methods to solve the offloading issue. The first approach is the optimal strategy, where data from each layer is collected and evaluated to identify the most suitable tasks for offloading. This strategy evaluates all possible offloading decisions (local, fog, cloud) for each task and selects the one with the minimum cost. However, it provides sub-optimal results with respectable performance, establishing a benchmark against which the other approach can be evaluated. This approach necessitates acquiring a significant quantity of real-time data, which is hard to obtain in real-world scenarios.

On the other hand, we recommend using reinforcement learning (RL) algorithms, particularly the DQN model, to achieve optimal solutions. This technique allows the agent to learn from experience and make decisions by storing them in a Q-table. The agent moves on to the next state by taking action a_t, s_{t+1}, after seeing a state S_t at time step t. It also obtains a reward, r_{t+1} [44]. This procedure allows the agent to continuously enhance its decision-making skills based on previous interactions, as shown in Figure 2.

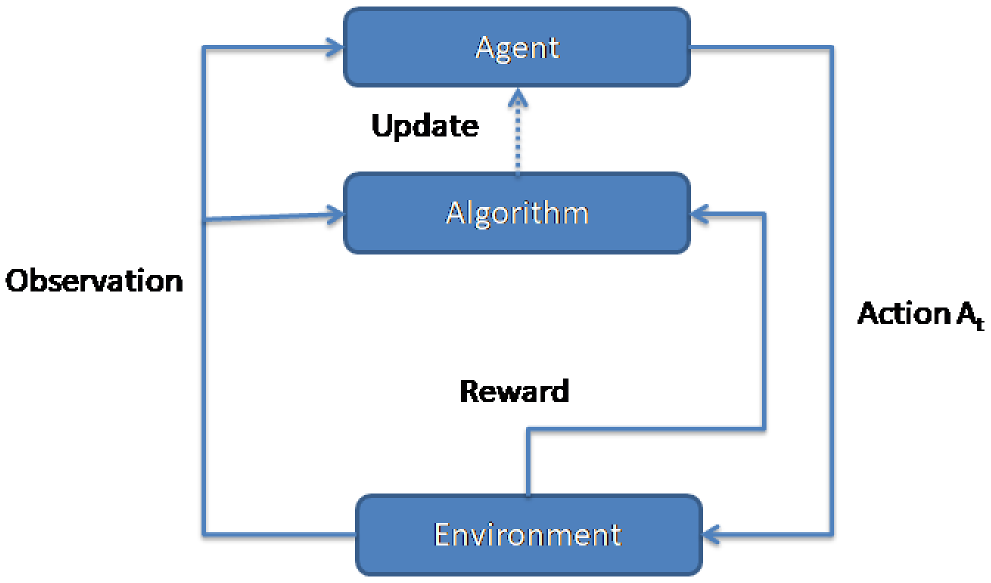


Figure 2. The interaction model of RL algorithms

4.1. Optimal Task Offloading Strategy

A task-offloading paradigm that can concentrate on the task execution location created by MDs is presented in this section. As mentioned, this strategy provides an algorithm to solve the optimization problem and identify sub-optimal solutions with low processing costs.

The algorithm decides where to process the task that the device generates by calculating the cost among all potential execution alternatives (local, fog, and cloud execution) and selecting the option with the lowest cost. Algorithm 1 describes the entire procedure in detail.

The process starts when the device generates the task to be processed as input, and the algorithm needs to collect data about MD, FN, and a cloud server. Then, for each task, the algorithm considers the characteristics of the underlying communications infrastructure, emphasizing the high-capacity, low-latency capabilities offered by optical networks, especially when tasks are moved to remote fog nodes or the cloud. Based on the minimum cost, the execution location is chosen as follows: (0) indicates local execution, (-1) indicates cloud execution, and values between 1 and 10 indicate fog node execution (assuming 10 fog nodes). However, when the location is different from (0) and (-1), it means that the result refers to the number of fog nodes that execute the task in the most cost-effective execution path, which is likely to be influenced by the availability of enhanced bandwidth through optical network links.

Algorithm 1 Optimal task offloading strategy

```
Require: MD, F, C, T.
Ensure: Execution_Location EL, Min_Cost MC .
 1: Function Optimal Strategy:
 2: for Each d in MD do
 3:   for Each Task in T do
 4:       Initialiser(EL, MC)
 5:       //Local Execution
 6:       MC= $C^{loc}_{d,Ti}$ //Using equation (7).
 7:       EL = 0
 8:       // Fog Execution
 9:     for Each Fog_node in F do
10:       Compute $C^{f}_{d,Ti}$ //Using equation(12)
11:       if ($C^{f}_{d,Ti}$ < MC ) then
12:         MC = $C^{f}_{d,Ti}$
13:         EL= Fog_Node
14:       end if
15:     end for
16:       //Cloud Execution
17:       Compute $C^{c}_{d,Ti}$ //Using equation (17)
18:     if ($C^{c}_{d,Ti}$ < MC) then
19:         MC = $C^{c}_{d,Ti}$
20:         EL = -1
21:         Compute cost(d, $T_i$) //Using equation (18)
22:     end if
23:   end for
24: end for
```

4.2. DQN-Based Task Offloading

DQN is a reinforcement learning technique that combines deep neural networks with Q-learning to optimize large state-action spaces. In the context of task offloading in IoT-fog-cloud computing networks, DQN optimizes the decision-making process by deciding whether computational tasks should be kept in local devices, sent to fog nodes, or sent to the cloud server. Therefore, in this paper, we switch to using the tuple (s_t, a_t, r_t), which contains state, action, and reward.

The state space includes various features such as current system status, size of a task, latency of the network, and the load level at the current time t. It can be formulated as:

$$s_t = \{W_{Ti}, D_{Ti}, Q_{Ti}, f_d, f_j, f_c, B_u 2f, Bf2c\}, \tag{20}$$

where W_{Ti} represents the total workload of the task, D_{Ti} is the data size of task, and the Q_{Ti} is the time task spend waiting in the queue. The computational capacities of different processing units are represented by f_d, f_j, and f_c, which represent the CPU frequencies of the device (d), the fog node (j), and the cloud server, respectively. The available bandwidth for data transmission is given by $B_u 2f$ and $Bf2c$, which refer to the bandwidth between the MD and the fog node and between the fog node and the cloud, respectively.

The action space defines the set of possible actions that an agent can perform in the environment, specifically representing task offloading decisions at each time step t. It is formulated as follows:

$$a_t = \{-1, 0, 1.....M\}, \tag{21}$$

where a_t=0 denotes local processing, a_t= -1 denotes cloud processing, and a_t= 1....M denotes fog node execution (assuming 10 fog nodes, so M = 10).

The reward function is aligned with the objective function of the system model, which reflects that the total cost could be minimized by optimizing offloading decisions as to whether the task is executed locally, in the fog layer, or the cloud layer. However, the primary purpose of this paper is to minimize both latency (denoted by L) and energy consumption (denoted by E) [45]. To be consistent with the objective of the model in this paper, we use negative rewards. When the objective function of the system is at its minimum level, the DRL can achieve the maximum reward [39]. The reward function is given as follows, based on the above considerations.

$$r_t(s_t, a_t) = -(\alpha * L + \beta * E), \tag{22}$$

where α and β are a weighting coefficient that measures normalized latency and energy, and balancing multiple objectives appropriately and combining them into a unified reward function.

The DQN model uses a neural network to approximate the Q-value function that calculates the expected future rewards given the quality of the offloading action and the current state. The network is trained and converges to a policy (π) by interacting with the environment: it makes offloading decisions, performs the desired actions (task completion time, energy used, etc.), and then refines the models based on observed performance (QoS metrics). In time, the DQN agent successfully identified the optimal offloading strategy that balanced latency and energy use, adapting dynamically to changing conditions in the IoT-fog-cloud environment. This approach is particularly effective in complex and dynamic environments where traditional methods might find it challenging to maintain optimal performance. An overview of an agent using the suggested DQN technique is illustrated in Figure 3.

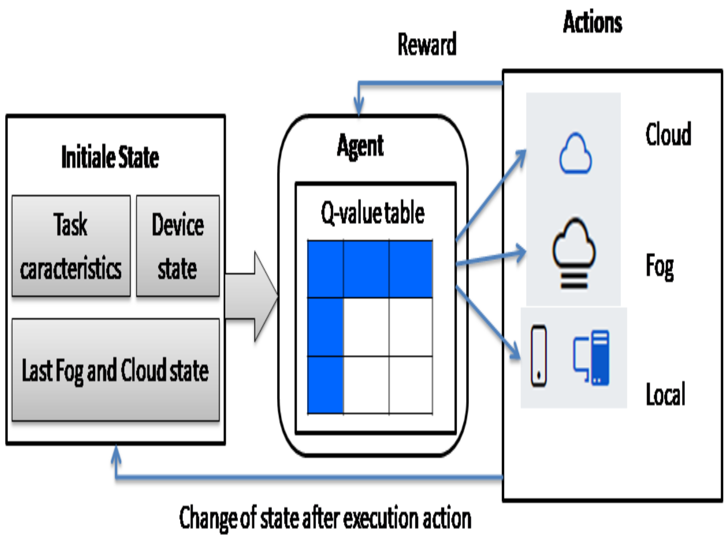


Figure 3. The process of task offloading decision of RL algorithms

Within a Q-learning algorithm [27], we are rewarded based on how well we execute an activity to go from one state to the next. To minimize the total of all rewards obtained during the execution of the algorithm, Q-learning chooses an action from a set of feasible actions at each state. In Q-learning's simplest form, every state and action is stored in a table value [28]. This table calculates the total reward that can be obtained by applying the Q-learning algorithm to each potential state and set of actions. To train this table, we update the Q-value whenever we perform an action (a_t) in a given state. To indicate the order of states we will go through when we execute this algorithm, we define (s1, s2, ...). Additionally, we define (a1, a2, ...) as the associated actions performed at every stage. The DQN-based task offloading strategy is summarized in Algorithm 2.

The DQN architecture has been designed to validate its usage with the proposed algorithm. In other words, the DQN architecture consists of two neural networks: the evaluation network and the target network. The evaluation network calculates Q-values, i.e, expected future rewards for each action. This evaluation network is updated frequently during the training phase. Then, on the other hand, the target network provides stable Q-value targets. This is synchronized with the evaluation network at regular intervals. Each of the two networks is characterized by a two-layer architectural configuration, comprising a three-hidden layer with 64 neurons at each layer. The hidden layers use the ReLu activation function, while the output layer uses linear activation for better prediction of the Q values.

Algorithm 2 DQN-based task offloading strategy

```
Require: Input different tasks from different devices, learning rate($\epsilon$), discount factor ($\gamma$).
Ensure: Optimal offloading decision and total cost.
 1: Function DQN Strategy:
 2: Initialize replay memory D to capacity N.
 3: Initialize total episode reward r = 0.
 4: for Each episode do
 5:   Reset environment to initial state s.
 6:   for Each step do
 7:     Observe actual state $s_t$.
 8:     Determine feasible action.
 9:     random = randomly choose from [0, 1]
10:     if (random < $\epsilon$) then
11:       $a_t$ = randomly select action from {0,1,2}
12:     else
13:       $a_t = \gamma * max_{action}$ Q($s_t, a_t$).
14:     end if
15:     Execute action $a_t$.
16:     Calculate reward $r_t$ using equation (22) .
17:     Observe next state $s_{t+1}$.
18:     Store ($s_t, a_t, r_t, s_{t+1}$) in replay memory.
19:     Update $Q(s_t, a_t)$ according to equation (23)
20:   end for
21: end for
```

Therefore, by acting at each step, the agent will choose the action that changes the state from s_t to state s_{t+1} under a policy (π), and then receive the reward (r_t). Furthermore, the Q-table is updated with the state (s_t) and action (a_t) as follows with the Bellman equation [37]:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \epsilon(r_t + \gamma * a_t maxQ(s_{t+1}, a_{t+1}) - Q(s_t, a_t)), \tag{23}$$

where r_t is the reward function earned from moving from state (s_t) to (s_{t+1}) by taking action a_t, and (γ) is the discount factor. The (γ) indicates how much we value earlier rewards; its value ranges between zero and one. However, (ϵ) represents the learning rate, which affects how quickly the table will converge.

To summarize, the DQN framework allows the offloading agent to easily learn optimal policies for task offloading suitable for the complex IoT-fog-cloud environment by approximating different actions with long-term rewards. This results in adaptive and efficient offloading strategies, instead of being static and predetermined.

5. Performance Analysis and Discussion

In this section, we use comprehensive simulations to test the performance of our suggested strategy. Python 3.19.13 was used on a 64-bit Windows operating system, equipped with an Intel (R)

Core (TM) i5-6300U CPU running at 2.50 GHZ and 8GB of RAM. Table 2 was used to determine the simulation parameters.

5.1. Simulation Model

IoT devices randomly receive different sets of data. 100 IoT devices, 10 fog nodes, and a cloud server are all included in the scenario. Each training sample is assumed to have a batch size of 32 for both fog nodes and IoT devices. Tasks are randomly generated with different sizes and different resource requirements. Table 3 provides a summary of the constraints and parameters necessary, together with their values, to carry out the simulation experiment of both suggested algorithms.

Table 3. Simulation parameters

Parameter	**Value**
Learning rate (ϵ)	0.001
Discount factor (γ)	0.99
Batch size	32
Replay memory size	100,000
Initial exploration rate (δ)	1.0
Maximum Episodes	1000
Maximum Steps per Episode	200
Latency Factor (α)	0.18
Energy Factor (β)	0.82
CPU Frequency of device(f_d)	2.0 GHz
The CPU frequency of the fog node(f_j)	2.5 GHz
The CPU frequency of the cloud server(f_c)	3.0 GHz
Energy efficiency of the device (η_d)	0.5
Energy efficiency of the fog node (η_j)	0.4
Energy efficiency of the cloud server (η_c)	0.3
Device Queue latency	5ms
Fog layer Queue latency	10ms
Cloud Server Queue latency	15ms
Bandwidth between the MD and fog node(B_{u2f})	0.1 W
The bandwidth between the fog node and cloud server (B_{f2c})	0.05W
Task data size (D_{Ti})	[10-500] MB
Task Workload (w_{Ti})	500 MFLOPS

5.2. The proposed algorithms convergence proof

This section evaluates the performance of the DQN proposed technique in task offloading. In the first episodes, execution times vary largely from 24 to 36 seconds, indicating that the execution times fluctuate significantly in the initial episodes to peer out with different offloading strategies. This illustrates the exploration phase of the DQN algorithm implementation, where the agent tries to experiment with different offloading strategies to make the agent understand the environment as well as the dynamics in it better. The convergence took place after episode 50, as the fitness value stabilized. This indicates that the algorithm has mainly transitioned from exploration to exploitation, having learned a nearly optimal policy for task offloading. The average execution time remained at low values and was stable between 27 and 29 seconds per episode for the majority of the proposed optimization process, as shown in Figure 4. After reaching convergence, the execution time is stable, making DQN an appropriate algorithm for real-time task offloading in dynamic environments.

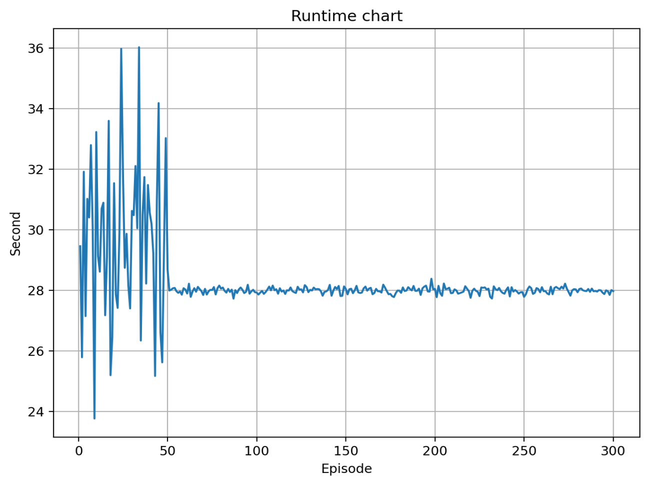


Figure 4. DQN approach execution time

Figure 5 illustrates the comparative learning curve of the DQN algorithm against the optimal strategy regarding the total cost incurred for each episode in 300 episodes. It is evident from this figure that the DQN algorithm converges in fewer episodes and thus is faster than the second strategy, mainly owing to its adaptive learning capability to thrive in complex and dynamically changing environments. The learning curve shows that costs will surely reduce because the DQN agent can converge to learning the optimal policy by iteratively improving its Q-value estimates through interactions with the system and improving user quality of experience (QoE).

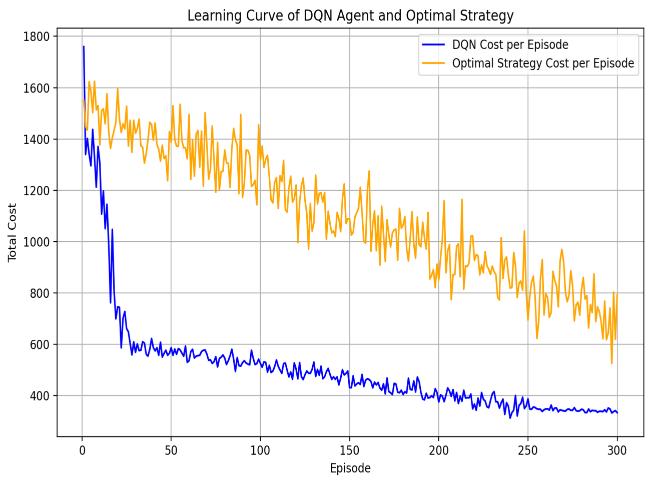


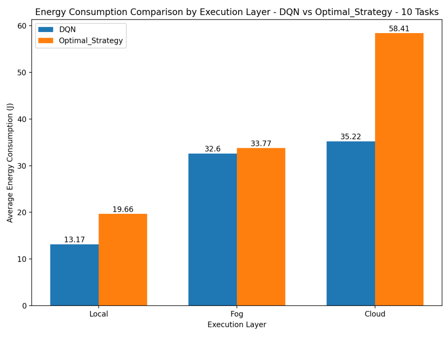
Figure 5. Optimal strategy and DQN training convergence process.

5.3. The proPosed Algorithms Comparative Analysis

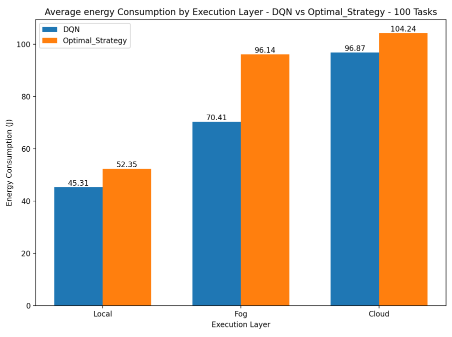
We have implemented the two algorithms that were presented in this article to evaluate their performance by comparing them with other existing algorithms in the same environment. In this section, we present a detailed comparative analysis of the proposed algorithms regarding their efficiency and performance based on various performance metrics and application scenarios with parameters mentioned in Table 3.

5.3.1. Energy consumption

Figure 6 shows the average energy consumption between the DQN technique and the optimal strategy for the local, fog, and cloud execution layers. These cases include one with 10 tasks and the other with 100 tasks. In both scenarios, the DQN strategy is generally more energy efficient at all levels of execution compared to the optimal strategy, especially in the cloud layer, where the greatest difference in energy savings is found. This high performance can be attributed to its adaptive learning capability, which enables it to dynamically optimize task-offloading decisions based on system state and workload conditions. Unlike the optimal strategy, which depends on static rules or established heuristics, the DQN uses reinforcement learning to minimize long-term cumulative energy consumption. The result indicates that the DQN algorithm achieves a significant reduction in energy consumption of 25% and 16% compared to the optimal strategy over 10 and 100 tasks, respectively. This indicates that although DQN scales well as the number of tasks increases and maintains superior energy efficiency in most cases, the choice of strategy may depend on the specific execution layer and the number of tasks.



(**a**)

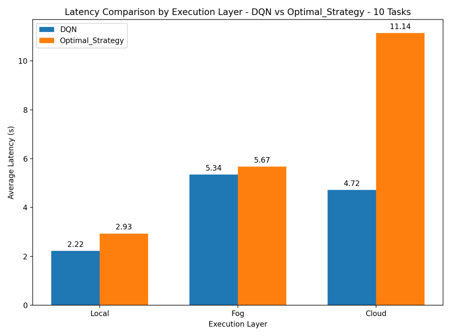


(**b**)

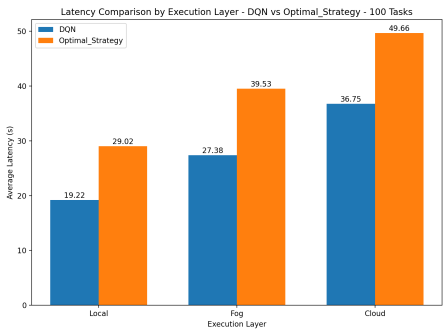
Figure 6. The average energy consumption for DQN and optimal strategy in different execution layers

5.3.2. Latency

Figure 7 illustrates the comparison of average latency between two strategies, DQN, and the optimal strategy, across three distinct execution layers. The performance comparison is done for a scenario with 10 tasks and 100 tasks. Generally, the DQN strategy offers better latency performance than the optimal strategy for various execution layers. The improvement is more pronounced at the cloud execution layer, suggesting that DQN may be particularly effective in environments with higher latency overhead. The main reason for the better performance is to explore the highly parallelizable leveraging of modern technology to process multiple states and procedures simultaneously. The optimal strategy may involve sequential algorithms, which limit its ability to execute in parallel. The results indicate that the DQN method can give an average improvement of 33% and 30% in latency compared to the optimal strategy for 10 and 100 tasks, respectively. This shows that DQN could be a better way to reduce latency through different computations.



(**a**)



(**b**)

Figure 7. The average latency for DQN and optimal strategy in different execution layers

5.4. Compared Method

On the other hand, the reference in this comparison will remain as algorithms DJA [34] and BAT [1] to our proposed optimal strategy and the DQN algorithm. Both BAT and DJA employ parameters and constraints similar to those of the proposed algorithm, but are based on different models within IoT-fog-cloud computing networks. The authors in paper [34] suggested a metaheuristic-based task offloading scheme by optimizing the task offloading based on reduced delay requirements and increased utilization of resources in a fog computing environment. The second citation represents an AI-based framework for offloading workloads and resource allocation in fog-cloud computing systems, according to the recommendation of the paper [1].

The following figure 8 compares the latency performance of four different strategies across a range of tasks from 1 to 100. The results show that the average latency ratio of all methods increases as the number of tasks increases, due to the greater contention for system resources, communication overhead, and queueing delays. The BAT algorithm adopts moderate latency, which is better than DJA but worse than DQN, and the optimal strategy because it is search-based in its approach, which performs well but may not be able to deal with dynamic changes of tasks, hence the higher latency compared to DQN. However, DJA is the algorithm that displays the most considerable latency compared to all the algorithms in this study because its deterministic heuristic approach lacks the adaptability to AI-based algorithms, resulting in suboptimal latency performance, particularly in scenarios with rapidly changing workloads or diverse IoT requests.

However, the DQN algorithm provides the minimum latency throughout, resulting in it being the most efficient strategy of them all. It maintains such low latency according to the increase in the number of tasks to justify its robustness and efficiency. The percentage of improvement in DQN is more than 30% compared to the optimal strategy and BAT, and reaches almost 40% compared with DJA. However, the optimal strategy starts with slightly more latency than the DQN algorithm but is lower than BAT and DJA.

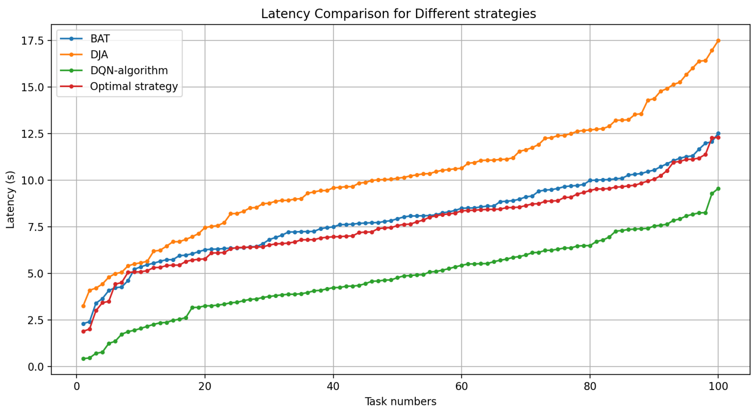


Figure 8. Latency Comparison across task offloading strategies

In Figure 9, the energy used to process tasks for each of the four techniques is shown with different task numbers. This figure indicates that when the number of tasks increases, the energy usage increases significantly. Compared to the other strategies, the DQN strategy can achieve the best overall performance. However, the optimal strategy starts to consume less or equal to the DQN algorithm for the first 30 tasks because it solves the problem using an exhaustive search or a perfect knowledge model, but is computationally impractical when the number of tasks increases. On the other hand, the BAT algorithm effectively considers a trade-off between computational resources and energy consumption. This enables it to achieve more efficient solutions than DJA, which lacks global optimization capabilities. As shown in the figure, the BAT and DJA strategies start with higher energy consumption (around 40 joules) than DQN, and the optimal strategy begins with around 20 joules. The DQN shows an improvement of 35% and 50% compared to BAT and DJA, respectively, which indicates that the proposed DQN-based strategy outperforms traditional optimization and heuristic algorithms.

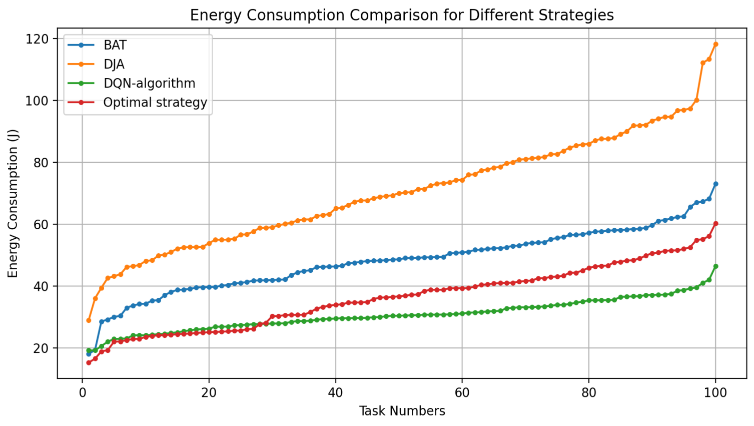


Figure 9. Energy consumption comparison across tasks offloading strategies

As seen in Figure 10, the costs of the different algorithms change based on the number of tasks. It is clear that the DQN algorithm outperforms all other approaches in performance, but above the number of tasks, the total cost of the optimal strategy decreases by 30% compared to DJA and BAT. Moreover, it is observed that the DQN strategy is 50% lower than both the BAT and the DJA strategies. This is because the efficiency of decision-making methods to deal with complex scenarios can make excellent use of the characteristics of tasks and the location of execution. DJA incurs higher costs compared to other strategies due to its dependence on static rule-based decision-making, poor resource utilization, and scalability constraints. While being dynamic, it is not capable of modulating itself with system state, resource heterogeneity, and multi-objective constraints within itself, making it different from the rest.

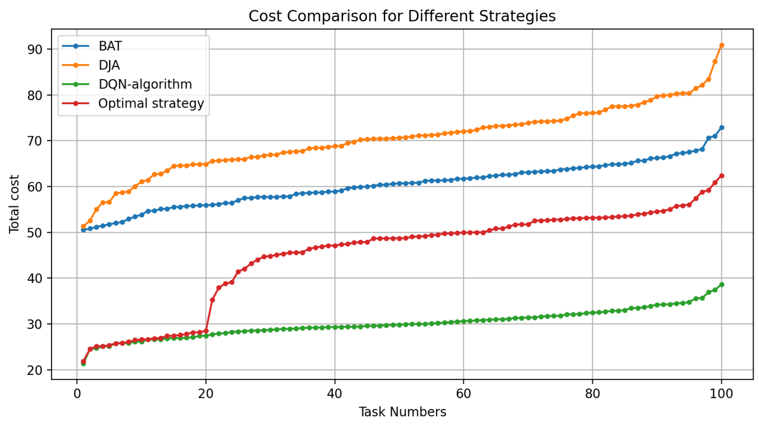


Figure 10. Total cost comparison across tasks offloading strategies

Among all the algorithms, the DQN algorithm is the clear winner in terms of energy efficiency, while the optimal strategy starts well but becomes progressively less competitive as the number of tasks increases. The BAT strategy will occupy the middle ground between the options, while DJA is the least efficient in terms of energy and latency after all.

6. Conclusions

Overall, the paper discusses relevant and well-organized studies on task offloading in hybrid IoT-fog-cloud computing networks, with a primary focus on improving latency and energy consumption performance. The suggested method uses DQN adaptive learning capabilities to optimize task offloading decisions in real time, resulting in a significant increase in system efficiency. Compared to BAT and DJA, which integrate enhanced learning and share similar constraints, the DQN strategy exhibits enhancements of 35% and 50% in energy consumption and 30% and 40% in latency.

Results indicate that latency reduction, and thereby energy savings, is possible by using the DQN-learning strategy, thus meeting the necessary trade-off, which is vital for the long life and responsiveness of IoT systems. Accordingly, this research further fills the growing corpus of knowledge on intelligent resource management in IoT and cloud computing. This article provides a solid foundation for further research and applications within this domain in the field of machine learning techniques.

Although further work should explore the use of the DQN model in real-world situations, complex application scenarios in the field of IoT should also be studied. Other aspects of machine learning that are evolving should also be studied to determine how best to complement or support this approach while maximizing the actual potential of the method in real-world applications. The results indicate that efficient and scalable IoT systems can be achieved by offloading and managing tasks using a DQN learning model, especially when supported by high-speed optical network infrastructure. This thereby enhances the user experience and robustness of the complete system.

Funding: This work was supported by the NTNU- Norwegian University of Norway and the JST ASPIRE Grant Number JPMJAP2326, Japan.

Institutional Review Board Statement: Not applicable for studies not involving humans or animals.

Informed Consent Statement: Not applicable

Data Availability Statement: Dataset available on request from the authors.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Aknan, M.; Arya, R. AI and Blockchain Assisted Framework for Offloading and Resource Allocation in Fog Computing. Journal of Grid Computing 2023, 21, 1–17, doi:10.1007/s10723-023-09694-7.

2. Safaei, B.; Mohammadsalehi, A.A.; Khoosani, K.T.; Zarbaf, S.; Monazzah, A.M.H.; Samie, F.; Bauer, L.; Henkel, J.; Ejlali, A. Impacts of Mobility Models on RPL-Based Mobile IoT Infrastructures: An Evaluative Comparison and Survey. IEEE Access 2020, 8, 167779–167829, doi:10.1109/ACCESS.2020.3022793.
3. Goudarzi, M.; Wu, H.; Palaniswami, M.; Buyya, R. An Application Placement Technique for Concurrent IoT Applications in Edge and Fog Computing Environments. IEEE Transactions on Mobile Computing 2021, 20, 1298–1311, doi:10.1109/TMC.2020.2967041.
4. Alasmari, M.K.; Alwakeel, S.S.; Alohali, Y.A. A Multi-Classifier-Based Algorithm for Energy-Efficient Tasks Offloading in Fog Computing. Sensors 2023, 23, 7209, doi:10.3390/s23167209.
5. Abdullah, S.; Jabir, A. A Lightweight Multi-Objective Task Offloading Optimization for Vehicular Fog Computing. Iraqi Journal for Electrical and Electronic Engineering 2021, 17 (1), 1–10. https://doi.org/10.37917/ijeee.17.1.8.
6. Shi, J.; Du, J.; Wang, J.; Wang, J.; Yuan, J. Priority-Aware Task Offloading in Vehicular Fog Computing Based on Deep Reinforcement Learning. IEEE Trans. Veh. Technol. 2020, 69 (12), 16067–16081. https://doi.org/10.1109/tvt.2020.3041929.
7. Alharbi, H. A.; Aldossary, M.; Jaber Almutairi; Elgendy, I. A. Energy-Aware and Secure Task Offloading for Multi-Tier Edge-Cloud Computing Systems. Sensors 2023, 23 (6), 3254–3254. https://doi.org/10.3390/s23063254.
8. Kumar, M.; Sharma, S. C.; Goel, A.; Singh, S. P. A Comprehensive Survey for Scheduling Techniques in Cloud Computing. Journal of Network and Computer Applications 2019, 143, 1–33. https://doi.org/10.1016/j.jnca.2019.06.006.
9. Jiang, Y.-L.; Chen, Y.-S.; Yang, S.-W.; Wu, C.-H. Energy-Efficient Task Offloading for Time-Sensitive Applications in Fog Computing. IEEE Systems Journal 2019, 13 (3), 2930–2941. https://doi.org/10.1109/jsyst.2018.2877850.
10. Iftikhar, S.; Gill, S. S.; Song, C.; Xu, M.; Aslanpour, M. S.; Toosi, A. N.; Du, J.; Wu, H.; Ghosh, S.; Chowdhury, D.; Golec, M.; Kumar, M.; Abdelmoniem, A. M.; Cuadrado, F.; Varghese, B.; Rana, O.; Dustdar, S.; Uhlig, S. AI-Based Fog and Edge Computing: A Systematic Review, Taxonomy and Future Directions. Internet of Things 2023, 21, 100674. https://doi.org/10.1016/j.iot.2022.100674.
11. Zhou, R.; Zhang, X.; Qin, S.; John C.S. Lui; Zhou, Z.; Huang, H.; Li, Z. Online Task Offloading for 5G Small Cell Networks. IEEE Transactions on Mobile Computing 2022, 21 (6), 2103–2115. https://doi.org/10.1109/tmc.2020.3036390.
12. Norsyafizan, W.; Mohd, S.; Kaharudin Dimyati; Muhammad Awais Javed; Idris, A.; Darmawaty Mohd Ali; Abdullah, E. Energy-Efficient Task Offloading in Fog Computing for 5G Cellular Network. Engineering Science and Technology an International Journal 2024, 50, 101628–101628. https://doi.org/10.1016/j.jestch.2024.101628.
13. Gupta, H.; Vahid Dastjerdi, A.; Ghosh, S. K.; Buyya, R. IFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in the Internet of Things, Edge and Fog Computing Environments. Software: Practice and Experience 2017, 47 (9), 1275–1296. https://doi.org/10.1002/spe.2509.
14. Wu, M.; Song, Q.; Guo, L.; Lee, I. Energy-Efficient Secure Computation Offloading in Wireless Powered Mobile Edge Computing Systems. IEEE Transactions on Vehicular Technology 2023, 72 (5), 6907–6912. https://doi.org/10.1109/tvt.2023.3236327.
15. Tseng, C.-L.; Cheng, C.-S.; Shen, Y.-H. A Reinforcement Learning-Based Multi-Objective Bat Algorithm Applied to Edge Computing Task-Offloading Decision Making. Applied Sciences 2024, 14 (12), 5088–5088. https://doi.org/10.3390/app14125088.
16. Hosny, K. M.; Awad, A. I.; Khashaba, M. M.; Fouda, M. M.; Mohsen Guizani; Mohamed, E. R. Optimized Multi-User Dependent Tasks Offloading in Edge-Cloud Computing Using Refined Whale Optimization Algorithm. IEEE Transactions on Sustainable Computing 2024, 9 (1), 14–30. https://doi.org/10.1109/tsusc.2023.3294447.
17. Song, F.; Xing, H.; Wang, X.; Luo, S.; Dai, P.; Li, K. Offloading Dependent Tasks in Multi-Access Edge Computing: A Multi-Objective Reinforcement Learning Approach. Future Generation Computer Systems 2022, 128, 333–348. https://doi.org/10.1016/j.future.2021.10.013.
18. Ma, S.; Song, S.; Yang, L.; Zhao, J.; Yang, F.; Zhai, L. Dependent Tasks Offloading Based on Particle Swarm Optimization Algorithm in Multi-Access Edge Computing. Applied Soft Computing 2021, 112, 107790. https://doi.org/10.1016/j.asoc.2021.107790.

19. Li, H.; Zhang, X.; Li, H.; Duan, X.; Xu, C. SLA-Based Task Offloading for Energy Consumption Constrained Workflows in Fog Computing. Future Generation Computer Systems 2024, 156, 64–76. https://doi.org/10.1016/j.future.2024.03.013.
20. Ale, L.; Zhang, N.; Fang, X.; Chen, X.; Wu, S.; Li, L. Delay-Aware and Energy-Efficient Computation Offloading in Mobile-Edge Computing Using Deep Reinforcement Learning. IEEE Transactions on Cognitive Communications and Networking 2021, 7 (3), 881–892. https://doi.org/10.1109/tccn.2021.3066619.
21. H. Sabireen; Venkataraman, N. A Hybrid and Light Weight Metaheuristic Approach with Clustering for Multi-Objective Resource Scheduling and Application Placement in Fog Environment. Expert Systems with Applications 2023, 223, 119895–119895. https://doi.org/10.1016/j.eswa.2023.119895.
22. Ren, J.; Yu, G.; He, Y.; Li, G. Y. Collaborative Cloud and Edge Computing for Latency Minimization. IEEE Transactions on Vehicular Technology 2019, 68 (5), 5031–5044. https://doi.org/10.1109/TVT.2019.2904244.
23. Chen, Y.; Li, W.; Huang, J.; Gao, H.; Deng, S. A Differential Evolution Offloading Strategy for Latency and Privacy Sensitive Tasks with Federated Local-Edge-Cloud Collaboration. ACM Transactions on Sensor Networks 2024. https://doi.org/10.1145/3652515.
24. Xin, J.; Li, X.; Zhang, L.; Zhang, Y.; Huang, S. Task Offloading in MEC Systems Interconnected by Metro Optical Networks: A Computing Load Balancing Solution. Optical Fiber Technology 2023, 81, 103543–103543. https://doi.org/10.1016/j.yofte.2023.103543.
25. Robles-Enciso, A.; Skarmeta, A. F. A Multi-Layer Guided Reinforcement Learning-Based Tasks Offloading in Edge Computing. Computer Networks 2023, 220, 109476. https://doi.org/10.1016/j.comnet.2022.109476.
26. Ma, L.; Wang, P.; Du, C.; Li, Y. Energy-Efficient Edge Caching and Task Deployment Algorithm Enabled by Deep Q-Learning for MEC. Electronics 2022, 11 (24), 4121. https://doi.org/10.3390/electronics11244121.
27. Zendebudi, A.; Choudhury, S. Designing a Deep Q-Learning Model with Edge-Level Training for Multi-Level Task Offloading in Edge Computing Networks. Applied Sciences 2022, 12 (20), 10664. https://doi.org/10.3390/app122010664.
28. Watkins, C. J. C. H.; Dayan, P. Q-Learning. Machine Learning 1992, 8 (3-4), 279–292. https://doi.org/10.1007/bf00992698.
29. Lu, H.; He, X.; Zhang, D. Security-Aware Task Offloading Using Deep Reinforcement Learning in Mobile Edge Computing Systems. Electronics 2024, 13 (15), 2933–2933. https://doi.org/10.3390/electronics13152933.
30. Samy, A.; Elgendy, I. A.; Yu, H.; Zhang, W.; Zhang, H. Secure Task Offloading in Blockchain-Enabled Mobile Edge Computing with Deep Reinforcement Learning. IEEE Transactions on Network and Service Management 2022, 1–1. https://doi.org/10.1109/tnsm.2022.3190493.
31. Fang, J.; Qu, D.; Chen, H.; Liu, Y. Dependency-Aware Dynamic Task Offloading Based on Deep Reinforcement Learning in Mobile Edge Computing. IEEE Transactions on Network and Service Management 2023, 1–1. https://doi.org/10.1109/tnsm.2023.3319294.
32. Wang, P.; Li, K.; Xiao, B.; Li, K. Multi-Objective Optimization for Joint Task Offloading, Power Assignment, and Resource Allocation in Mobile Edge Computing. IEEE Internet of Things Journal 2021, 1–1. https://doi.org/10.1109/jiot.2021.3132080.
33. Habtamu Mohammed Birhanie; Mohammed Oumer Adem. Optimized Task Offloading Strategy in IoT Edge Computing Network. Journal of King Saud University - Computer and Information Sciences 2024, 36 (2), 101942–101942. https://doi.org/10.1016/j.jksuci.2024.101942.
34. Kumari, N.; Jana, P. K. A Metaheuristic-Based Task Offloading Scheme with a Trade-off between Delay and Resource Utilization in IoT Platform. Cluster computing 2023. https://doi.org/10.1007/s10586-023-04193-6.
35. Kumar, D.; Baranwal, G.; Shankar, Y.; Vidyarthi, D. P. A Survey on Nature-Inspired Techniques for Computation Offloading and Service Placement in Emerging Edge Technologies. World Wide Web 2022. https://doi.org/10.1007/s11280-022-01053-y.
36. Lin, C.-C.; Deng, D.-J.; Suwatcharachaitiwong, S.; Li, Y.-S. Dynamic Weighted Fog Computing Device Placement Using a Bat-Inspired Algorithm with Dynamic Local Search Selection. Mobile Networks and Applications 2020, 25 (5), 1805–1815. https://doi.org/10.1007/s11036-020-01565-9.
37. Chicone, C. Stability Theory of Ordinary Differential Equations. Encyclopedia of Complexity and Systems Science 2009, 8630–8649. https://doi.org/10.1007/978-0-387-30440-3_516.
38. Kumar, D.; Baranwal, G.; Shankar, Y.; Vidyarthi, D. P. A Survey on Nature-Inspired Techniques for Computation Offloading and Service Placement in Emerging Edge Technologies. World Wide Web 2022. https://doi.org/10.1007/s11280-022-01053-y.
39. Tu, Y.; Chen, H.; Yan, L.; Zhou, X. Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning for Efficient Edge Computing in IoT. Future Internet 2022, 14 (2), 30. https://doi.org/10.3390/fi14020030.

40. Eldeeb, H. B.; Naser, S.; Bariah, L.; Sami Muhaidat; Murat Uysal. Digital Twin-Assisted OWC: Towards Smart and Autonomous 6G Networks. IEEE Network 2024, 38 (6), 153–162. https://doi.org/10.1109/mnet.2024.3374370.
41. Yan, P.; Choudhury, S. Deep Q-Learning Enabled Joint Optimization of Mobile Edge Computing Multi-Level Task Offloading. Computer Communications 2021, 180, 271–283. https://doi.org/10.1016/j.comcom.2021.09.028.
42. Tang, M.; Wong, V. W. S. Deep Reinforcement Learning for Task Offloading in Mobile Edge Computing Systems. IEEE Transactions on Mobile Computing 2020, 21 (6), 1–1. https://doi.org/10.1109/tmc.2020.3036871.
43. Chiang, Y.; Hsu, C.-H.; Chen, G.-H.; Wei, H.-Y. Deep Q-Learning Based Dynamic Network Slicing and Task Offloading in Edge Network. IEEE Transactions on Network and Service Management 2022, 1–1. https://doi.org/10.1109/tnsm.2022.3208776.
44. Penmetcha, M.; Min, B.-C. A Deep Reinforcement Learning-Based Dynamic Computational Offloading Method for Cloud Robotics. IEEE Access 2021, 1–1. https://doi.org/10.1109/access.2021.3073902.
45. Li, J.; Yang, Z.; Chen, K.; Ming, Z.; Cheng, L. Dependency-Aware Task Offloading Based on Deep Reinforcement Learning in Mobile Edge Computing Networks. Wireless Networks 2023, 30 (6), 1–13. https://doi.org/10.1007/s11276-023-03283-y.