electronics

MDPI

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 02310, Algeria
[2] LIA laboratory, Department of Computer Science, FSEI, Hassiba Ben Bouali University of Chlef, Chlef 02310, Algeria; r.bechar@univ-chlef.dz (R.B.); w.kadri@univ-chlef.dz (W.K.)
[3] Faculty of Information Technology and Electrical Engineering, NTNU- Norwegian University of Science and Technology, 7491 Trondheim, Norway
[4] Faculty of Engineering Science and Technology, UiT- The Arctic University of Norway, 8514 Narvik, 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; ssahmoud@fsm.edu.tr
[7] Data Science Application and Research Center (VEBIM), Fatih Sultan Mehmet Vakif University, Istanbul 34015, Turkey
* Correspondence: a.benaboura@univ-chlef.dz (A.B.); tu.d.ho@ntnu.no (T.D.H.)

Abstract

The exponential proliferation of the Internet of Things (IoT) and optical IoT (O-IoT) has introduced substantial challenges concerning computational capacity and energy efficiency. IoT devices generate vast volumes of aggregated data and require intensive processing, often resulting in elevated latency and excessive energy consumption. Task offloading has emerged as a viable solution; however, many existing strategies fail to adequately optimize both latency and energy usage. This paper proposes a novel task-offloading approach based on deep Q-network (DQN) learning, designed to intelligently and dynamically balance these critical metrics. The proposed framework continuously refines real-time task offloading decisions by leveraging the adaptive learning capabilities of DQN, thereby substantially reducing latency and energy consumption. To further enhance system performance, the framework incorporates optical networks into the IoT–fog–cloud architecture, capitalizing on their high-bandwidth and low-latency characteristics. This integration facilitates more efficient distribution and processing of tasks, particularly in data-intensive IoT applications. Additionally, we present a comparative analysis between the proposed DQN algorithm and the optimal strategy. Through extensive simulations, we demonstrate the superior effectiveness of the proposed DQN framework across various IoT and O-IoT scenarios compared to the BAT and DJA approaches, achieving improvements in energy consumption and latency of 35%, 50%, 30%, and 40%, respectively. These findings underscore the significance of selecting an appropriate offloading strategy tailored to the specific requirements of IoT and O-IoT applications, particularly with regard to environmental stability and performance demands.

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

Academic Editor: Hung-Yu Chien

Received: 3 June 2025
Revised: 22 July 2025
Accepted: 27 July 2025
Published: 1 August 2025

Citation: Benaboura, A.; Bechar, R.; Kadri, W.; Ho, T.D.; Pan, Z.; Sahmoud, S. Latency-Aware and Energy-Efficient Task Offloading in IoT and Cloud Systems with DQN Learning. *Electronics* **2025**, *14*, 3090. https://doi.org/10.3390/electronics14153090

1. Introduction

The Internet of Things (IoT) has become an integral component of modern life due to its wide range of applications and its ability to interconnect various devices for communication and information exchange [1]. Optical IoT (O-IoT) is a new subset of IoT that uses optical communication technology to allow high-bandwidth, high-speed data transfer between connected devices, improving network scalability and overall performance [2]. IoT and O-IoT technologies are increasingly being deployed across multiple sectors, including healthcare, industrial automation, smart homes, and emergency response systems. The widespread connectivity enabled by the IoT has led to a surge in data generation, commonly referred to as data explosion. According to the International Data Corporation (IDC), by 2025, the number of connected IoT devices is projected to exceed 41 billion, collectively generating more than 79 ZB of data annually [3]. This figure is expected to escalate to 500 billion devices by 2030. However, mobile devices (MDs) contributing to this ecosystem often suffer from inherent limitations such as constrained computational resources, limited battery life, and insufficient storage capacity [4], necessitating robust data management platforms. Moreover, the increasing demand for services with low latency, high fairness, and enhanced spectral and energy efficiency, such as interactive real-time gaming, virtual reality (VR), and augmented reality (AR), further accentuates the need for optimized system architectures [5].

Cloud computing represents a cost-effective and efficient solution for executing and storing massive volumes of data generated by mobile devices (MDs) [6]. It enables on-demand resource provisioning for a wide range of services, eliminating the need for additional hardware or software infrastructure. Furthermore, computationally intensive applications and algorithms can be deployed on the cloud, allowing MDs to offload heavy processing tasks. However, the geographical distance between cloud data centers and end devices can introduce latency and degrade service quality, particularly in time-sensitive applications such as online gaming and video streaming [7]. Consequently, a new computing paradigm is required to support latency-sensitive applications, which demand real-time data processing with minimal delay, in order to ensure high performance and effectively meet stringent quality of service (QoS) requirements.

For services requiring instantaneous data processing, fog devices offer the computational and storage capabilities of cloud computing while significantly reducing latency and maximizing bandwidth utilization. The deployment of a fog node (FN) has emerged as a practical and innovative solution within the fog computing paradigm, particularly for supporting real-time applications [1]. In many use cases, mobile devices (MDs) undertake substantial computational tasks; however, due to their constrained energy, processing, and storage capacities, these tasks must be offloaded to nearby FNs [8]. This architectural shift enables data to be processed locally, closer to the data source, rather than being routinely transmitted to distant cloud servers. By leveraging the processing and storage capabilities of fog infrastructure, latency can be minimized and bandwidth can be optimized for applications that demand real-time responsiveness and intensive computational resources [9,10]. Furthermore, the integration of optical networks presents a promising solution for enabling frequent resource scheduling and seamless information exchange among servers, thereby enhancing the responsiveness and efficiency of both fog and cloud environments [11]. To ensure optimal performance, advanced resource management and task scheduling techniques are essential for effectively operating fog devices and controlling the offloading of computational tasks [12].

Despite the advantages of fog computing in terms of reduced resource consumption compared to remote cloud computing, task offloading to the cloud remains a prevalent practice. Therefore, within the IoT, fog, and cloud paradigms, efficient mechanisms are

required to determine when tasks should be offloaded and to ensure their accurate allocation to the most suitable computing resources [6]. Moreover, integrated cloud–fog networks face several critical challenges, including network latency, communication protocols, bandwidth capacity, operational costs, security, offloading strategies, and authentication mechanisms [13]. These factors significantly influence key quality of service (QoS) metrics such as energy efficiency, execution time, and service cost. As discussed in [14–18], task requests can be offloaded to fog nodes (FNs) or cloud servers instead of being executed on local devices, thereby reducing processing time and conserving energy.

Motivated by these challenges, this work proposes a latency-aware and energy-efficient task offloading approach in fog and cloud computing networks using deep Q-network (DQN) learning. The study focuses on large-scale IoT devices and fog nodes (FNs) interconnected via optical networks, which form a crucial component of the task offloading architecture [19,20]. Our approach leverages the adaptive learning capabilities of deep reinforcement learning (DRL) to enable IoT devices to make optimal, real-time task offloading decisions, thereby enhancing system performance with respect to latency, energy consumption, and resource efficiency [21].

The primary objective of this research is to introduce a novel task offloading methodology for IoT–fog–cloud computing environments, with explicit consideration of both energy efficiency and latency constraints. The DQN model is deployed within mobile devices (MDs) to support intelligent offloading decisions. The key contributions of this paper are summarized as follows:

- A task offloading problem is formulated within a collaborative IoT–fog–cloud framework, accounting for real-world constraints such as device heterogeneity and fluctuating network conditions.
- An energy-efficient and latency-aware algorithm based on DQN learning is proposed to optimize task offloading decisions. The method provides a structured approach to assigning tasks across different layers of the computing hierarchy.
- The proposed model is evaluated and validated through comprehensive simulations, demonstrating its effectiveness in enhancing QoS metrics, particularly in reducing energy consumption and application latency when compared with existing approaches.

The remainder of this paper is organized as follows. Section 2 provides a review of the related literature. Section 3 introduces the problem formulation, including the objective function and the system model. Section 4 presents the proposed algorithm along with its corresponding pseudocode. Section 5 evaluates the performance of the proposed algorithm and discusses the simulation results. Finally, Section 6 concludes the paper and outlines directions for future research.

2. Related Works

Several studies have addressed optimization challenges associated with task offloading in cloud and fog computing networks [22]. Among the primary concerns impacting quality of service (QoS) metrics—such as service quality, availability, and reliability—are energy consumption and latency. This section reviews recent and significant contributions in the literature, highlighting their methodologies, limitations, and distinctions from the approach proposed in this paper.

In [23], the authors provide an overview of contemporary research in the cloud computing domain, introducing a refined whale optimization algorithm (RWOA) to support offloading of latency-sensitive, computationally intensive, and energy-demanding tasks. Tasks may be partitioned across multiple processing layers, including local, mobile edge computing (MEC), and cloud platforms [24]. Moreover, numerous users can offload their workloads to fog and cloud servers simultaneously [25]. The authors of [1] propose an

intelligent framework for selecting optimal resources to support latency-sensitive IoT applications, utilizing a metaheuristic BAT algorithm to optimize service cost, energy consumption, and latency. Their bat-inspired evolutionary approach dynamically explores optimal resources in a collaborative fog–cloud environment, based on food-searching behavior [26].

In [27], a joint optimization approach using deep Q-learning is proposed for task offloading at both the device and edge levels. Chiang et al. [28] enhance resource allocation and dynamic reconfiguration of network slices in a multi-tenant edge computing system using Q-learning-based deep learning. Their work focuses on jointly addressing dynamic slice scaling and task offloading to maximize service provider (SP) profits in multi-tenant edge computing (EC) environments. Similarly, in [29], a deep reinforcement learning (DRL)-based scheme is introduced for jointly offloading tasks with interdependencies. The authors model task dependencies using directed graphs with discrete loops and formulate the offloading problem as minimizing the average cost of energy and time (CET) for users.

To support efficient offloading in UAV-assisted MEC systems, a computation offloading optimization algorithm based on deep deterministic policy gradient (DDPG) is proposed in [30], which accounts for both task characteristics and communication environments. In [31], a fuzzy logic-based task scheduler is introduced to make informed offloading decisions by evaluating task properties and identifying the most suitable processing layer—whether locally, on collaborative fog nodes, or in the cloud.

In the healthcare domain, [32] presents a distributed collaborative dependency task offloading strategy based on deep reinforcement learning (DCDO-DRL), specifically designed to address challenges associated with radiomic-based medical imaging diagnostic model (RIDM) tasks. Furthermore, Zhang et al. [33] apply the Stackelberg game-theoretic paradigm to improve resource allocation in cooperative intelligent transportation systems. Stackelberg game theory has proven effective in hierarchical decision-making scenarios involving multiple stakeholders and constrained resources, as evidenced by the aforementioned studies.

Table 1 summarizes 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.

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.
[6]	Minimize energy	MCEETO	local devices, fog, and cloud.
[34]	Minimize latency	Edge-ward	local devices, edge, fog, and cloud.
[35]	Minimize energy	SEE	local devices and MEC.
[29]	Minimize network latency and energy	MOBA-CV-SARSA	local device and edge server.
[23]	Minimize latency, energy and cost	RWOA	Local device, MEC, and cloud.
[36]	Minimize energy consumption	MSTEC and HREC	Local device, fog, and cloud.
[37]	Minimize energy consumption	DRL-based	Local device, fog, and cloud.
[38]	Minimize cost and latency	PSO	Local device, fog, and cloud.

Table 1. *Cont.*

Reference	Objectives	Proposed Solution	Executions Locations
[39]	Minimize latency	Collaborative cloud–edge scheme	Local device, edge, and cloud.
[40]	Minimize latency	federated learning-based	Local device, edge, cloud.
[11]	Minimize delay	TD-based CLB-TO and GA-based CLB-TO	Local device, edge servers connected by optical network
[17]	Minimize latency	Scheduling and queue management algorithms	Local device, edge, and cloud.
[27]	Minimize latency and energy	Deep Q-learnin	Local device and edge.
[41]	Minimize latency	LSTM and dual DQN	Local device and edge.
[28]	Minimize latency and energy	DRL-based	Local device and edge.
[30]	Minimize latency	DDPG-based	Local devices, fog, and cloud.
[31]	Minimize latency, energy consumption, network usage	fuzzy logic-based	Local devices, fog, and cloud.
[33]	Minimize latency,	RL-based	Local devices, fog, and cloud .
Our	Minimize latency, energy, and cost	DQN-based	Local device, fog, and cloud.

2.1. Energy-Aware Task Offloading

In recent years, energy-efficient task offloading has gained increased attention in the context of cloud and fog computing. In [6], Alasmari, M. K., et al. addressed this challenge by proposing an intelligent allocation technique aimed at optimizing energy consumption in IoT devices. The study introduces the Multi-Classifier for Energy Efficient Tasks Offloading (MCEETO) algorithm, which selects the most suitable fog devices for task placement using a multi-classifier-based decision mechanism. The MCEETO algorithm demonstrates significant effectiveness in reducing energy consumption and emphasizes the critical role of energy conservation within fog computing environments. The authors compare the performance of MCEETO against two baseline strategies: the edge-ward and cloud-only approaches. The edge-ward strategy represents an edge-focused task placement approach, while the cloud-only method assumes that all application modules are executed in centralized cloud data centers [34].

In another study, the authors of [35] introduced a secrecy energy efficiency (SEE) model by optimizing transmission power, time allocation, and task partitioning under both energy and secrecy constraints. They examined secure offloading in wireless-powered MEC systems and proposed a physical-layer security-enabled scheme to enhance confidentiality during task offloading.

Similarly, in [36], a low-delay scheduling algorithm—Minimal Schedule Time with Energy Constraint (MSTEC)—was proposed to support energy-constrained fog computing workflows. To further improve system reliability under energy limitations, a High Reliability with Energy Constraint (HREC) algorithm was also presented for managing fog computing workflows. In another contribution, Ale et al. [37] proposed a deep reinforcement learning (DRL)-based approach to determine the optimal number of resources and select the most suitable edge server for executing offloaded tasks. Their objective was to minimize energy consumption while maximizing the number of successfully completed tasks.

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. [38] 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. [34] 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 [39]. However, the goal of using a collaborative cloud and edge computing method is to reduce the delay. Tang et al. [28] 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. [40] 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 [42] 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 [43] 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 [11] 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 have good performance on the objective targets, but these works need to balance contradictory goals. However, many studies overlook the collaborative role of fog and cloud computing networks within the optical network or without it [11,27–29,35,41]. Meanwhile, most studies that consider the collaboration of fog and cloud have failed to consider latency, energy, and cost simultaneously [6,16,34,35,38–41], 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, including the unique characteristics of Optical IoT environments. 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, including O-IoT deployments.

3. System Model and Problem Formulation

This section introduces a three-level model that captures various task scenarios. First, each component of the model is described in detail. Subsequently, the task offloading model is presented, providing an overview of the proposed framework. Finally, the objective function underlying the research is formulated and discussed.

3.1. System Model

This study adopts a three-layer collaborative model comprising the device, fog, and cloud layers, as illustrated in Figure 1. The first layer, also referred to as the infrastructure layer, consists of IoT and O-IoT devices represented by mobile devices (MDs) that either process tasks locally or offload them to higher layers. This layer is characterized by the lowest latency but also limited computational capacity. The second layer—the fog layer—is composed of multiple fog nodes (FNs) geographically distributed and interconnected via an optical network [44], providing intermediate latency and computational capabilities [6]. FNs function as small-scale servers offering localized processing power for both conventional IoT and Optical IoT applications. The third layer corresponds to the cloud layer, which exhibits relatively higher latency but offers substantial computational capacity, supported by large-scale data centers housing numerous high-performance servers [45]. Communication between these layers occurs over both wired and wireless links, with each connection tier allocated a defined maximum bandwidth to ensure efficient data transmission.

Our proposed system consists of a network of mobile devices (MDs) responsible for collecting heterogeneous data requiring computational processing. Table 2 summarizes the key notations used throughout this paper.

Each MD may complete a computing task (T_i) where $T_i \epsilon \{ T_1, T_2, \ldots, T_N \}$, 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} [22].

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

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_{Ti}^{k}	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.
$P_{d,Ti}^{loc}$	Time processing of task T_i locally.
$L_{d,Ti}^{loc}$	The total latency of processing task T_i.
$E_{d,Ti}^{loc}$	Energy consumption of task T_i that processing locally.
$C_{d,Ti}^{loc}$	Overall cost of task T_i that processing locally.
$T_{d,Ti}^{f}$	The transmission time of task T_i to fog layer.
$P_{d,Ti}^{f}$	The processing time of task T_i in fog layer.
$L_{d,Ti}^{f}$	The total latency of processing task T_i in fog layer.
$E_{d,Ti}^{f}$	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.
$C_{d,Ti}^{f}$	Cost of processing task T_i over fog layer.
$T_{d,Ti}^{c}$	The transmission time of task T_i to cloud server.
$P_{d,Ti}^{c}$	The processing time of task T_i in cloud server.
$L_{d,Ti}^{c}$	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.
$C_{d,Ti}^{c}$	Cost of processing task T_i over cloud server.

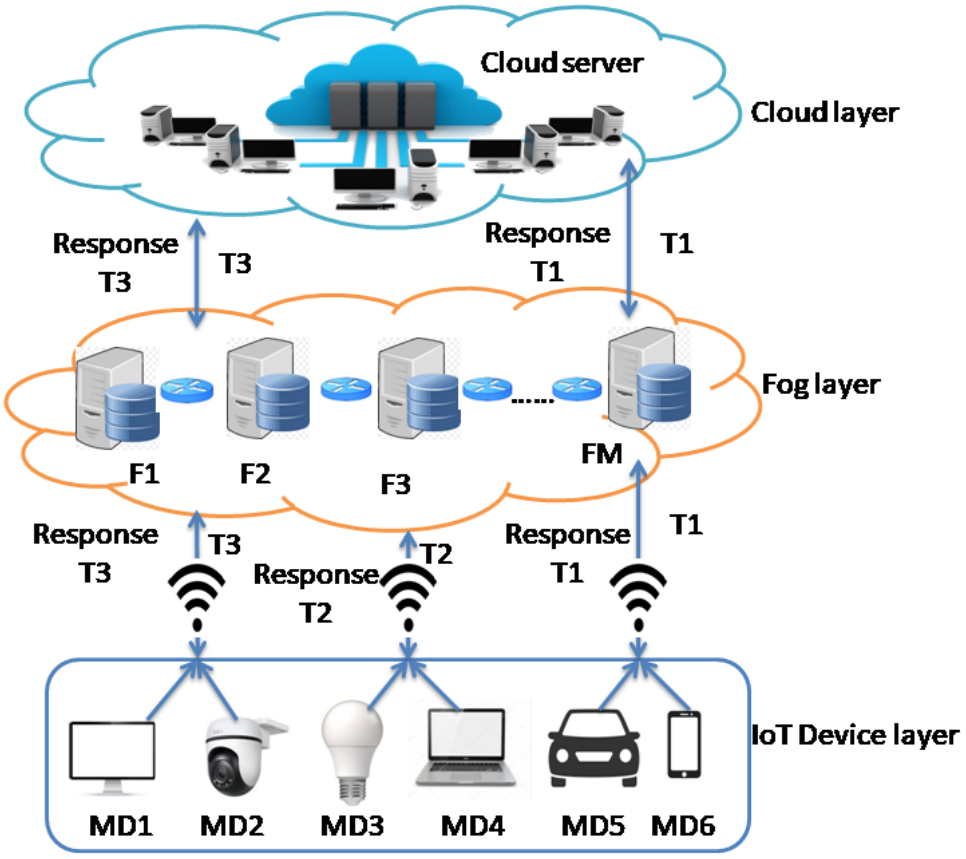


Figure 1. Overview of IoT–fog–cloud architecture.

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 [46]. Also, the decision matrix $X_{Ti}^{K} \epsilon$ {0,1} is introduced, which indicates the choice to offload task i, which, in turn, 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 ($X_{Ti}^{loc} = 1$), fog node execution by ($X_{Ti}^{f} = 1$), and cloud execution by ($X_{Ti}^{C} = 1$).

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:

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

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:

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

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 follows:

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

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 follows:

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

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:

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

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:

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

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):

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

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 follows:

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

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):

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

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.

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

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:

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

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:

$$C^c_{d,Ti} = \alpha \times E^c_{d,Ti} + \beta \times L^c_{d,Ti}. \tag{17}$$

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:

$$cost = \sum_{i=1}^{N} (X^{loc}_{Ti} \times C^{loc}_{d,Ti} + X^f_{Ti} \times C^f_{d,Ti} + X^c_{Ti} \times C^c_{d,Ti}). \tag{18}$$

3.4. Problem Formulation

In this section, we formulate the problem for a multi-mobile device (MD) scenario with multiple generated tasks, integrating the previously described models. Our objective is to minimize the overall cost by considering several quality of service (QoS) factors, including task data size, communication bandwidth—particularly leveraging the high throughput capabilities of optical networks—transmission and processing times, as well as the maximum tolerable delay for task execution.

This cost function accounts for the total execution cost incurred when tasks are processed locally, at the fog layer, or in the cloud, encompassing computation, communication, and data size components. By enabling efficient task offloading, the proposed method aims to reduce both latency and energy consumption through optimal offloading decisions supported by optical network infrastructures.

Unlike many prior approaches that rely on static allocation or simplistic heuristics, our DQN-based strategy continuously learns from the environment to iteratively improve offloading policies. Furthermore, in contrast to existing deep reinforcement learning (DRL)-based frameworks that typically focus on fog–cloud cooperation or assume simplified network models, our approach jointly optimizes resource allocation across local, fog, and cloud layers. It explicitly incorporates practical constraints such as task deadlines, energy budgets for cloud data transmission, bandwidth availability between MDs and fog nodes (FNs) enhanced by optical networks, and queue stability at each layer. By integrating these realistic considerations, the proposed method achieves substantial reductions in system latency and energy consumption while ensuring strict QoS compliance, thereby outperforming conventional offloading schemes.

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

$$Min(cost) = min \sum_{i=1}^{N} (X_{Ti}^{loc} \times C_{d,Ti}^{loc} + \sum_{F=1}^{M} (X_{Ti}^{f} \times C_{d,Ti}^{f}) + X_{Ti}^{c} \times C_{d,Ti}^{c}), \quad (19)$$

s.t.

C1: $X_{Ti}^{loc} \epsilon \{0,1\}$

C2: $X_{Ti}^{f} \epsilon \{0,1\}$

C3: $X_{Ti}^{c} \epsilon \{0,1\}$

C4: $X_{Ti}^{loc} + X_{Ti}^{f} + X_{Ti}^{c} = 1 \ \ \forall i \epsilon N$

C5: $B_{Ti} > 0 \ \ \forall i \epsilon N$

C6: $X_{Ti}^{loc} \times L_{d,Ti}^{loc} + X_{Ti}^{f} \times L_{d,Ti}^{f} + X_{Ti}^{c} * L_{d,Ti}^{c} <= \tau_{Ti} \ \forall i \epsilon N$

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 three 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 section introduces a task offloading paradigm that employs two distinct methods to address the offloading problem. The first method is the optimal strategy, wherein data from each layer is collected and analyzed to identify the most suitable offloading decisions for each task. This strategy exhaustively evaluates all possible offloading options—local, fog, and cloud—and selects the one minimizing the overall cost. While this approach delivers sub-optimal yet respectable performance, it serves as a benchmark against which alternative methods can be assessed. However, the optimal strategy requires the acquisition of substantial real-time data, which is often impractical to obtain in real-world deployments.

Conversely, we propose the use of reinforcement learning (RL) algorithms, specifically the deep Q-network (DQN) model, to derive near-optimal solutions. This approach enables the agent to learn from experience by storing action–state values in a Q-table. At each time step t, upon observing the current state S_t, the agent selects an action a_t and transitions to the subsequent state s_{t+1}, receiving a corresponding reward r_{t+1} [47]. Through this iterative process, the agent continually refines its decision-making policy based on accumulated experience.

4.1. Optimal Task Offloading Strategy

This section presents a task offloading paradigm focused on determining the optimal execution location for tasks generated by mobile devices (MDs) within IoT, Optical IoT, and optical network environments. As previously noted, this strategy employs an algorithm to solve the optimization problem, yielding sub-optimal solutions with minimal processing costs.

The algorithm determines the processing location for each generated task by evaluating the cost associated with all possible execution alternatives—local, fog, and cloud—and selecting the option that minimizes this cost. Algorithm 1 provides a detailed description of the complete procedure.

The process begins when a device generates a task as input, and the algorithm subsequently gathers data from the MD, fog nodes (FNs), and cloud servers. For each task, the algorithm considers the characteristics of the underlying communication infrastructure, with particular emphasis on the high-capacity, low-latency advantages provided by optical networks when tasks are offloaded to remote fog nodes or the cloud. The execution location is then assigned based on the minimum computed cost: (0) denotes local execution, (-1) represents cloud execution, and values ranging from 1 to 10 correspond to execution at one of the fog nodes (assuming a total of 10 fog nodes). When the selected location differs from (0) and (-1), the value indicates the specific fog node responsible for executing the task via the most cost-effective execution path, which is typically influenced by the availability of enhanced bandwidth afforded by optical network links and the deployment of O-IoT components [48].

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.

Algorithm 1 Optimal task offloading strategy

Input Mobile_devices MD, Fog_node F, Cloud_server C, Tasks T.
Output Execution_Location EL, Min_Cost MC.

Function Optimal Strategy:
for Each d in MD **do**
  for Each Task in T **do**
    Initialiser(EL, MC)
    //Local Execution
    MC= $C^{loc}_{d,Ti}$ //Using Equation (7).
    EL = 0
    // Fog Execution
    for Each Fog_node in F **do**
      Compute $C^{f}_{d,Ti}$ //Using Equation (12)
      if ($C^{f}_{d,Ti}$ < MC) **then**
        MC = $C^{f}_{d,Ti}$
        EL= Fog_Node
      end if
    end for
      //Cloud Execution
      Compute $C^{c}_{d,Ti}$ //Using Equation (17)
    if ($C^{c}_{d,Ti}$ < MC) **then**
      MC = $C^{c}_{d,Ti}$
      EL = -1
      Compute cost(d, T_i) //Using Equation (18)
    end if
  end for
end for

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 follows:

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

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:

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

where $a_t = 0$ denotes local processing, $a_t = -1$ denotes cloud processing, and $a_t = 1 \ldots$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 in the cloud layer. However, the primary purpose of this paper is to minimize both latency (denoted by L) and energy consumption (denoted by E) [49]. 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 [50]. The reward function is given as follows, based on the above considerations.

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

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

Based on our preliminary experiments, we evaluated multiple combinations of the weighting parameters α and β, and determined that the values $\alpha = 0.18$ and $\beta = 0.82$ provide a well-balanced trade-off between communication and computation costs. This balance ensures a low overall system cost while satisfying task deadline constraints. Specifically, increasing α places greater emphasis on local computation costs, potentially leading to higher energy consumption at the device level but reduced network utilization. Conversely, increasing β prioritizes offloading tasks to cloud or fog resources, which may reduce local energy consumption but incur higher transmission costs and possible delays.

The deep Q-network (DQN) model employs a neural network to approximate the Q-value function, which assesses both immediate costs and expected future rewards based on the quality of the offloading action and the current system state. The network is trained through continuous interaction with the IoT–fog–cloud environment: it makes offloading decisions, executes corresponding actions (e.g., task completion time, energy consumption), and refines its policy based on observed performance metrics (QoS indicators). Over time, the DQN agent successfully identifies an optimal offloading strategy that balances latency and energy consumption, dynamically adapting to the evolving conditions of the IoT–fog–cloud environment, including real-time fluctuations in optical network status and resource availability within O-IoT scenarios. This adaptive approach is especially effective in complex and dynamic settings where traditional static methods struggle to maintain optimal performance.

Initially, the agent observes the environment state (depicted as the "Initial State"), which includes the status of infrastructure components (fog/cloud availability, network conditions) as well as task and device parameters (computational requirements, energy constraints). Subsequently, the agent selects an offloading action—local, fog, or cloud—that maximizes the predicted cumulative reward based on its Q-value function, represented either by a Q-table or neural network. After executing the chosen action, the environment provides immediate feedback in the form of cost-based rewards, reflecting actual performance outcomes such as latency and energy utilization.

Using the updated system state and received reward, the agent applies temporal difference learning to update its policy (π), effectively balancing short-term costs and long-term benefits. Through repeated interactions (illustrated by the cyclic arrows in Figure 2), the agent progressively enhances its decision-making, ultimately achieving context-aware, adaptive offloading performance. An overview of the agent utilizing the proposed DQN method is presented in Figure 2.

Within the Q-learning framework [51], the agent receives rewards based on the effectiveness of actions taken to transition from one state to the next. The objective of Q-learning is to minimize the cumulative cost (or maximize cumulative rewards) accumulated over the execution of the algorithm by selecting actions from a feasible set at each state. In its simplest form, Q-learning stores a value for every state-action pair in a Q-table [52], which estimates the expected cumulative reward for performing a particular action in a given state. The Q-values are updated iteratively whenever an action a_t is executed in state s_t. We denote the sequence of visited states as $(s_1, s_2, \ldots)$ and the corresponding actions as $(a_1, a_2, \ldots)$.

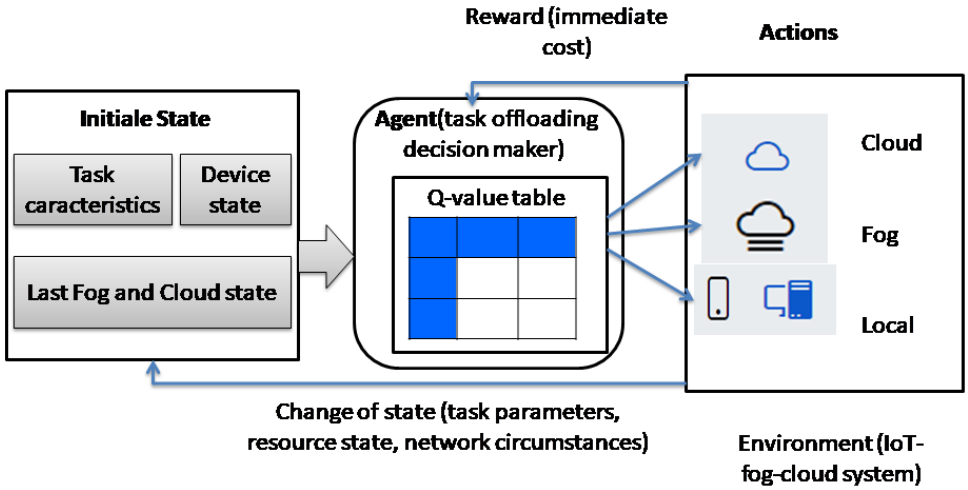


Figure 2. The process of task offloading decision of RL algorithms.

The DQN-based task offloading strategy is summarized in Algorithm 2.

Algorithm 2 DQN-based task offloading strategy

Input Tasks T, learning rate(ϵ), discount factor (γ).
Output Optimal offloading decision and total cost.

Function DQN Strategy:
Initialize replay memory D to capacity N.
Initialize total episode reward r = 0.
for Each episode **do**
  Reset environment to initial state s.
  for Each step **do**
    Observe actual state s_t.
    Determine feasible action.
    random = randomly choose from [0, 1]
    if (random < ϵ) **then**
      a_t = randomly select action from {0,1,2}
    else
      $a_t = \gamma * max_{action}$ Q(s_t, a_t).
    end if
    Execute action a_t.
    Calculate reward r_t using Equation (22) .
    Observe next state s_{t+1}.
    Store (s_t, a_t, r_t, s_{t+1}) in replay memory.
    Update $Q(s_t, a_t)$ according to Equation (23)
  end for
end for

The DQN architecture employed to implement the proposed algorithm consists of two neural networks: the evaluation network and the target network. The evaluation network estimates Q-values—i.e., the expected future rewards for each possible action—and is updated frequently during training. In contrast, the target network provides stable Q-value targets and is synchronized with the evaluation network at fixed intervals to stabilize learning. Both networks share a similar architecture composed of three hidden layers, each containing 64 neurons. The hidden layers utilize the ReLU activation function, while the output layer employs a linear activation to enable precise Q-value predictions.

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 [53]:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \epsilon(r_t + \gamma \times a_t max Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)), \quad (23)$$

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

This section presents comprehensive simulation experiments to evaluate the performance of the proposed strategy. The simulations were implemented in Python 3.19.13 on a 64-bit Windows operating system, equipped with an Intel(R) Core(TM) i5-6300U CPU running at 2.50 GHz and 8 GB of RAM. The computer used is an HP EliteBook 840 G3, manufactured by Hewlett-Packard (USA), sourced from Germany. The simulation parameters are summarized in Table 3.

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	5 ms
Fog layer Queue latency	10 ms
Cloud Server Queue latency	15 ms
Bandwidth between the MD and fog node (B_{u2f})	0.1 W
The bandwidth between the fog node and cloud server (B_{f2c})	0.05 W
Task data size (D_{Ti})	[10–500] MB
Task Workload (w_{Ti})	500 MFLOPS

5.1. Simulation Model

In the simulation scenario, 100 IoT devices receive randomly generated datasets with varying sizes and resource requirements. The environment also includes 10 fog nodes and a single cloud server. For training purposes, a batch size of 32 samples is used for both fog nodes and IoT devices. Table 3 lists the key constraints and parameter values employed to conduct the experiments for both proposed algorithms.

5.2. The Proposed Algorithm's Convergence Analysis

This section evaluates the performance of the proposed DQN-based task offloading technique with respect to convergence behavior. During the initial episodes, execution times exhibit considerable variability, ranging from 24 to 36 s. This fluctuation reflects the exploration phase of the DQN algorithm, wherein the agent experiments with various offloading strategies to better understand the environment and its dynamics.

Convergence is observed after approximately episode 50, as indicated by the stabilization of the fitness value. This suggests that the algorithm has effectively transitioned from exploration to exploitation, having learned a near-optimal policy for task offloading.

Following convergence, the average execution time remains consistently low and stable, oscillating between 27 and 29 s per episode throughout the remainder of the optimization process, as illustrated in Figure 3. The stable execution time post-convergence demonstrates that the DQN algorithm is well-suited for real-time task offloading in dynamic environments.

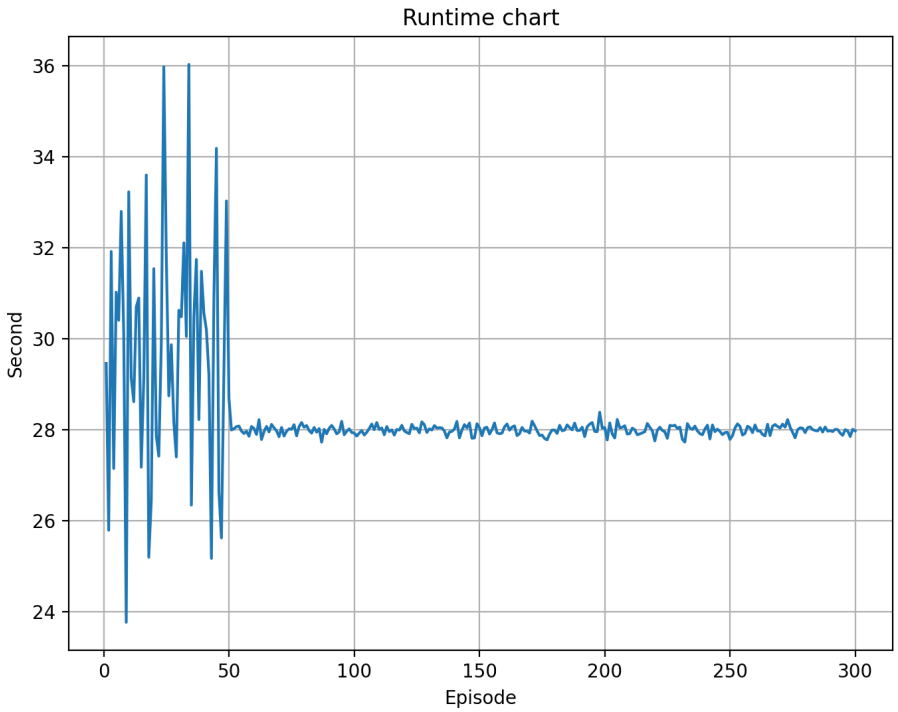


Figure 3. DQN approach execution time.

Figure 4 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 decrease 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).

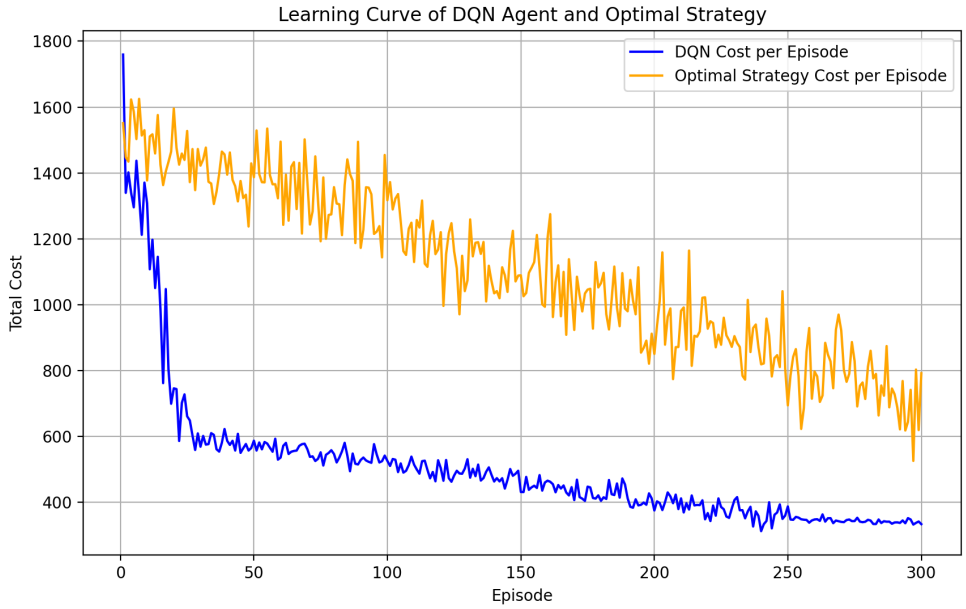


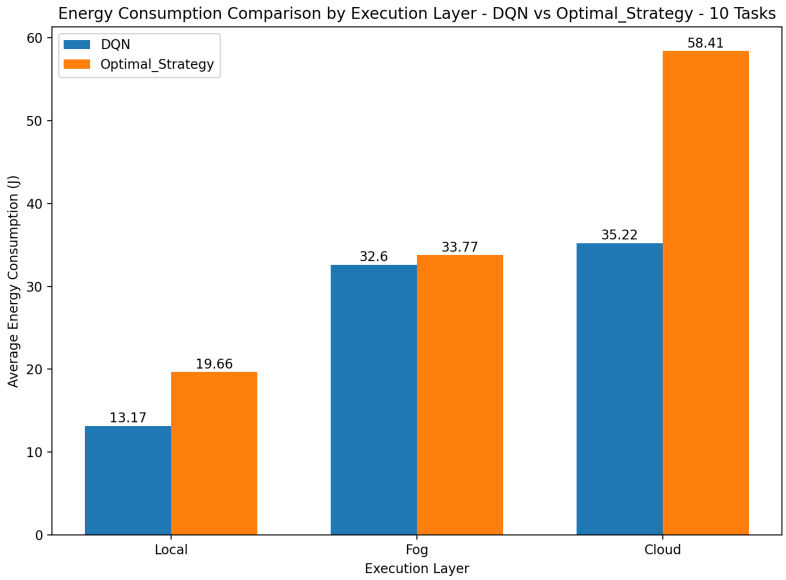
Figure 4. Optimal strategy and DQN training convergence process.

5.3. The Proposed Algorithms Comparative Analysis

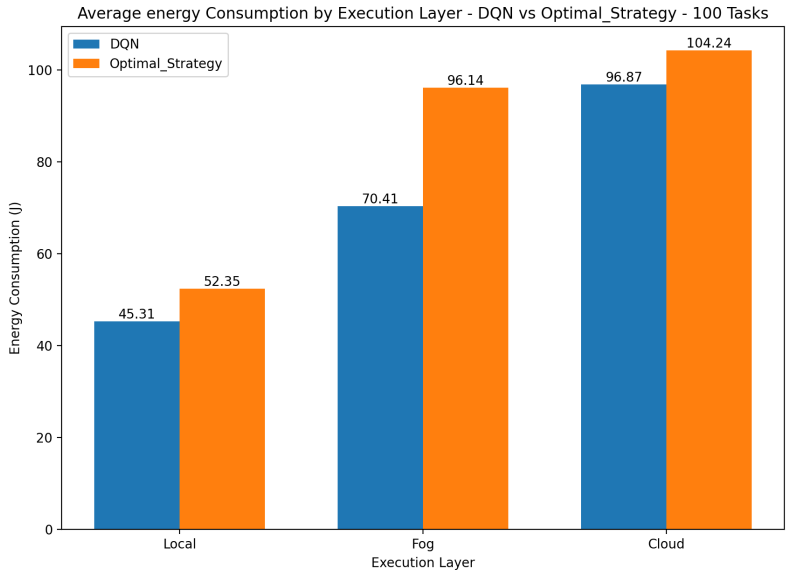
We have implemented the two algorithms that were presented in this article to evaluate their performance by comparing them with other existing algorithms in the same environment. In this section, we present a detailed comparative analysis of the proposed algorithms regarding their efficiency and performance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

5.3.1. Energy Consumption

Figure 5 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**)

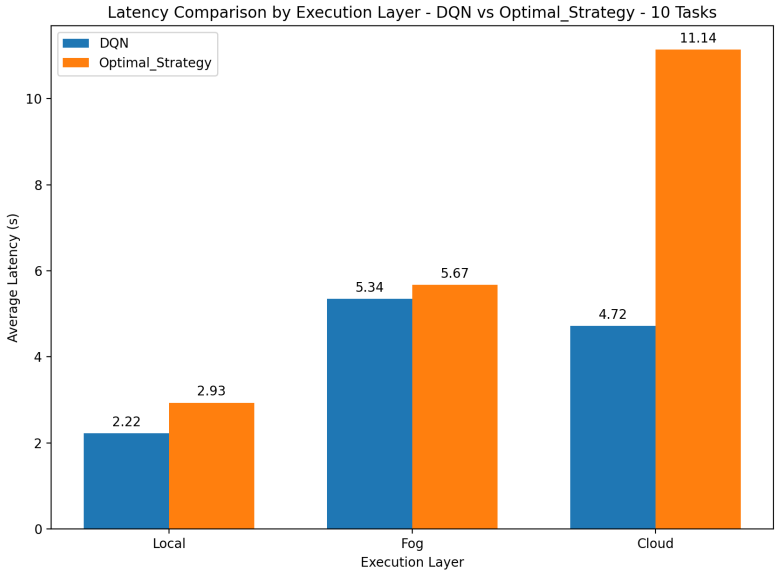


(**b**)

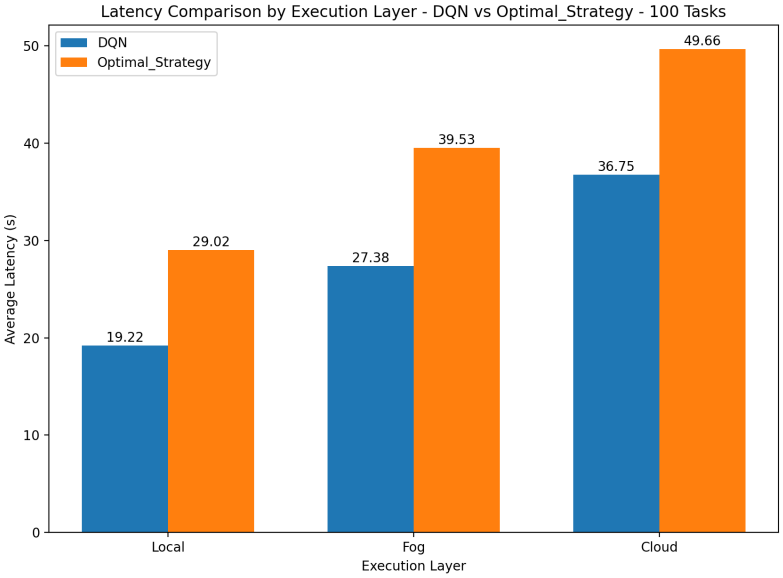
Figure 5. The average energy consumption for DQN and optimal strategy in different execution layers for (**a**) 10 tasks and (**b**) 100 tasks.

5.3.2. Latency

Figure 6 illustrates the comparison of average latency between two strategies, DQN, and the optimal strategy, across three distinct execution layers. A performance comparison is performed 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 6. The average latency for DQN and optimal strategy in different execution layers for (**a**) 10 tasks and (**b**) 100 tasks.

5.4. Compared Methods

In this study, the proposed optimal strategy and the DQN algorithm are compared against two reference algorithms: DJA [43] and BAT [1]. Both BAT and DJA utilize parameters and constraints similar to those of the proposed approach but rely on different models within the IoT–fog–cloud computing framework. In contrast, the DDPG-based algorithm [30] operates under distinct environmental conditions. Specifically, [43] introduces a metaheuristic-based task offloading scheme aimed at minimizing delay and maximizing resource utilization within fog computing environments. The BAT algorithm [1] proposes an AI-driven framework for workload offloading and resource allocation in fog–cloud systems that can be enhanced with optical network capabilities. Meanwhile, [30] presents a computation offloading optimization approach based on deep deterministic policy gradient (DDPG) tailored for UAV-assisted MEC systems.

Figure 7 compares the latency performance of these five strategies across varying task volumes ranging from 1 to 100. As expected, the average latency of all methods increases with the number of tasks due to higher contention for system resources, increased communication overhead, and queuing delays. The BAT algorithm demonstrates moderate latency, outperforming DJA but underperforming relative to DQN, DDPG, and the optimal strategy. This is attributable to BAT's search-based methodology, which, although effective, may lack responsiveness to dynamic task variations, resulting in higher latency compared to the adaptive DQN approach. Conversely, DJA exhibits the highest latency among all algorithms considered, as its deterministic heuristic approach lacks the flexibility and adaptability characteristic of AI-driven methods. This limitation leads to suboptimal latency performance, especially in environments with rapidly changing workloads and heterogeneous IoT requests.

The DDPG-based method excels in handling continuous state and action spaces, enabling it to learn fine-grained offloading policies that adapt effectively to dynamic environmental conditions, which explains its superior latency performance among the benchmark algorithms.

Nonetheless, the DQN algorithm consistently achieves the lowest latency, establishing it as the most efficient strategy evaluated. Its robust performance is sustained even as the number of tasks increases, demonstrating both scalability and adaptability. Quantitatively, the DQN approach improves latency by over 30%, compared to the optimal strategy and BAT, and achieves nearly a 40% improvement relative to DJA. Although the optimal strategy starts with slightly higher latency than DQN, it still outperforms BAT and DJA throughout the evaluated range.

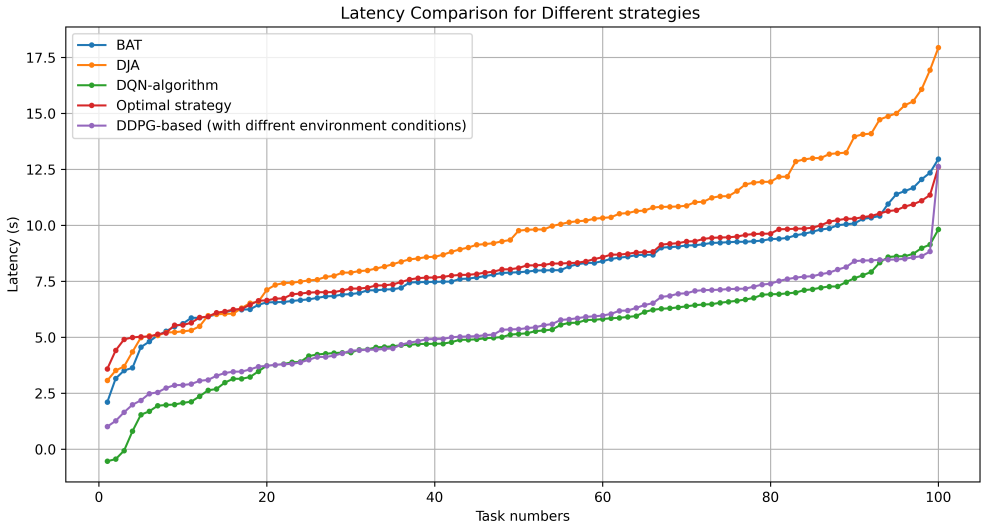


Figure 7. Latency Comparison across task offloading strategies.

Figure 8 illustrates the energy consumption associated with processing tasks across the five evaluated techniques, plotted against varying task quantities. The figure demonstrates a significant increase in energy usage as the number of tasks rises. Among the compared strategies, the DQN approach achieves the best overall energy efficiency. Notably, the optimal strategy initially consumes energy levels comparable to or lower than the DQN algorithm for the first 30 tasks, as it relies on exhaustive search or perfect knowledge of the system. However, this approach becomes computationally infeasible as the task volume grows.

In contrast, the BAT algorithm effectively balances computational resource utilization and energy consumption, resulting in more efficient energy usage than the DJA method,

which lacks global optimization capabilities. The DDPG-based approach exhibits relatively high energy consumption, comparable to that of the DJA strategy.

As depicted, the BAT and DJA methods start with higher energy consumption, approximately 40 joules, compared to the DQN approach, while the optimal strategy begins near 20 joules. Quantitatively, the DQN method demonstrates improvements of approximately 35% and 50% relative to the BAT and DJA algorithms, respectively, underscoring the superiority of the proposed DQN-based strategy over traditional optimization and heuristic methods.

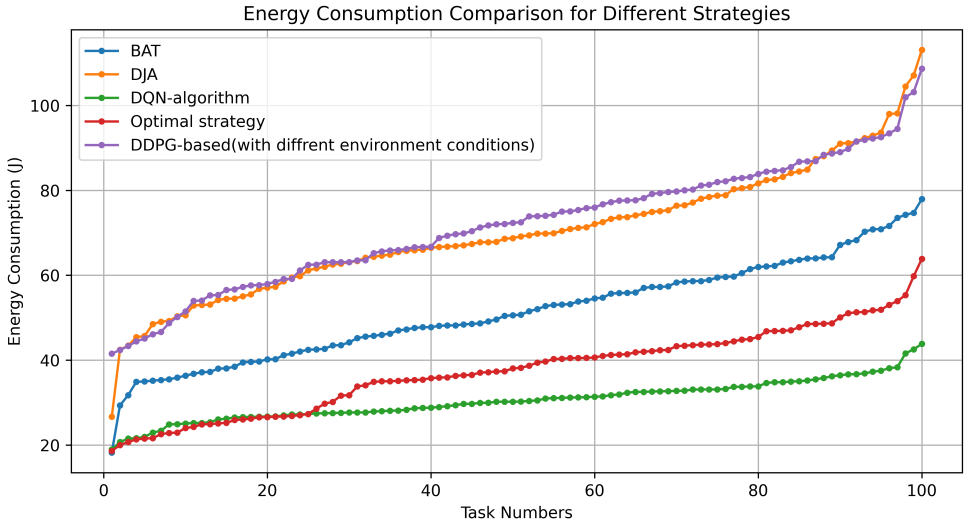


Figure 8. Energy consumption comparison across tasks offloading strategies.

As illustrated in Figure 9, the cost metrics of the various algorithms fluctuate with the increasing number of tasks. The DQN algorithm consistently outperforms all other approaches in terms of overall cost. Beyond a certain task threshold, the total cost associated with the optimal strategy decreases by approximately 30%, compared to the DJA and BAT algorithms, while the DDPG-based approach demonstrates moderate performance across varying environmental conditions. Notably, the DQN strategy achieves costs that are roughly 50% lower than those of both the BAT and DJA methods. This superior performance is attributed to the DQN's effective decision-making capabilities, which adeptly leverage task characteristics and execution locations to address complex scenarios.

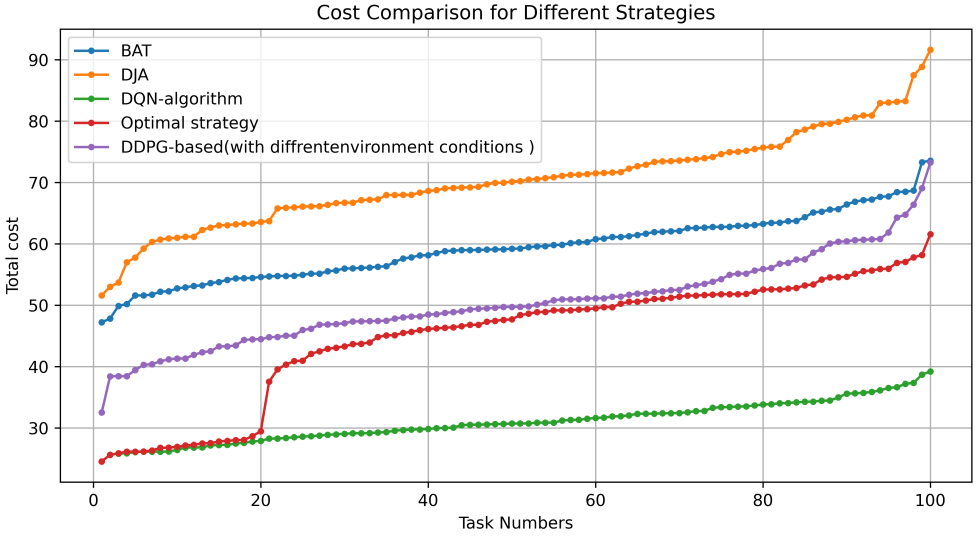


Figure 9. Total cost comparison across tasks offloading strategies.

In contrast, the DJA algorithm incurs higher costs relative to the other strategies due to its reliance on static, rule-based decision-making, resulting in suboptimal resource utilization and limited scalability. Although dynamic in nature, DJA lacks the adaptability to effectively respond to changes in system state, resource heterogeneity, and multi-objective constraints, differentiating it from the more flexible approaches examined.

The comparative analysis of five task offloading strategies—BAT, DJA, DQN-based algorithm, optimal approach, and DDPG-based method—reveals significant performance differences across three critical metrics: cost, energy consumption, and latency. The mean and standard deviation of the three metrics accross 100 tasks are given in Table 4. The DQN-based algorithm consistently outperforms all other methods, demonstrating a balanced and effective approach. Additionally, it exhibits low variability across all metrics, as evidenced by a cost standard deviation of less than 0.012, which translates into a narrow 95% confidence interval, thereby statistically confirming the robustness and reliability of its performance.

Table 4. Mean and standard deviation of the three metrics across 100 tasks.

Strategy	Metrics	Mean	Standard Deviation
BAT	Cost	58.7	0.024
	Energy consumption	55.0	0.18
	Latency	8.7	0.07
DJA	Cost	65.2	0.031
	Energy consumption	70.3	0.25
	Latency	10.2	0.09
DQN-based algorithm	Cost	28.0	0.012
	Energy consumption	30.2	0.01
	Latency	5.1	0.04
Optimal strategy	Cost	30.5	0.015
	Energy consumption	35.0	0.13
	Latency	6.0	0.05
DDPG-based (with different environment constraints)	Cost	45.3	0.029
	Energy consumption	50.5	0.2
	Latency	4.5	0.06

Among the evaluated algorithms, the DQN approach emerges as the clear leader in energy efficiency. Although the optimal strategy initially performs well, its competitiveness diminishes as the number of tasks increases. This decline is attributable to the fact that the optimal strategy serves as a theoretical benchmark derived under idealized assumptions with complete and static knowledge of the environment—conditions rarely met in real-world IoT–fog–cloud networks, especially when incorporating optical networks and supporting Optical IoT applications. Recent studies, such as [31], have further demonstrated that offloading performance can be enhanced under realistic and dynamic conditions by integrating advanced DRL techniques with adaptive decision-making processes. This underscores the value of DRL-based methods like the proposed DQN algorithm, which can continuously learn and adapt its decisions online to improve responsiveness and energy efficiency in IoT, O-IoT, and optical network environments.

The BAT algorithm occupies an intermediate position in terms of energy and latency performance, whereas DJA ranks as the least efficient method. The DQN-based method's ability to adapt to real-time fluctuations and practical constraints—through continuous learning and adjustment based on environmental feedback—enables it to achieve performance comparable to or exceeding that of the theoretical optimal approach. This

adaptability makes the DQN particularly suitable for dynamic scenarios characterized by unpredictable workloads and network variability.

6. Conclusions

This paper has presented a comprehensive and well-structured study on task offloading in hybrid IoT–fog–cloud computing networks, including optical networks and O-IoT, with a particular emphasis on optimizing latency and energy consumption. The proposed approach leverages the adaptive learning capabilities of deep Q-networks (DQN) to dynamically optimize task offloading decisions in real time, resulting in substantial improvements in overall system efficiency. Compared to the BAT and DJA algorithms, which incorporate heuristic and metaheuristic techniques under similar constraints, the DQN strategy achieves enhancements of approximately 35% and 50% in energy consumption and 30% and 40% in latency, respectively. Notably, DQN is especially well-suited for problems characterized by discrete and finite action spaces, in contrast to methods such as DDPG or PPO, which target continuous or hybrid action domains. Within comparable resource allocation and offloading frameworks, DQN offers a favorable balance between computational complexity and decision-making accuracy.

The results demonstrate that adopting the DQN learning strategy enables effective latency reduction and concomitant energy savings, which are critical for prolonging the operational lifespan and responsiveness of IoT systems. Consequently, this work contributes to the expanding body of knowledge on intelligent resource management in IoT, wireless networks, O-IoT, and cloud computing environments, laying a solid foundation for future research and practical applications employing machine learning techniques.

Future work should explore the deployment of the DQN model in real-world, large-scale IoT and O-IoT scenarios with increased complexity. Additionally, investigating complementary machine learning methods and evolving algorithms may further enhance or augment the proposed approach. Moreover, the use of dynamic multi-objective optimization techniques could be investigated to simultaneously optimize latency, energy consumption, and other relevant metrics. This approach would provide a more flexible and adaptive framework for addressing the complex trade-offs inherent in real-world IoT–fog–cloud task offloading scenarios [54,55].

Author Contributions: Methodology, A.B., W.K., T.D.H., Z.P. and S.S.; Software, R.B.; Validation, A.B. and R.B.; Formal analysis, A.B., R.B., Z.P. and S.S.; Investigation, A.B.; Resources, A.B.; Data curation, A.B. and R.B.; Writing—original draft, A.B.; Writing—review & editing, A.B., W.K., T.D.H. and S.S.; Visualization, A.B.; Supervision, T.D.H. and Z.P.; Funding acquisition, T.D.H. All authors have read and agreed to the published version of the manuscript.

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. *J. Grid Comput.* **2023**, *21*, 1–17. [CrossRef]
2. Selim Demir, M.; Hossien, B.; Murat Uysal, E. Relay-Assisted Handover Technique for Vehicular VLC Networks. *ITU J. Future Evol. Technol.* **2022**, *3*, 11–19. [CrossRef]

3. 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. [CrossRef]
4. Goudarzi, M.; Wu, H.; Palaniswami, M.; Buyya, R. An Application Placement Technique for Concurrent IoT Applications in Edge and Fog Computing Environments. *IEEE Trans. Mob. Comput.* **2021**, *20*, 1298–1311. [CrossRef]
5. Chow, C.-W. Recent Advances and Future Perspectives in Optical Wireless Communication, Free Space Optical Communication and Sensing for 6G. *J. Light. Technol.* **2024**, *42*, 3972–3980. [CrossRef]
6. 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. [CrossRef]
7. Abdullah, S.; Jabir, A. A Lightweight Multi-Objective Task Offloading Optimization for Vehicular Fog Computing. *Iraqi J. Electr. Electron. Eng.* **2021**, *17*, 1–10. [CrossRef]
8. 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*, 16067–16081. [CrossRef]
9. Alharbi, H.A.; Aldossary, M.; Almutairi, J.; Elgendy, I.A. Energy-Aware and Secure Task Offloading for Multi-Tier Edge-Cloud Computing Systems. *Sensors* **2023**, *23*, 3254. [CrossRef]
10. Kumar, M.; Sharma, S.C.; Goel, A.; Singh, S.P. A Comprehensive Survey for Scheduling Techniques in Cloud Computing. *J. Netw. Comput. Appl.* **2019**, *143*, 1–33. [CrossRef]
11. 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. *Opt. Fiber Technol.* **2023**, *81*, 103543. [CrossRef]
12. Jiang, Y.-L.; Chen, Y.-S.; Yang, S.-W.; Wu, C.-H. Energy-Efficient Task Offloading for Time-Sensitive Applications in Fog Computing. *IEEE Syst. J.* **2019**, *13*, 2930–2941. [CrossRef]
13. Iftikhar, S.; Gill, S.S.; Song, C.; Xu, M.; Aslanpour, M.S.; Toosi, A.N.; Du, J.; Wu, H.; Ghosh, S.; Chowdhury, D.; et al. AI-Based Fog and Edge Computing: A Systematic Review, Taxonomy and Future Directions. *Internet Things* **2023**, *21*, 100674. [CrossRef]
14. Lu, H.; He, X.; Zhang, D. Security-Aware Task Offloading Using Deep Reinforcement Learning in Mobile Edge Computing Systems. *Electronics* **2024**, *13*, 2933. [CrossRef]
15. 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 Trans. Netw. Serv. Manag.* **2022**, *19*, 4872–4887. [CrossRef]
16. Fang, J.; Qu, D.; Chen, H.; Liu, Y. Dependency-Aware Dynamic Task Offloading Based on Deep Reinforcement Learning in Mobile Edge Computing. *IEEE Trans. Netw. Serv. Manag.* **2023**, *2*, 1403–1415. [CrossRef]
17. 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 Things J.* **2021**, *9*, 11737–11748. [CrossRef]
18. Eldeeb, H.B.; Naser, S.; Bariah, L.; Muhaidat, S.; Uysal, M. Digital Twin-Assisted OWC: Towards Smart and Autonomous 6G Networks. *IEEE Netw.* **2024**, *38*, 153–162. [CrossRef]
19. Zhou, R.; Zhang, X.; Qin, S.; Lui, J.C.S.; Zhou, Z.; Huang, H.; Li, Z. Online Task Offloading for 5G Small Cell Networks. *IEEE Trans. Mob. Comput.* **2022**, *21*, 2103–2115. [CrossRef]
20. Eldeeb, H.B.; Selmy, H.A.I.; Elsayed, H.M.; Badr, R.I.; Uysal, M. Efficient Resource Allocation Scheme for Multi-User Hybrid VLC/IR Networks. In Proceedings of the 2022 IEEE Photonics Conference (IPC), Vancouver, BC, Canada, 13–17 November 2019; pp. 1–2. [CrossRef]
21. Norsyafizan, W.; Mohd, S.; Dimyati, K.; Awais Javed, M.; Idris, A.; Mohd Ali, D.; Abdullah, E. Energy-Efficient Task Offloading in Fog Computing for 5G Cellular Network. *Eng. Sci. Technol. Int. J.* **2024**, *50*, 101628. [CrossRef]
22. 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**, *25*, 2049–2107. [CrossRef]
23. Hosny, K.M.; Awad, A.I.; Khashaba, M.M.; Fouda, M.M.; Guizani, M.; Mohamed, E.R. Optimized Multi-User Dependent Tasks Offloading in Edge-Cloud Computing Using Refined Whale Optimization Algorithm. *IEEE Trans. Sustain. Comput.* **2024**, *9*, 14–30. [CrossRef]
24. 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 Gener. Comput. Syst.* **2022**, *128*, 333–348. [CrossRef]
25. 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. *Appl. Soft Comput.* **2021**, *112*, 107790. [CrossRef]
26. 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. *Mob. Netw. Appl.* **2020**, *25*, 1805–1815. [CrossRef]
27. Yan, P.; Choudhury, S. Deep Q-Learning Enabled Joint Optimization of Mobile Edge Computing Multi-Level Task Offloading. *Comput. Commun.* **2021**, *180*, 271–283. [CrossRef]
28. 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 Trans. Netw. Serv. Manag.* **2022**, *20*, 369–384. [CrossRef]

29. 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. *Appl. Sci.* **2024**, *14*, 5088. [CrossRef]
30. Hadi, M.U.; Abbasi, A.B. Optimizing UAV Computation Offloading via MEC with Deep Deterministic Policy Gradient. *Trans. Emerg. Telecommun. Technol.* **2023**, *35*, e4874. [CrossRef]
31. Abdulazeez, D.H.; Askar, S.K. A Novel Offloading Mechanism Leveraging Fuzzy Logic and Deep Reinforcement Learning to Improve IoT Application Performance in a Three-Layer Architecture within the Fog-Cloud Environment.*IEEE Access* **2024**, *12*, 39936–39952. [CrossRef]
32. Liu, Q.; Tian, Z.; Wang, N.; Lin, Y. DRL-Based Dependent Task Offloading with Delay-Energy Tradeoff in Medical Image Edge Computing. *Complex Intell. Syst.* **2024**, *10*, 3283–3304. [CrossRef]
33. Zhang, S.; Tong, X.; Chi, K.; Gao, W.; Chen, X.; Shi, Z. Stackelberg Game-Based Multi-Agent Algorithm for Resource Allocation and Task Offloading in MEC-Enabled C-ITS. *IEEE Trans. Intell. Transp. Syst.* **2025**, 1–12. [CrossRef]
34. 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 Pract. Exp.* **2017**, *47*, 1275–1296. [CrossRef]
35. Wu, M.; Song, Q.; Guo, L.; Lee, I. Energy-Efficient Secure Computation Offloading in Wireless Powered Mobile Edge Computing Systems. *IEEE Trans. Veh. Technol.* **2023**, *72*, 6907–6912. [CrossRef]
36. Li, H.; Zhang, X.; Li, H.; Duan, X.; Xu, C. SLA-Based Task Offloading for Energy Consumption Constrained Workflows in Fog Computing. *Future Gener. Comput. Syst.* **2024**, *156*, 64–76. [CrossRef]
37. 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 Trans. Cogn. Commun. Netw.* **2021**, *7*, 881–892. [CrossRef]
38. Sabireen, H.; Venkataraman, N. A Hybrid and Light-Weight Metaheuristic Approach with Clustering for Multi-Objective Resource Scheduling and Application Placement in Fog Environment. *Expert Syst. Appl.* **2023**, *223*, 119895. [CrossRef]
39. Ren, J.; Yu, G.; He, Y.; Li, G.Y. Collaborative Cloud and Edge Computing for Latency Minimization. *IEEE Trans. Veh. Technol.* **2019**, *68*, 5031–5044. [CrossRef]
40. 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 Trans. Sens. Netw.* **2024**, **20**, 1–22. [CrossRef]
41. Tang, M.; Wong, V.W.S. Deep Reinforcement Learning for Task Offloading in Mobile Edge Computing Systems. *IEEE Trans. Mob. Comput.* **2020**, *21*, 1985–1997. [CrossRef]
42. Birhanie, H.M.; Adem, M.O. Optimized Task Offloading Strategy in IoT Edge Computing Network. *J. King Saud Univ. Comput. Inf. Sci.* **2024**, *36*, 101942. [CrossRef]
43. Kumari, N.; Jana, P.K. A Metaheuristic-Based Task Offloading Scheme with a Trade-off between Delay and Resource Utilization in IoT Platform. *Clust. Comput.* **2023**, *27*, 4589–4603. [CrossRef]
44. Ahmadi, K.; Serdijn, W.A. Advancements in Laser and LED-Based Optical Wireless Power Transfer for IoT Applications: A Comprehensive Review. *IEEE Internet Things J.* **2025**, *12*, 18887–18907. [CrossRef]
45. Robles-Enciso, A.; Skarmeta, A.F. A Multi-Layer Guided Reinforcement Learning-Based Tasks Offloading in Edge Computing. *Comput. Netw.* **2023**, *220*, 109476. [CrossRef]
46. 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*, 4121. [CrossRef]
47. Penmetcha, M.; Min, B.-C. A Deep Reinforcement Learning-Based Dynamic Computational Offloading Method for Cloud Robotics. *IEEE Access* **2021**, *9*, 60265–60279. [CrossRef]
48. Krishnamoorthy, A.; Safi, H.; Younus, O.; Kazemi, H.; Osahon, I.N.O.; Liu, M.; Liu, Y.; Babadi, S.; Ahmad, R.; Asim, I.; et al. Optical Wireless Communications: Enabling the next Generation Network of Networks. *IEEE Veh. Technol. Mag.* **2025**, *20*, 20–39. [CrossRef]
49. Li, J.; Yang, Z.; Chen, K.; Ming, Z.; Cheng, L. Dependency-Aware Task Offloading Based on Deep Reinforcement Learning in Mobile Edge Computing Networks. *Wirel. Netw.* **2023**, *30*, 1–13. [CrossRef]
50. 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*, 30. [CrossRef]
51. Zendebudi, A.; Choudhury, S. Designing a Deep Q-Learning Model with Edge-Level Training for Multi-Level Task Offloading in Edge Computing Networks. *Appl. Sci.* **2022**, *12*, 10664. [CrossRef]
52. Watkins, C.J.C.H.; Dayan, P. Q-Learning. *Mach. Learn.* **1992**, *8*, 279–292. [CrossRef]
53. Chicone, C. Stability Theory of Ordinary Differential Equations. In *Encyclopedia of Complexity and Systems Science*; Springer: New York, NY, USA, **2009**; pp. 8630–8649. [CrossRef]

54. Sahmoud, S.; Topcuoglu, H.R. Dynamic multi-objective evolutionary algorithms in noisy environments. *Inf. Sci.* **2023**, *634*, 650–664. [CrossRef]
55. Tema, E.Y.; Sahmoud, S.; Kiraz, B. Radar placement optimization based on adaptive multi-objective meta-heuristics. *Expert Syst. Appl.* **2024**, *239*, 122568. [CrossRef]