future internet

MDPI

Article

Proximal Policy Optimization for Efficient D2D-Assisted Computation Offloading and Resource Allocation in Multi-Access Edge Computing

Chen Zhang [1], Celimuge Wu [2], Min Lin [1], Yangfei Lin [2] and William Liu [3,*]

[1] College of Computer Science and Technology, Inner Mongolia Normal University, Saihan District, Hohhot 010096, China; 20214021013@mails.imnu.edu.cn (C.Z.)
[2] Graduate School of Informatics and Engineering, The University of Electro-Communications, Tokyo 1828585, Japan; celimuge@uec.ac.jp (C.W.); linyangfei@uec.ac.jp (Y.L.)
[3] School of Computing, Electrical and Applied Technologies, Unitec Institute of Technology, Auckland 1025, New Zealand
* Correspondence: wliu@unitec.ac.nz

Abstract: In the advanced 5G and beyond networks, multi-access edge computing (MEC) is increasingly recognized as a promising technology, offering the dual advantages of reducing energy utilization in cloud data centers while catering to the demands for reliability and real-time responsiveness in end devices. However, the inherent complexity and variability of MEC networks pose significant challenges in computational offloading decisions. To tackle this problem, we propose a proximal policy optimization (PPO)-based Device-to-Device (D2D)-assisted computation offloading and resource allocation scheme. We construct a realistic MEC network environment and develop a Markov decision process (MDP) model that minimizes time loss and energy consumption. The integration of a D2D communication-based offloading framework allows for collaborative task offloading between end devices and MEC servers, enhancing both resource utilization and computational efficiency. The MDP model is solved using the PPO algorithm in deep reinforcement learning to derive an optimal policy for offloading and resource allocation. Extensive comparative analysis with three benchmarked approaches has confirmed our scheme's superior performance in latency, energy consumption, and algorithmic convergence, demonstrating its potential to improve MEC network operations in the context of emerging 5G and beyond technologies.

Keywords: multi-access edge computing (MEC); 5G networks; Device-to-Device (D2D); proximal policy optimization (PPO); Markov decision process (MDP); computation offloading; collaborative offloading; resource allocation

Citation: Zhang, C.; Wu, C.; Lin, M.; Lin, Y.; Liu, W. Proximal Policy Optimization for Efficient D2D-Assisted Computation Offloading and Resource Allocation in Multi-Access Edge Computing. *Future Internet* **2024**, *16*, 19. https://doi.org/10.3390/fi16010019

Academic Editor: Paolo Bellavista

Received: 11 December 2023
Revised: 29 December 2023
Accepted: 29 December 2023
Published: 2 January 2024

1. Introduction

With the development of fifth-generation mobile networks (5G) and more advanced mobile communication networks, the number of mobile device users is expected to increase from 5.17 billion to 7.33 billion by the end of 2023, according to Statista [1]. In this rapidly evolving digital landscape, MEC servers, as a pivotal technology for enhancing the computational capabilities of mobile devices, offer users improved quality of service (QoS) and quality of experience (QoE). By decentralizing cloud computing, the MEC adeptly manages burgeoning data traffic, mitigates network congestion, and curtails latency, evolving to incorporate diverse technologies, including WiFi [2] and fixed access [3] (ETSI ISG [4]).

Amidst such advancements, MEC networks face the intricate challenge of efficient task management and coordination due to their compositional complexity and inherent variability. This complexity is further augmented in distributed layouts, which, unlike their centralized counterparts that optimize globally, offer escalated flexibility and scalability, essential for vast networks operating without central oversight.

Device-to-Device (D2D) communication has surfaced as a promising paradigm within MEC to bolster collaborative offloading efforts [5]. Studies [6,7] have exemplified enhancements in network capacity, energy efficiency, and latency reduction through D2D-facilitated offloading. In resource-constrained environments, Zhou et al. employed the game theory to devise a D2D-assisted offloading algorithm optimizing task computation time [8], while Xiao et al. tackled the content caching through an improved multi-objective bat algorithm [9]. Abbas et al. considered cooperative-aware task offloading to balance the number of completed tasks against energy and cost [10], and Li et al. proposed a security-conscious, energy-aware task offloading framework [11].

While the existing research has laid a robust groundwork for harnessing idle computational resources in MEC, optimizing the utilization of these resources remains a significant challenge. The integration of D2D technology introduces a spectrum of offloading modalities and platforms, enriching user services but also complicating offloading decisions with additional considerations like local computational capacities and the availability of proximate nodes.

Addressing the dynamic, transient nature of D2D-integrated MEC networks, this paper proposes a PPO-based D2D-assisted computation offloading and resource allocation scheme. As shown in Figure 1, we consider a single unit in the MEC network, including two access points (APs) and multiple requesters and collaborators deployed at the edge of the network, where the user equipment (UE) can choose the appropriate computation method, such as local computation, edge computation, D2D computation, and migration computation, according to its own needs within the coverage of the APs. Our contribution is as follows:

- We construct an authentic MEC network environment mirroring the actual MEC architecture, utilizing ad hoc wireless technology and computational devices, moving beyond mere simulation-based studies.
- Considering the MEC network's complexity, dynamics, and randomness, we refine the neural network training features on the UE to include CPU utilization, transmission delay, task execution time, task count, and the aggregate of transmission and computation energy consumption.
- We introduce a PPO-based D2D-assisted computational offloading and resource allocation scheme to amplify terminal device resource utilization. By formulating a Markov decision process (MDP) model, we seek to minimize the time loss and energy consumption, deriving an optimal offloading strategy via PPO application.
- The experimental results show that our scheme outperforms the other three baselines in terms of delay, energy consumption, and algorithm convergence.

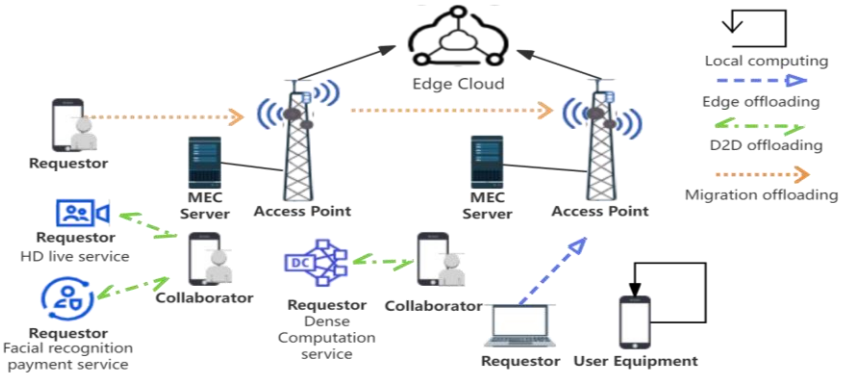


Figure 1. Multi-access edge computing scenarios.

The remaining sections are organized as follows. In Section 2, we review the related work. Section 3 defines the system model, including the communication and computation models, and formally defines the D2D auxiliary computation offloading and resource allocation problem. Section 4 describes the MDP model for this problem and the specific

application of the PPO algorithm to this experiment. Section 5 describes the experimental environment and parameter settings and compares this scheme with three other baselines. Section 6 concludes the paper.

2. Related Work

In the multi-access edge computing (MEC) domain, computational offloading emerges as a multifaceted challenge, demanding innovative solutions [12–14], which are crucial for enhancing network efficiency. Research in this domain has predominantly concentrated on three approaches: mathematical optimization algorithms, control algorithms, and artificial intelligence (AI)-driven optimization algorithms. Among these, AI and machine learning (ML)-based algorithms have shown particular promise in dynamic and complex MEC environments due to their adaptability and learning capabilities [15].

The studies to date have focused on optimizing latency and energy consumption during computational offloading. Abbas et al. utilized meta-heuristic algorithms, such as ant colony, whale, and grey wolf optimization, to establish an offloading strategy aimed at minimizing energy and delay, albeit within the confines of a singular MEC server context [16]. While meta-heuristic algorithms are robust, their complexity and numerous parameters often complicate the tuning process. In the realm of control algorithms, Lin et al. proposed drone-assisted dynamic resource allocation leveraging Lyapunov optimization to ensure delay efficiency and reliability of offloading services [17]. Chang et al. contributed a balanced framework using SDN-based controller load balancing with Lyapunov optimization for improved offloading efficiency and reduced latency [18].

The precision of mathematical optimization for linear problems contrasts with its constrained adaptability to dynamic changes, whereas control algorithms excel in system stability and robustness but can be complex and less suitable for non-linear systems. Deep reinforcement learning (DRL), marrying deep learning with reinforcement learning, has emerged as a potent approach in MEC for managing dynamic tasks, with techniques like deep Q-network (DQN) and advantage actor–critic (A2C) algorithms gaining traction [19–21].

With the endeavor to overcome the limitations of mathematical optimization algorithms and control class algorithms, many researchers have used DRL methods for their self-learning and self-adaptation qualities in MEC resource allocation. In [19], the authors demonstrated the superiorities and applications of ML and deep learning (DL) methods in MEC environments for two key tasks: automatic identification and offloading of "untrusted tasks" and task scheduling on MEC servers, in particular using a flow-shop-like scheduling approach. Gao et al. developed an enhanced scheme based on the deep deterministic policy gradient (DDPG), which exhibited notable performance in reducing energy consumption, managing load status, and minimizing latency. However, this scheme did not adequately consider the involved behavioral characteristics of the devices [20]. To minimize the computing delay and energy consumption of UE, Liang et al. used DQN and DDPG to process large-scale state spaces and obtained the task offloading ratio and power allocation suitable for each UE. Silva et al. utilized A2C, empowered UEs to make intelligent offloading decisions, and demonstrated efficacy in simulated OpenAI Gym scenarios but did not fully capture the dynamism of mobile devices [21].

Although DRL demonstrates its superiority in resource management, the challenge of high user concurrency persists, often impacting QoS and QoE due to resource contention, and the challenge of high user concurrency persists, often impacting QoS and QoE due to resource contention [22]. D2D communication's potential in offloading via high-speed, direct connections between devices has thus garnered academic focus. Li et al. addressed the energy consumption optimization problem for UE-assisted MEC computational offloading in mobile environments and proposed a DRL-based computational offloading model for D2D-MEC. Finally, they demonstrated that this scheme consumes the least amount of energy and cost in continuous time [23]. Lin et al. investigated a D2D collaborative computing offloading design for two users dynamically exchanging computational loads over a D2D link. The design achieved minimax optimization for a given finite time horizon [24]. Guan

et al. studied the multi-user collaborative partial offloading and computational resource allocation problem, using a DQN algorithm to maximize the number of collaborative devices to maximize the computational resources under the maximum latency constraint of the application and limited computational resources [25]. Liu et al. investigated a computational offloading scheme in a multi-user UAV system, where a UAV with a computational task can offload part of the task to a nearby assistant to satisfy a latency constraint [26]. Fan et al. used the Lagrange multiplier method to solve the problem of computing resource allocation and adopted a greedy strategy to select D2D users, substantially reducing the average execution delay of the task [27]. They all increase the computing power by increasing the number of devices rather than fully utilizing the per-device computing resources.

D2D communication, a key component in enhancing MEC, has also received significant attention. Its ability to facilitate high-speed, short-range connections makes it a valuable asset in offloading tasks. However, research in this area has often focused on increasing the computational power through additional devices rather than optimizing per-device resource utilization. Our study contributes to this field by proposing a PPO-based D2D-assisted computation offloading and resource allocation scheme. This scheme not only addresses the dynamic nature of MEC networks but also optimizes resource utilization at a per-device level, filling a critical gap in the current research landscape.

3. System Model

In this study, we propose a D2D-assisted MEC server computation offloading and resource allocation scheme for multi-access edge computing scenarios. This model aims to minimize the total offloading cost, including task execution time and energy consumption. The proposed system model, as shown in Figure 2, is crucial to understanding how UE can optimize offloading decisions based on the device state, proximity to computational resources, and the nature of computational tasks. The system model consists of three parts: the network model, the communication model, and the computational model, which will be introduced in detail.

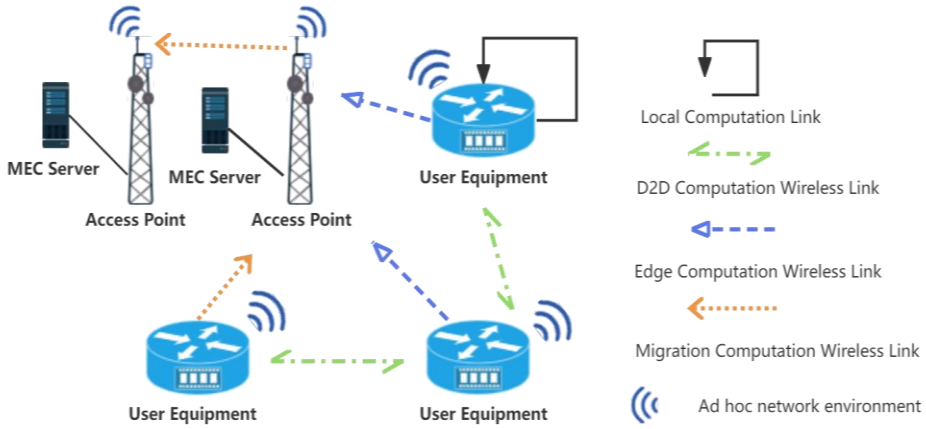


Figure 2. MEC offloading model.

3.1. Network Model

The network model consists of multiple UEs and distributed MEC servers. In an ad hoc network [28,29] environment, each UE can generate computationally intensive tasks, while the MEC servers and available computing nodes nearby provide the necessary computational resources. M represents the maximum workload of the MEC server and C represents the maximum computational resources provided by the MEC server. Among the network entities, let $U = \{1, 2, 3, \ldots, i, \ldots, N\}$ as the set of all UEs in the network. All UEs can be represented as $UE_i = \{m_i, f_i, t_i, r_i, s_i\}$, where m_i denotes the size of the user task. f_i is the operating frequency of the CPU for processing computational tasks and t_i represents the maximum tolerable time for the current UE transmission or computation. The variable r_i represents the current computing state. When UE_i executes a computation task, $r_i = 1$;

otherwise, $r_i = 0$. The variables $s_i \in \{0, 1, 2, 3\}$ denote the system offloading decision set, representing different offloading strategies. The variables $S = \{s_i, i \in U\}$ represent the system offloading strategies, which represent different offloading decisions: $s_i = 0$ for local computation, $s_i = 1$ for offloading the task via D2D, $s_i = 2$ for offloading the task via the current MEC, and $s_i = 3$ for migrating the task to another MEC server, which is called migration calculation in this article.

3.2. Communication Model

The communication model defines the channels through which UEs and MEC servers interact. When an UE requires the edge server computation, the computation task for offloading must be transferred from the UE to the AP, and then the AP is assigned to the MEC server for computation. This process requires data transmission over the network, so the network bandwidth is one of the key factors affecting the overall system performance. To optimize the offloading process, we conceptualize a communication model that imposes a bandwidth limit between the UE and the AP. This measure is designed to ensure equitable bandwidth distribution among UEs, thereby mitigating potential network congestion. It simultaneously governs energy expenditure while striving to maintain an equilibrium between system performance and operational costs. The details are as follows: we take $B = \{1, 2, \ldots, B\}$ as the set of communication links of the AP. W_b is the bandwidth of the AP and we divide the available bandwidth resource of the AP into equal-width subchannels W_{sub}, then the bandwidth of each subchannel can be denoted as $W_{sub,b} = W_b/B$. The matrix M_b denotes the transmission state of the AP, $M_b = [0, 1], b \in B$, and $M_b = 1$ indicates that the AP provides an offloading service on its communication link. $M_b = 0$ indicates that no offloading service is being provided on its communication link. Thus, the bandwidth resource is denoted as:

$$\sum_{b \in B} M_b W_{sub,b}. \tag{1}$$

In the resource-constrained scenario, when the requester offloads the task to the MEC server, the task is first sent from the UE_i to the AP via wireless transmission. We assume that the wireless channel state remains constant while each computational task is transmitted between the UE and the AP. $p_{i,b}$ denotes the transmission power that requester i sends to the AP, $h_{i,b}$ denotes the channel gain during its transmission, σ^2 represents the power of the noise, $d_{i,b}$ denotes the distance between the requester UE_i and the AP, and θ is a standardized path loss propagation exponent so that the path loss can be denoted as $d_{i,b}^{-\theta}$; then, the signal-to-noise ratio (SNR) can be computed as:

$$SNR_{i,b} = \frac{p_{i,b} h_{i,b} d_{i,b}^{-\theta}}{\sigma^2}. \tag{2}$$

After obtaining the SNR, the transmission rate from the requester to the AP can be calculated as

$$R_{i,b} = W_{sub,b} log_2(1 + SNR_{i,b}). \tag{3}$$

In order to avoid the interference of UEs transmitting between D2D communication links, we assume that each D2D communication link can be obtained as an orthogonal subchannel; then, $W_{i,c}$ denotes the bandwidth of the subchannel. Similarly, we denote the signal-to-noise ratio of the D2D communication link, then the transmission rate of D2D communication between requester UE_i, and collaborator c is denoted as follows:

$$R_{i,c} = W_{i,c} log_2(1 + SNR_{i,c}). \tag{4}$$

Since the executor that performs the task of the requester may be a MEC server or another UE (collaborator), based on the above expression, the transmission rate between the requester UE_i and the executor may be expressed as $R_{i,b}$ or $R_{i,c}$.

3.3. Computational Model

We studied four computing models for devices to choose from under the MEC network environment: local computing, D2D peer devices computing, edge computing, and migration computing. The computation delay and energy consumption for each model are as follows.

3.3.1. Local Computing

If the computational requirements of the computing tasks are within the processing power of the UE, the device performs computing tasks locally. We assume that the CPU in UE_i operates at a frequency of f_i, with c_i representing the local computational power in terms of CPU cycles. Therefore, the local computation time can be calculated as follows:

$$t_i^{local} = \frac{c_i}{f_i}. \tag{5}$$

Parameter k is based on the findings presented in reference [30]. Local energy consumption can be mathematically represented as

$$E_i^{local} = k f_i (c_i)^2. \tag{6}$$

Incorporating the influence of chip architecture [31], the effective switching capacitance κ is introduced in the equation. The total energy consumption is provided by the following expression (α, β are loss factors):

$$C_i^{local} = \alpha t_i^{local} + \beta E_i^{local}. \tag{7}$$

3.3.2. Edge Computing

When the computing power of local devices is not enough to efficiently handle tasks or to handle tasks that require a fast response, edge computing can provide lower latency due to its proximity to the data source. The m_i is the size of the task from the requester UE_i and $R_{i,b}$ represents the rate at which UE_i transmits tasks to the MEC server, transmission delay is

$$t_{i,b}^{tran} = \frac{m_i}{R_{i,b}}. \tag{8}$$

The time requirement of edge computing is

$$t_{i,b}^{edge} = \frac{c_i^{edge}}{f_i^{edge}}. \tag{9}$$

$p_{i,b}$ denotes the transmission power that requester UE_i sends to the MEC server. The transmission energy loss is denoted as

$$E_{i,b}^{tran} = p_{i,b} t_{i,b}^{tran}. \tag{10}$$

Computation energy loss is denoted as

$$E_{i,b}^{edge} = p_{i,b} t_{i,b}^{edge}. \tag{11}$$

The total consumption, including the loss of time and energy, provides

$$C_i^{edge} = \alpha \left(t_{i,b}^{tran} + t_{i,b}^{edge} \right) + \beta \left(E_{i,b}^{tran} + E_{i,b}^{edge} \right). \tag{12}$$

3.3.3. Migrating the Computation

When the MEC server experiences an excessive workload or when neighboring servers have available resources, the UE_i can migrate its computational tasks to the adjacent

servers. This migration incurs additional transmission overhead for the UE_i, which involves transferring data between MEC servers. The magnitude of the transmission overhead is determined based on the allocated computational resources provided by the target server in response to the migration request from the UE_i. The i, b' in $t_{i,b'}^{mig}$, $t_{i,b'}^{edge'}$, and $E_{i,b'}^{mig}$ signifies the transmission from UE_i to the AP where another MEC server is located; therefore, the transmission delay is

$$t_{i,b'}^{mig} = \frac{m_i}{R_i^{mig}} + Z_i^{mig}. \tag{13}$$

Assuming that the migration cost only depends on the task size, which is denoted as $Z_i^{mig} = \delta m_i$, the transmission energy loss is

$$E_{i,b'}^{mig} = p_{i,b'} t_{i,b'}^{mig}. \tag{14}$$

The computational delay and energy consumption of the UE_i is computed based on the resource allocation strategy of the migration server, with the computational formula remaining unchanged. Consequently, the total cost of the migration computation is expressed as

$$C_i^{mig} = \alpha \left(t_{i,b'}^{mig} + t_{i,b'}^{edge'} \right) + \beta E_{i,b'}^{mig}. \tag{15}$$

3.3.4. D2D Device Computing

m_i is the size of the task and $R_{i,c}$ represents the rate at which the requester, UE_i, transmits tasks to the collaborator; thus, the transmission delay is

$$t_{i,c}^{tran} = \frac{m_i}{R_{i,c}}. \tag{16}$$

$c_i^{collaborator}$ and $f_i^{collaborator}$ represent the number of CPU cycles required by the collaborator UE_i to complete the computational task m_i of the requester UE_i, and the current CPU working frequency, respectively. The time requirement of D2D computing is

$$t_{i,c}^{d2d} = \frac{c_i^{collaborator}}{f_i^{collaborator}}. \tag{17}$$

$p_{i,c}$ denotes the transmission power that requester UE_i sends to the collaborator. The transmission energy loss is denoted as

$$E_{i,c}^{tran} = p_{i,c} t_{i,c}^{tran}. \tag{18}$$

Computation energy loss is denoted as

$$E_{i,c}^{d2d} = p_{i,c} t_{i,c}^{d2d}. \tag{19}$$

The total consumption provides

$$C_i^{d2d} = \alpha \left(t_{i,c}^{tran} + t_{i,c}^{d2d} \right) + \beta \left(E_{i,c}^{tran} + E_{i,c}^{d2d} \right). \tag{20}$$

3.4. Problem Formulation

To enhance the efficiency of computational task offloading, we constructed a mathematical model tailored to minimize the total execution cost within the MEC framework. This mathematical model integrates the inherent complexities of the network's architecture, the communication resource in use, and the dynamics computation at play in MEC environments. Recognizing the finite nature of MEC server resources and the associated costs of server utilization, our model incorporates economic considerations. Specifically, due to MEC servers having limited resources, assuming that each strategy repeatedly selects an

edge computing strategy will lead to a high cost. Therefore, we assigned very low rewards to successively select strategies for MEC as a main task processor; however, even if such a strategy emerges, it slowly disappears in the iterations.

We used the following equation to express the cost consumption of the four task offloading strategies of the UE:

$$C_i^{ue} = \begin{cases} C_i^{local} s_i \\ C_i^{d2d} s_i \\ C_i^{edge} s_i \\ C_i^{mig} s_i \end{cases}, \forall i \in U_i, s_i \in \{0, 1, 2, 3\}. \tag{21}$$

Since the aim of this study was to minimize the total cost of delay and energy consumption of UEs, the bandwidth resources of AP should be satisfied:

$$\sum_{b \in B} M_b W_{sub,b} \leq W_b, \tag{22}$$

where W_b represents the bandwidth of AP. C_{all} represents all UE devices, which can be calculated as

$$C_{all} = \sum_{n=1}^{N} C_i^{ue}, \forall i \in U_i. \tag{23}$$

Under the constraints of maximum tolerable delay and energy consumption, the problem is expressed as

$$\begin{aligned} &Min \mathrm{C}_{\mathrm{all}} \\ &C1 : s_i \in 0, 1, 2, 3, \forall i \in UE_i; \\ &C2 : t_i^{local} + t_{i,b}^{tran} + t_{i,b}^{edge} + t_{i,c}^{tran} + t_{i,c}^{d2d} + t_{i,b'}^{mig} + t_{i,b'}^{edge'} < t_i, \forall i \in U_i; \\ &C3 : E_i^{local} + E_{i,b}^{tran} + E_{i,b}^{edge} + E_{i,c}^{tran} + E_{i,c}^{d2d} + E_{i,b'}^{mig} + E_{i,b'}^{edge'} < E_i, \forall i \in U_i; \\ &C4 : \textstyle\sum_{n=1}^{N} C_i \leq C_{all}, \forall i \in U_i. \end{aligned} \tag{24}$$

Based on satisfying the QoS and QoE required by users, the solution of this problem can be expressed as solving the minimum total offloading cost C under the optimal offloading policy vector S.

4. Multi-Objective Deep Reinforcement Learning Based on the PPO Algorithm

In this section, we introduce the application of the proximal policy optimization (PPO) algorithm, a robust deep reinforcement learning approach, under the MEC environment. The PPO model was chosen for its notable benefits in multi-objective optimization, characterized by stability, adaptability, and efficient sample utilization, making it highly suitable for the dynamic and uncertain environments of MEC. The problem of computational offloading and resource allocation was first constructed as a Markov decision process (MDP) model, and the PPO algorithm was used to obtain the optimal solution.

4.1. Markov Decision Process

Our approach constructs the computational offloading and resource allocation problem as an MDP. The MDP is defined by a state space encompassing key metrics like transfer latency, computational delay, energy consumption, an action space for offloading strategies, and a reward function designed to minimize latency and energy consumption. This formulation is crucial for generating optimized offloading policies based on the current state of the network. In the context of computational offloading and resource allocation within the edge environments, we define the MDP as a quintuple, encompassing the state space (S), action space (A), state transition probability matrix (P), reward function (R), and the discount factor (γ). The crux of the MDP's application in this scenario lies in its ability to generate a policy that dictates offloading decisions based on the current state.

- State: The state of the environment consists of several metrics for the local device, the MEC server, and the collaborator devices. Transfer latency used by the computational tasks, computational latency, transfer energy consumption, computational energy consumption, CPU utilization of the current device, and the number of computational tasks.
- Action: The action space consists of four offloading strategies: local computing, D2D computing, edge computing, and migration computing.
- Probability: The state probability transfer matrix can be expressed as

$$P_{ss'}^{a} = P\left[S_{t+1} = s' \middle| S_t = s, A_t = a\right]. \tag{25}$$

- Reward: The rewards are provided at each episode. Considering the goal of minimizing UE latency and energy consumption, the defined reward should be negatively correlated with this goal. It can be expressed as follows:

$$R(s,a) = \sum_{n=i}^{N} \gamma\left(C_i^{local} - C_i^{d2d} - C_i^{edge} - C_i^{mig}\right), \forall i \in U_i. \tag{26}$$

- γ: is the discount factor, $\gamma \in [0,1]$.

For the computational model of the system, the state can be composed of a number of metrics from the local device, the MEC server, collaborator devices, and the offloaded device. After executing each decision, the agent will receive an immediate reward to measure the goodness of the current decision [32]. The reward function is shown in Equation (26), and the value of the state is dependent on the current reward versus the possible future reward. The return of state is defined as follows:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^3 R_{t+3} + \ldots + \gamma^n R_{t+n} = \sum_{n=0}^{\infty} \gamma^n R_{t+n+1}. \tag{27}$$

γ is used to estimate the current value of future rewards, then the long-term payoff can be expressed as the expected value of G_t, defined as

$$V(s_t, \pi) = E_\pi[G_t] = E_\pi[R_{t+1} + \gamma V(s_{t+1}, \pi)]. \tag{28}$$

$V(s_t, \pi)$ denotes the value function of state s under a given strategy π, S_t denotes the next state, and π denotes the strategy.

The problem is thus formulated as solving the optimal policy π^* of the system model to maximize the long-term value of the model $V(s, \pi)$, expressed as follows:

$$\pi^* = \underbrace{argmax}_{\pi} V((s_{t+1}, \pi)). \tag{29}$$

4.2. PPO Algorithm Application

The application of the PPO algorithm is directed at optimizing the policy function using a neural network. This approach is particularly adapted to navigating the complexities of the non-convex optimization challenges characteristic of intricate MEC scenarios. PPO distinguishes itself by balancing the exploration of innovative strategies with the exploitation of established, efficacious actions facilitated by its distinct objective function and policy update methodology. This allows for effective handling of the non-convex optimization problem prevalent in complex MEC scenarios. The PPO algorithm, with its objective function and policy update mechanism, ensures a balance between the exploration of new strategies and the exploitation of known effective actions [33].

To evaluate the versatility of various reinforcement learning (RL) algorithms and to fine-tune hyperparameters for broader applicability, we decoupled the environmental module from the neural network. This separation enables the environmental module to

focus exclusively on generating environmental feedback and computing rewards, a critical step in enhancing the scheme's generality, as illustrated in Figure 3. The environmental module's interfaces, namely reset and step, are integral to the RL process. The reset interface initializes or refreshes the environment to a baseline state, providing the initial observation post-training initiation or episode completion. In contrast, the step interface progresses the environment in response to the agent's actions, delivering subsequent observations, instant rewards, termination flags, and additional context-specific data. This delineation of responsibilities ensures a coherent and systematic interaction between the PPO algorithms and the environment module, promoting an efficient learning cycle predicated on action-derived feedback.

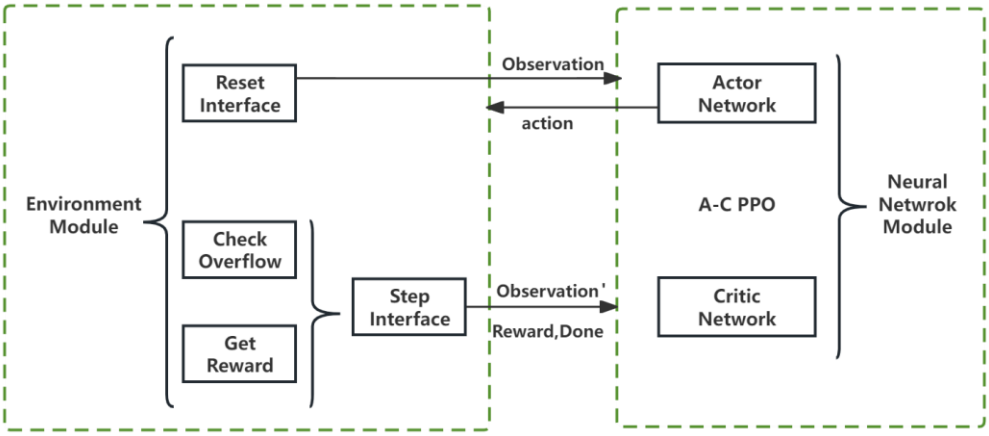


Figure 3. The interaction process between two modules.

Within the neural network module, the PPO algorithm updates the neural network parameters based on state returns by sampling historical offloading data. This process involves two distinct neural networks: the actor network, which is responsible for updating the current policy by determining action probabilities, and the critic network, which evaluates the policy's expected returns. The functions of the actor and critic neural networks are to update and evaluate the current strategy, respectively. Upon receiving the current environment state as input, the actor network employs the neural network's parameters, θ, to ascertain the likelihood of undertaking a particular action within state s. Concurrently, the critic network computes the value of the current policy, effectively estimating the expected returns. This dual-structured approach enables the PPO algorithm to harmoniously balance the need for exploration, which involves experimenting with novel actions with the necessity of exploitation or optimizing known effective actions. Through this iterative interaction, the PPO algorithm learns and hones strategies, progressively navigating towards an optimal policy.

The objective function of the policy, denoted as $J(\theta)$ (Equation (30)), integrates with $\hat{A}_t$, the estimator of the advantage function with time step t, and π_θ is the strategy function.

$$J(\theta) = E_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t\right], \tag{30}$$

$$J(\theta) = \hat{E}_t\left[\min\left(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]. \tag{31}$$

The PPO algorithm utilizes the principle of importance sampling to update the strategy with the experience generated by the old strategy. It solves the redundant process in reinforcement learning that requires recollecting experience after each policy update, thus effectively utilizing past experience and speeding up the learning process. The principle of importance sampling requires $r_t(\theta)$ (expressed as Equation (32)) to be as close to 1 as

possible, and here, a restriction is placed on the objective function of the online network, see Equation (31), in which ϵ is generally taken to be 0.2 [34].

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}. \tag{32}$$

Optimal policies can be obtained by optimizing the metric based on gradient-based algorithms: $\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\theta_t)$, where $\nabla_\theta J$ is the gradient of J with respect to θ, t is the time step, and α is the optimization rate. Summarizing the above, the algorithm training process is presented in Algorithm 1.

Algorithm 1 Training the algorithm of PPO.

Initialize Clip factor ϵ, discount factor γ, learning rate λ_a and λ_c, update times of each training step Rp, actor network θ, critic network ω, and replay buffer;
For episode = 1, . . ., 500,
Obtain the Observation from the environment;
Execute the action a[n];
Evaluate reward r[n], policy entropy S(s[n]), the log-probability of a[n],
and store these variables in the experience buffer;
Store the transition {s[n], a[n], r[n], s[n+1]} into experience buffer;
If *mod (n, N) = 0* ***then*** reset the training environment;
end if n = n + 1;
Calculate the advantage function A[n];
for up = 1, . . ., Rp **do**
Calculate the current policy entropy S(s[n])
Calculate the update ratios $r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta old}(a_t|s_t)$
Update actor θ and critic ω with the transitions in the experience buffer according to $\theta \leftarrow \theta - \lambda_a \nabla^{\text{actor}} J(\theta)$ and $\omega \leftarrow \omega - \lambda_b \nabla^{\text{critic}} J(\theta)$;
Update the policy entropy and log probability in the experience buffer;
end for Clear experience buffer;
end for Output policy parameters θ and critic parameters ω.

5. Performance Evaluation

This section evaluates the performance of the proposed PPO-based D2D-assisted computation offloading and resource allocation scheme in a realistic MEC environment.

5.1. Experiment Setup

In this study, we deployed a real MEC network environment. As shown in Figure 4, this edge network consists of two APs, three UEs, and two MEC servers. The APs and UEs were all modeled by a Raspberry Pi 4B, which consumes an average of 6.4 W in operation and 2.7 W in idle mode [35]. All UEs were equipped with RPi Camera V2 model 8-megapixel external cameras and were randomly dispersed in an environment with a radius of approximately 5 m centered on the AP. The UEs had a local computational power of 1.5 GHz. The two MEC servers were simulated by personal laptops, which were configured with an 11th Intel(R) Core (TM) i5@2.40GHz CPU, NVIDIA 1650Ti GPU, and 16 GB RAM.

On top of putting the MEC environment in place, we introduced a real-time surveillance recognition system based on the YoloV5 algorithm [36] as a computational task for each UE, with the size of a computational task ranging from 370 kb to 450 kb, generating 20 unprocessed 256 × 256 images per second as a pending task. After testing, all UEs and MEC servers fulfilled the requirements for running the YOLOv5 object detection algorithm.

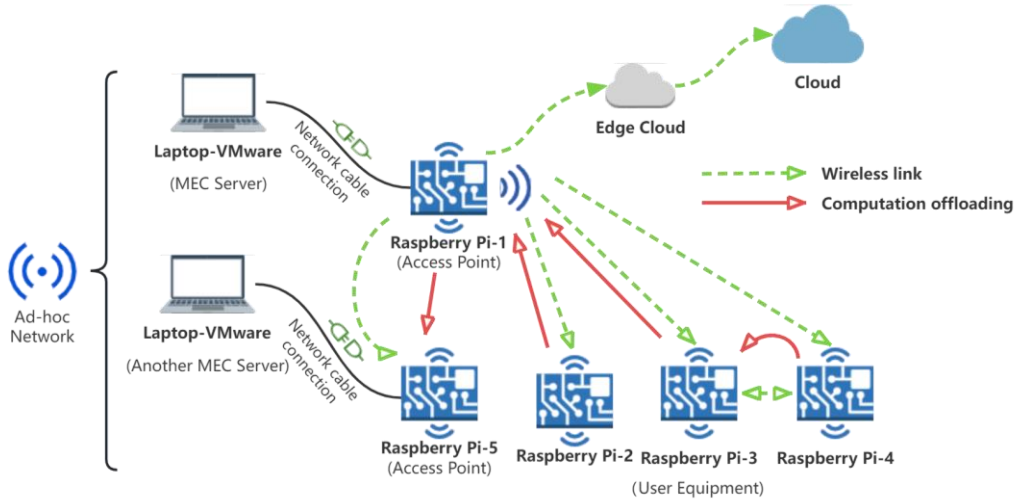


Figure 4. Experimental environment.

Subsequently, we analyzed the actual performance of the random scheme, DQN scheme, A2C scheme, and PPO scheme in edge computing. The network environment parameters and the training parameters of the four schemes are shown in Table 1. The user devices were uniformly distributed in an area with a radius of approximately 5 m. Our proposed PPO Scheme consists of the actor, a policy network responsible for generating actions, together with the critic, a value function network for estimating the value or the advantage of a state to guide the policy updates. The structure of the actor network consists of an input layer consisting of 15 neurons, two hidden layers containing (128, 64) neurons, and an output layer containing a fully connected layer. The weights of the actor network are updated by maximizing the agent's objective function, whereas the weights of the critic network are updated by minimizing the loss of the mean-square error between the estimate of the value function and the true return. The activation function was ReLU, the current empirical data and the Adam optimizer were sampled during the weight update process, and the learning rates of the actor network and the critic network were set to 1×10^{-4} and 5×10^{-4}, respectively.

Table 1. Environment settings.

Description	Parameters	Values
The bandwidth of the wireless channel	W	10 MHz
The maximum transmission power of UEs	P_i^{UE}	14.5 dBm
The maximum transmission power of the MEC server	$P_i^{mec\ server}$	20 dBm
The data size of the task	m_i	370–450 kb
The computational power of UEs	f_i^{UE}	1.5 GHz/s
The computational power of the MEC server	$f_i^{mec\ server}$	4.2 GHz/s
Transmission rate of UEs	$R_{i,b}$	-
Transmission rate of the MEC server	$R_{i,c}$	-

We compared our PPO-based approach against three baseline algorithms: random scheme, DQN scheme, and A2C scheme. These comparisons aimed to demonstrate the superiority of our approach in various MEC scenarios. We compared the proposed PPO algorithmic scheme with a random offloading strategy, DQN strategy, and A2C strategy.

- Random Scheme: Each task randomly selects the execution device, including edge servers, migration to other edge server computation, D2D computation, and local computation.
- DQN Scheme: The DQN (deep Q-network) algorithm [37] is a deep reinforcement learning method that combines traditional Q-learning and deep learning techniques.

Its core features include the use of deep neural networks to approximate the Q-function so as to learn strategies directly from the complex.

- A2C Scheme: The A2C (advantage actor–critic) algorithm is a widely used method in the field of reinforcement learning [38], which is a variant of the AC algorithm. The A2C algorithm improves this framework by introducing the advantage function, which improves learning efficiency and stability.
- PPO Scheme: PPO can be considered an improved version of the AC algorithm that introduces a limit on the magnitude of policy updates to prevent excessive policy changes in a single update [39]. This approach aims to avoid excessive jumps in the strategy space, thus improving the stability of the learning process.

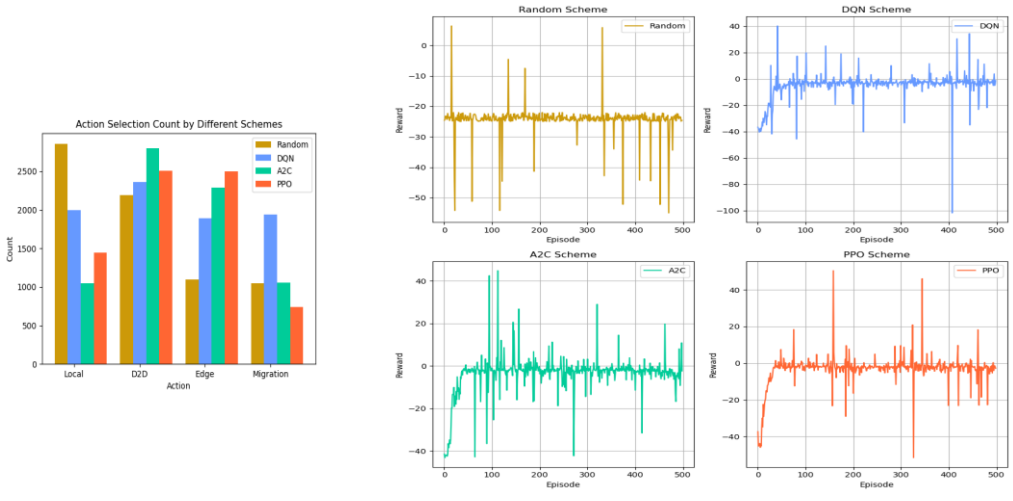
For the above four frames, the training parameters are shown in Table 2.

Table 2. Hyperparameters of algorithms.

Parameters	Values
Learning rate λ_a, λ_c (PPO and A2C)	1×10^{-4}, 5×10^{-4}
Learning rate λ(DQN)	1×10^{-4}
Discount factor γ	0.98
Clipping factor ϵ	0.2
Entropy factor ψ	0.01
Replay buffer size of DQN	10,000
Sample size in the empirical buffer	64
Activation functions and optimizers	ReLu, Adam
Hidden sizes	128, 64

5.2. Experimental Analysis

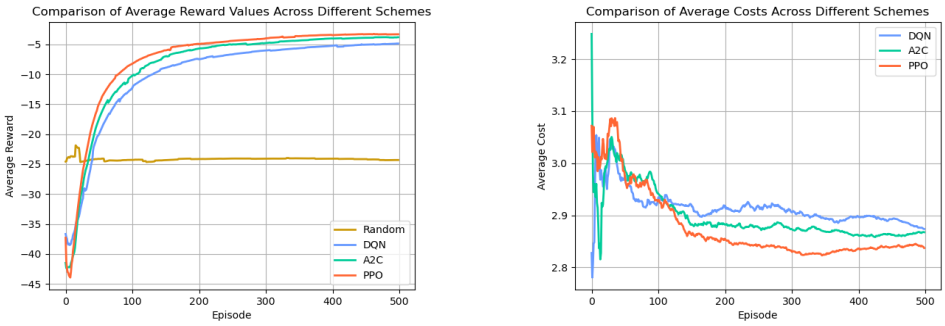
In this study, we analyzed the performance of four frameworks during the training process by recording their action selection and reward values, as illustrated in Figure 5a,b. Regarding convergence, the reward value of the PPO algorithm stabilized around step 340, while the A2C algorithm and the DQN algorithm showed stabilization at approximately steps 400 and 430, respectively. However, the random policy consistently demonstrated low reward values from the 57th step onward and was thus not considered for convergence analysis. Through an in-depth examination of the result, it became evident that reinforcement learning algorithms play a crucial role in offloading decisions. Notably, the PPO algorithm exhibited superior convergence compared to the other three benchmarked algorithms, as shown in Figure 5b.



(**a**) (**b**)

Figure 5. (**a**) Action selection count by different schemes; (**b**) training process for the four schemes.

Figure 6a shows the line graphs of the average reward value of the four schemes. The random scheme was in a low reward state compared with the other three schemes, and the average reward fluctuated slightly around −24 after 57 steps; the average reward value of the DQN scheme was around −5.0 after convergence, the average reward value of the A2C scheme was around −3.7 after convergence, and the average reward was still on the rise after 400 steps, while the average reward value of the PPO scheme was over −5.0 after convergence. The PPO scheme had an average reward value of around −3.3 after convergence and was relatively smooth, better than the other three baselines. In terms of training stability, the stability of the PPO algorithm was better than the other three baselines within 0–150 steps, DQN was more stable between 151 and 400 steps, and the A2C algorithm was smoother after 400 steps. Overall, DQN performed better in terms of stability. From Figures 5b and 6a, we observed that the DRL algorithm effectively adapts to complex edge computing environments through end-to-end optimization and thus achieves better performance compared to the randomness strategy. Whereas the randomness strategy lacks targeted decisions, and its results do not provide meaningful information, we further focused on the DQN algorithm, A2C algorithm, and PPO algorithm.

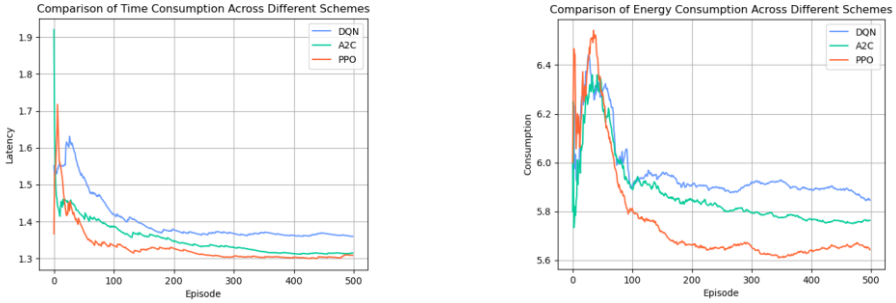


(**a**) (**b**)

Figure 6. (**a**) Comparison of average reward values across different schemes; (**b**) comparison of average costs across different schemes.

Secondly, as shown in Figure 6b, we recorded the weighted average of the delay and energy consumption during the training process. Time cost represents the cumulative sum of transmission delay and computation delay. Energy cost encompasses the combined sum of transmission energy consumption and computation energy consumption. The ultimate average cost is the weighted sum of time cost and energy cost, with the respective weight parameters for time cost and energy cost determined by the neural network. PPO had the highest average total cost in 0–30 steps, and as the algorithms were trained, it became gradually smoother after 300 steps at around 2.87, which was the lowest average total cost compared to the other algorithms, and the overall view of the weighted average of the total cost was more stable.

As shown in Figure 7a,b, the performance metrics of delay and energy consumption were compared among the different algorithms. The DQN algorithm demonstrated relative stability in both delay and energy consumption; however, its learning efficacy was marginally outpaced by the A2C and PPO algorithms. After 300 steps, the A2C algorithm's performance in the delay reduction closely approximated that of the PPO algorithm, yet it lagged slightly behind in energy consumption efficiency. Overall, when evaluating the learning outcomes, the PPO algorithm emerged as the superior performer in comparison to the DQN and A2C models, particularly in terms of optimizing the trade-off between the delay and energy consumption.



(**a**) (**b**)

Figure 7. (**a**) Comparison of time consumption across different schemes; (**b**) comparison of energy consumption across different schemes.

6. Conclusions

In this paper, we proposed a PPO-based D2D-assisted computation offloading and resource allocation scheme for multi-access edge computing environments. The proposed approach leverages the actor–critic based on the PPO method in reinforcement learning to optimize offloading decisions based on the task attributes and device performance, resulting in reduced latency and lower energy consumption. Additionally, the collaborative offloading strategy between the D2D and MEC servers greatly improved the utilization of the edge computational resources. The implementation of this scheme marks a significant advancement in addressing the complex challenges inherent in the MEC networks, particularly relevant in the evolving 5G landscape. However, we recognize the limitations in the current study: one being the time-varying nature of the communication model channel parameters as influenced by the environment and the other being the protection of sensitive user information and data during D2D communication. In the future, we will extend our work. On the one hand, we will use real-time monitoring and sensing techniques to accurately measure the dynamic characterization of channel parameters, and on the other hand, we will incorporate privacy-preserving aspects into the common offloading process within D2D-assisted MEC networks.

Author Contributions: Conceptualization, W.L. and C.W.; methodology, C.W., W.L., M.L., Y.L. and C.Z.; software, C.Z. and Y.L.; validation, C.Z. and Y.L.; formal analysis, W.L., C.W. and M.L.; investigation, W.L., C.W. and M.L.; resources, W.L. and C.W.; writing—original draft preparation, C.Z. and Y.L.; writing—review and editing, W.L., C.Z. and Y.L.; visualization, C.Z. and Y.L.; supervision, C.W. and W.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research and development work was supported by the MIC/SCOPE #JP235006102.

Data Availability Statement: The data can be shared up on request. The data are not publicly available due to privacy protection considerations.

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

References

1. How Many Smartphones Are in the World? Available online: https://www.bankmycell.com/blog/how-many-phones-are-in-the-world#part-3) (accessed on 10 December 2023).
2. Chen, C.; Liu, B.; Wan, S.; Qiao, P.; Pei, Q. An Edge Traffic Flow Detection Scheme Based on Deep Learning in an Intelligent Transportation System. *IEEE Trans. Intell. Transp. Syst.* **2020**, *22*, 1840–1852. [CrossRef]
3. Taleb, T.; Samdanis, K.; Mada, B.; Flinck, H.; Dutta, S.; Sabella, D. On multi-access edge computing: A survey of the emerging 5G network edge cloud ar-chitecture and orchestration. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1657–1681. [CrossRef]
4. Anon. The Standard, News from ETSI—Issue 2. 2017. Available online: https://www.etsi.org/images/files/ETSInewsletter/etsinewsletter-issue2-2017.pdf (accessed on 10 December 2023).
5. Sun, M.; Xu, X.; Huang, Y.; Wu, Q.; Tao, X.; Zhang, P. Resource Management for Computation Offloading in D2D-Aided Wireless Powered Mobile-Edge Computing Networks. *IEEE Internet Things J.* **2020**, *8*, 8005–8020. [CrossRef]
6. Yang, X.; Luo, H.; Sun, Y.; Guizani, M. A Novel Hybrid-ARPPO Algorithm for Dynamic Computation Offloading in Edge Computing. *IEEE Internet Things J.* **2022**, *9*, 24065–24078. [CrossRef]

7. Guo, H.; Wang, Y.; Liu, J.; Liu, C. Multi-UAV Cooperative Task Offloading and Resource Allocation in 5G Advanced and Beyond. *IEEE Trans. Wirel. Commun.* **2023**. [CrossRef]
8. Zhou, S.; Jadoon, W. Jointly Optimizing Offloading Decision and Bandwidth Allocation with Energy Constraint in Mobile Edge Computing Environment. *Computing* **2021**, *103*, 2839–2865. [CrossRef]
9. Xiao, Z.; Shu, J.; Jiang, H.; Lui, J.C.S.; Min, G.; Liu, J.; Dustdar, S. Multi-Objective Parallel Task Offloading and Content Caching in D2D-aided MEC Networks. *IEEE Trans. Mob. Comput.* **2022**, *22*, 6599–6615. [CrossRef]
10. Abbas, N.; Sharafeddine, S.; Mourad, A.; Abou-Rjeily, C.; Fawaz, W. Joint computing, communication and cost-aware task offloading in D2D-enabled Het-MEC. *Comput. Netw.* **2022**, *209*, 108900. [CrossRef]
11. Li, Z.; Hu, H.; Hu, H.; Huang, B.; Ge, J.; Chang, V. Security and energy-aware collaborative task offloading in D2D communication. *Future Gener. Comput. Syst.* **2021**, *118*, 358–373. [CrossRef]
12. Keshavarznejad, M.; Rezvani, M.H.; Adabi, S. Delay-aware optimization of energy consumption for task offloading in fog envi-ronments using metaheuristic algorithms. *Clust. Comput.* **2021**, *24*, 1825–1853. [CrossRef]
13. Khoobkar, M.H.; Dehghan Takht Fooladi, M.; Rezvani, M.H.; Sadeghi, M.M.G. Partial offloading with stable equilibrium in fog-cloud envi-ronments using replicator dynamics of evolutionary game theory. *Clust. Comput.* **2022**, *25*, 1393–1420. [CrossRef]
14. Islam, A.; Debnath, A.; Ghose, M.; Chakraborty, S. A Survey on Task Offloading in Multi-access Edge Computing. *J. Syst. Arch.* **2021**, *118*, 102225. [CrossRef]
15. Mehrabi, M.; You, D.; Latzko, V.; Salah, H.; Reisslein, M.; Fitzek, F.H.P. Device-enhanced MEC: Multi-access edge computing (MEC) aided by end device compu-tation and caching: A survey. *IEEE Access* **2019**, *7*, 166079–166108. [CrossRef]
16. Abbas, A.; Raza, A.; Aadil, F.; Maqsood, M. Meta-heuristic-based offloading task optimization in mobile edge computing. *Int. J. Distrib. Sens. Netw.* **2021**, *17*, 15501477211023021. [CrossRef]
17. Lin, J.; Huang, L.; Zhang, H.; Yang, X.; Zhao, P. A Novel Lyapunov based Dynamic Resource Allocation for UAVs-assisted Edge Computing. *Comput. Netw.* **2022**, *205*, 108710. [CrossRef]
18. Chang, S.; Li, C.; Deng, C.; Luo, Y. Low-latency controller load balancing strategy and offloading decision generation algorithm based on lyapunov optimization in SDN mobile edge computing environment. *Clust. Comput.* **2023**, 1–21. [CrossRef]
19. Zabihi, Z.; Eftekhari Moghadam, A.M.; Rezvani, M.H. Reinforcement Learning Methods for Computation Offloading: A Sys-tematic Review. *ACM Comput. Surv.* **2023**, *56*, 17.
20. Gao, H.; Wang, X.; Ma, X.; Wei, W.; Mumatz, S. Com-DDPG: A multiagent reinforcement learning-based offloading strategy for mobile edge computing. *arXiv* **2020**, arXiv:2012.05105.
21. Silva, C.; Magaia, N.; Grilo, A. Task Offloading Optimization in Mobile Edge Computing based on Deep Reinforcement Learning. In Proceedings of the Int'l ACM Conference on Modeling Analysis and Simulation of Wireless and Mobile Systems, Montreal, QC, Canada, 30 October–3 November 2023; pp. 109–118.
22. Han, Y.; Zhu, Q. Joint Computation Offloading and Resource Allocation for NOMA-Enabled Multitask D2D System. *Wirel. Commun. Mob. Comput.* **2022**, *2022*, 5349571. [CrossRef]
23. Li, G.; Chen, M.; Wei, X.; Qi, T.; Zhuang, Q. Computation offloading with reinforcement learning in d2d-mec network. In Proceedings of the 2020 International Wireless Communications and Mobile Computing (IWCMC), Limassol, Cyprus, 15–19 June 2020; pp. 69–74.
24. Lin, Q.; Wang, F.; Xu, J. Optimal Task Offloading Scheduling for Energy Efficient D2D Cooperative Computing. *IEEE Commun. Lett.* **2019**, *23*, 1816–1820. [CrossRef]
25. Guan, X.; Lv, T.; Lin, Z.; Huang, P.; Zeng, J. D2D-Assisted Multi-User Cooperative Partial Offloading in MEC Based on Deep Reinforcement Learning. *Sensors* **2022**, *22*, 7004. [CrossRef] [PubMed]
26. Liu, W.; Xu, Y.; Qi, N.; Yao, J.; Zhang, Y.; He, W. Joint computation offloading and resource allocation in UAV swarms with multi-access edge computing. In Proceedings of the 2020 International Conference on Wireless Communications and Signal Processing (WCSP), Nanjing, China, 21–23 October 2020; pp. 280–285.
27. Fan, N.; Wang, X.; Wang, D.; Lan, Y.; Hou, J. A collaborative task offloading scheme in d2d-assisted fog computing networks. In Proceedings of the 2020 IEEE Wireless Communications and Networking Conference (WCNC), Seoul, Republic of Korea, 25–28 May 2020; pp. 1–6.
28. Ferrer, A.J.; Marquès, J.M.; Jorba, J. Towards the decentralised cloud: Survey on approaches and challenges for mobile, ad hoc, and edge computing. *ACM Comput. Surv. (CSUR)* **2019**, *51*, 1–36. [CrossRef]
29. Al-Absi, M.A.; Al-Absi, A.A.; Sain, M.; Lee, H. Moving Ad Hoc Networks—A Comparative Study. *Sustainability* **2021**, *13*, 6187. [CrossRef]
30. Silver, D.; Huang, A.; Maddison, C.J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. Mastering the game of Go with deep neural networks and tree search. *Nature* **2016**, *529*, 484–489. [CrossRef] [PubMed]
31. Hao, W.; Yang, S. Small Cell Cluster-Based Resource Allocation for Wireless Backhaul in Two-Tier Heterogeneous Networks with Massive MIMO. *IEEE Trans. Veh. Technol.* **2017**, *67*, 509–523. [CrossRef]
32. Luo, Q.; Luan, T.H.; Shi, W.; Fan, P. Deep Reinforcement Learning Based Computation Offloading and Trajectory Planning for Multi-UAV Cooperative Target Search. *IEEE J. Sel. Areas Commun.* **2022**, *41*, 504–520. [CrossRef]
33. Wu, C.; Bi, W.; Liu, H. Proximal policy optimization algorithm for dynamic pricing with online reviews. *Expert Syst. Appl.* **2023**, *213*, 119191. [CrossRef]

34. Zhu, W.; Rosendo, A. Proximal policy optimization smoothed algorithm. *arXiv* **2020**, arXiv:2012.02439.
35. Power Consumption Benchmarks of Raspberry Pi 4B. Available online: https://www.pidramble.com/wiki/benchmarks/power-consumption (accessed on 10 December 2023).
36. Jiang, P.; Ergu, D.; Liu, F.; Cai, Y.; Ma, B. A Review of Yolo Algorithm Developments. *Procedia Comput. Sci.* **2022**, *199*, 1066–1073. [CrossRef]
37. Tong, Z.; Chen, H.; Deng, X.; Li, K.; Li, K. A scheduling scheme in the cloud computing environment using deep Q-learning. *Inf. Sci.* **2020**, *512*, 1170–1191. [CrossRef]
38. François-Lavet, V.; Henderson, P.; Islam, R.; Bellemare, M.G.; Pineau, J. An introduction to deep reinforcement learning. *Found. Trends Mach. Learn.* **2018**, *11*, 219–354. [CrossRef]
39. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.