







Article

Joint Optimization of Service Migration and Resource Allocation in Mobile Edge–Cloud Computing

Zhenli He ^{1,2,3} , Liheng Li ¹ , Ziqi Lin ¹ , Yunyun Dong ^{1,3} , Jianglong Qin ^{1,2}  and Keqin Li ^{4,*} 

¹ School of Software, Yunnan University, Kunming 650504, China; hezl@ynu.edu.cn (Z.H.); liliheng@stu.ynu.edu.cn (L.L.); linziqi@stu.ynu.edu.cn (Z.L.); dongyy929@ynu.edu.cn (Y.D.); qinjianglong@ynu.edu.cn (J.Q.)

² Yunnan Key Laboratory of Software Engineering, Yunnan University, Kunming 650504, China

³ Engineering Research Center of Cyberspace, Yunnan University, Kunming 650504, China

⁴ Department of Computer Science, State University of New York at New Paltz, New Paltz, NY 12561, USA

* Correspondence: lik@newpaltz.edu

Abstract: In the rapidly evolving domain of mobile edge–cloud computing (MECC), the proliferation of Internet of Things (IoT) devices and mobile applications poses significant challenges, particularly in dynamically managing computational demands and user mobility. Current research has partially addressed aspects of service migration and resource allocation, yet it often falls short in thoroughly examining the nuanced interdependencies between migration strategies and resource allocation, the consequential impacts of migration delays, and the intricacies of handling incomplete tasks during migration. This study advances the discourse by introducing a sophisticated framework optimized through a deep reinforcement learning (DRL) strategy, underpinned by a Markov decision process (MDP) that dynamically adapts service migration and resource allocation strategies. This refined approach facilitates continuous system monitoring, adept decision making, and iterative policy refinement, significantly enhancing operational efficiency and reducing response times in MECC environments. By meticulously addressing these previously overlooked complexities, our research not only fills critical gaps in the literature but also enhances the practical deployment of edge computing technologies, contributing profoundly to both theoretical insights and practical implementations in contemporary digital ecosystems.

Keywords: Advantage Actor–Critic; deep reinforcement learning; mobile edge–cloud computing; resource allocation; service migration



Citation: He, Z.; Li, L.; Lin, Z.; Dong, Y.; Qin, J.; Li, K. Joint Optimization of Service Migration and Resource Allocation in Mobile Edge–Cloud Computing. *Algorithms* **2024**, *17*, 370. <https://doi.org/10.3390/a17080370>

Academic Editor: Grammati Pantziou

Received: 30 June 2024

Revised: 12 August 2024

Accepted: 19 August 2024

Published: 21 August 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

1.1. Motivation

In recent years, the rapid expansion of Internet of Things (IoT) devices and mobile applications has catalyzed the development of mobile edge–cloud computing (MECC). This innovative paradigm combines the extensive computational resources of cloud computing with the immediacy of edge computing to meet essential requirements for low latency, high reliability, and superior quality of service (QoS) [1]. By harnessing the robust data handling and computational capabilities of cloud data centers, MECC efficiently manages large datasets and executes complex computations beyond the processing power of edge devices alone [2,3]. Additionally, MECC significantly reduces latency by processing data closer to their source, enhancing performance for latency-sensitive applications such as autonomous driving, augmented reality (AR), virtual reality (VR), and real-time gaming [4–8].

MECC also improves system reliability by enabling edge nodes to autonomously manage critical operations, ensuring uninterrupted service even during network disruptions. Furthermore, the architecture of MECC supports dynamic and scalable resource allocation, optimizing cloud resources for computationally intensive tasks while delegating real-time processing to edge servers (ESs) [9]. This adaptability makes MECC highly suitable for

modern digital ecosystems, capable of efficiently handling variable network loads and diverse application demands while seamlessly integrating smart technologies.

Despite its advancements MECC faces significant challenges, primarily stemming from user mobility and the dynamic computational demands of contemporary mobile applications, which can lead to potential delays and instabilities, adversely impacting QoS [10,11]. Common issues arise when users move to locations farther from their current ES, necessitating the offloading of tasks back to the original server to maintain service continuity. This often results in increased communication delays and can substantially degrade user experience, particularly in latency-sensitive applications. While recent advancements in dynamic computational offloading and resource allocation have significantly mitigated issues related to user mobility by optimizing offloading and allocation decisions to minimize response times, there remains a notable gap in fully addressing the dependencies of task execution on the underlying environments and user contexts at ESs.

Current research has responded to these operational challenges by prioritizing service migration strategies that relocate task execution environments and user contexts to ESs closer to the user's real-time location, thereby maintaining or enhancing QoS [12]. However, these studies often overlook the critical impact of the reduction in computational time caused by migration processes, which can be detrimental to the performance of time-sensitive services. Furthermore, the interplay between service migration decisions and resource allocation strategies remains underexplored. Effective strategies should not only minimize data transfer delays by considering user proximity but also assess the computational capacities of various ESs to ensure that after migration the resources are adequate to handle the relocated services without causing service timeouts. Although some studies have acknowledged the complex interdependence between service migration and resource allocation, they often fail to account for scenarios where tasks are not completed before migration. This oversight can lead to data loss, increased delays, and significantly compromised QoS.

In summary, while significant progress has been made in addressing various aspects of migrating services and allocating resources within MECC environments, critical gaps remain. Developing comprehensive strategies that simultaneously enhance the efficiency of service migration and resource distribution, considering unfinished tasks and their contexts, is crucial. Addressing these gaps will significantly enhance the efficiency and effectiveness of MECC environments, aligning their capabilities with the evolving demands of contemporary digital ecosystems.

1.2. Our Contributions

In the rapidly evolving domain of MECC, the critical interplay between service migration and resource allocation commands increasing attention. This paper explores these intricacies by establishing robust service migration and computational models, enhanced by a reinforcement learning-based strategy aimed at minimizing service response times. Addressing gaps identified in previous research, particularly the overlooked aspects of user context migration and the inadequate attention to computational resource allocation, our research introduces an innovative framework. It adeptly manages user mobility and the need for uninterrupted service by migrating unfinished user data and context. Moreover, the framework dynamically adjusts service provisions and resource allocations in real time, significantly enhancing QoS within strict time constraints. This approach not only ensures service continuity but also improves responsiveness across MECC environments.

The main contributions of this paper are summarized as follows:

- **Comprehensive optimization of service migration and resource allocation:** We provide a thorough problem formulation that simultaneously optimizes service migration and resource allocation within MECC frameworks. Addressing the challenges of heterogeneous ES environments, our study tackles the intertwined issues of user mobility and fluctuating computational demands. The optimization strategically aims to mini-

mize average response times, substantially enhancing QoS while meeting rigorous temporal constraints.

- **Strategic transformation into a Markov decision process (MDP):** Moving from theoretical models, this paper adeptly transforms the joint optimization challenge into an MDP. We introduce a novel deep reinforcement learning (DRL)-based algorithm to tackle this MDP, autonomously adapting migration and resource allocation strategies without relying on prior knowledge of system states.
- **Rigorous evaluation through simulation:** The efficacy and robustness of our proposed DRL-based dynamic migration and resource allocation strategy are rigorously tested through comprehensive simulations. Performance metrics, including task failure rate and average task response delay, serve as benchmarks. The results demonstrate that our DRL-based approach sustains high service quality and markedly reduces average response delays, thereby outperforming established benchmarks in this paper.

The remainder of the paper is organized as follows: Section 2 summarizes the related work. Section 3 describes the system model in detail, which includes the migration model, computational model, and problem description. Section 4 provides a detailed description of the proposed A2C-based approach. The results of the simulation experiments are discussed in Section 5. Finally, Section 6 summarizes the paper and outlines future research directions.

2. Related Works

In this section, we review existing research on the joint optimization of dynamic computation offloading and resource allocation, alongside the optimization of service migration policies in MECC environments. We then highlight the distinctions between our study and the prior works, elucidating the unique contributions and advancements our research offers in this domain.

2.1. Joint Optimization of Dynamic Computation Offloading and Resource Allocation

This section reviews existing research on strategies for task offloading and resource management in dynamic network environments.

Liu et al. [13] considered the mobility characteristics of user equipment (UE) and proposed a dual time-scale framework that resolves user-server association problems by incorporating long-term channel interference, workload, and server computational constraints with short-term dynamic task offloading and resource allocation. Wang et al. [14] developed a decentralized offloading framework, accommodating mobile users dynamically entering or exiting an MEC system, and adapting to their varying offloading demands. Yang et al. [15] introduced a priority-driven multi-agent (PDMA) cooperative task offloading algorithm to address the dynamic characteristics of task arrivals, mobility of devices, and load imbalances across ESs. Liang et al. [16] investigated joint task cache placement and offloading in mobile edge computing systems characterized by dynamic task arrivals. Fang et al. [17] devised a dynamic task offloading algorithm using DRL, considering dependency relationships among user tasks to optimize task offloading and resource allocation decisions effectively. This algorithm seeks to minimize task completion time and reduce device energy consumption amid channel variations. Zhu et al. [18] proposed a scheduling algorithm that integrates communication and dynamic tasks by considering the vehicle's mobility patterns and task sizes in vehicular networks supported by an intelligent reflecting surface (IRS) for MEC services. Ma et al. [19] based their joint offloading strategy on vehicle mobility patterns, aiming to minimize the weighted sum of execution time and computation costs, considering both response delay and economic factors. Dang et al. [20] introduced a task offloading cost model for scenarios involving multiple vehicles and MEC servers, utilizing the DDPG algorithm to make decisions that minimized the overall system's task processing costs. Liao et al. [21] explored task execution queues and priorities within a multi-MEC server environment, dividing computation offloading into power scheduling and task offloading phases. In the power scheduling phase, the focus is on minimizing energy consumption through optimal transmission power and CPU frequency settings, while

the task offloading phase aims to reduce execution latency through strategic offloading decisions. Huang et al. [22] addressed a dynamic Internet of Vehicles (IoV) architecture that supports both MEC and cloud computing, employing a DRL-based algorithm to optimize task offloading and resource allocation for self-driving vehicles on straight roads. This algorithm aims to minimize processing costs by optimizing computational offloading and bandwidth allocation, adhering to processing delay and transmission rate constraints. Xu et al. [23] proposed a vehicular edge computing architecture utilizing non-orthogonal multiple access (NOMA), focusing on cooperative resource optimization among ESs to maximize the service ratio through game theory and convex optimization methods.

While recent advancements in dynamic computational offloading and resource allocation have significantly mitigated issues related to user mobility by optimizing offloading and allocation decisions to minimize response times, there remains a notable gap when it comes to fully addressing the dependencies of task execution on the underlying environments and user contexts at ESs. In scenarios where user mobility is rapid, optimizing task offloading alone proves insufficient. This is due to the fact that task execution often depends on specific characteristics of the ESs and user contexts. Consequently, integrating service migration strategies that dynamically adapt to these environmental and contextual dependencies is crucial. Such strategies not only complement the offloading process but are also essential to effectively reduce delays caused by task routing and result returns, thereby enhancing overall service quality.

2.2. Optimization of Service Migration Strategy

Numerous studies have focused on optimizing service migration policies to ensure QoS.

Liu et al. [24] aimed to efficiently manage the allocation of diverse heterogeneous resources and user tasks to maximize system utility in the context of vehicular edge computing. Their innovative hybrid computing offloading strategy, incorporating both vehicle-to-infrastructure and vehicle-to-vehicle communications, allows for service migration to other ESs when an ES's computing capacity is insufficient, thus achieving load balancing. Liang et al. [25] addressed mobility management across different time scales, proposing a framework that integrates service migration with transmission power adjustments. This strategy enables making service migration decisions at a broader time scale, while adjusting transmission power at a finer scale to support task offloading, aiming to minimize long-term energy consumption and ensure reliable computational offloading. Researchers in [26] introduced a digital twin edge network architecture, distinguishing between latency-sensitive and latency-insensitive tasks. By utilizing real-time and historical data to predict future user movements, they tailored service migration decisions to reduce costs while maintaining QoS. Peng et al. [27] considered the comprehensive migration costs associated with service migration, integrating both computing and communication costs. In dynamic networks, they employed reinforcement learning combined with transfer learning to derive effective migration strategies within a dynamic vehicular edge computing environment. Xu et al. [28] tackled user mobility by modeling the service scheduling problem in an MEC environment, proposing a service management method using a probabilistic approach to effectively reduce service delay and migration costs. Wang et al. [29] emphasized the importance of balancing benefits and service costs during migration, proposing a dynamic service migration algorithm based on DRL aimed at minimizing the weighted sum of service delay and migration costs. Li et al. [30] introduced an edge caching strategy balancing energy and latency, utilizing a deep neural network for predicting future request content, followed by determining an optimal caching placement with the branch-and-bound algorithm and refining service migration strategies using a DQN algorithm to reduce service latency. Chen et al. [31] used a DRL algorithm to optimize service migration decisions in ES, focusing on minimizing user-perceived latency and system energy consumption, addressing service interruptions caused by user mobility. Additionally, Chen et al. [32] developed a service migration optimization algorithm based on deep recursive Q-learning,

aiming to minimize both user latency and system energy consumption by considering user mobility and the coverage range of ESs.

Current research on service migration predominantly focuses on cost reduction but often overlooks the significant reduction in computational time during migration, which can adversely impact the performance of time-sensitive services. Moreover, the dynamics between service migration strategies and resource allocation decisions have not been extensively studied. Effective strategies must not only minimize data transfer delays by considering user proximity but also rigorously evaluate the computational capacities of various ESs. This ensures that the resources available post-migration are adequate to support the services without causing timeouts. Joint consideration of service migration and resource allocation is crucial for improving system performance and user experience.

2.3. Joint Optimization of Service Migration and Resource Allocation

This section explores the challenges associated with optimizing service migration and resource allocation in MECC environments, a topic that has received limited attention in existing research.

Liang et al. [33] addressed user mobility in cellular networks to ensure seamless task migration between base stations without compromising resource efficiency or link reliability. Their research optimized migration and handover policies by jointly managing computational and radio resources. The policy framework integrated virtualization, I/O interference between virtual machines, and challenges associated with wireless multi-access. They developed a solution based on relaxation and rounding that includes an optimal iterative algorithm and a novel integer-recovery design. This approach surpassed traditional rounding methods by leveraging derived problem properties and applying matching theory. Additionally, their study included "hotspot mitigation", aiming to redistribute the load from overloaded to idle servers or base stations. Simulation results validated the effectiveness of their policies in multi-cell MECC networks, demonstrating near-optimal performance in managing joint service migration and base station handover. Building on this, Liu et al. [34] proposed a method to reduce access latency for IoT users in MECC by jointly optimizing service migration and resource allocation. They introduce a Service Migration and Resource Allocation (SMRA) algorithm based on DRL, which accounts for the mobility of IoT users. This algorithm determines whether to migrate services, identifies optimal migration destinations, and allocates resources using the long short-term memory (LSTM) and the parameterized deep Q-network (PDQN) algorithms.

Despite the recognition of the interdependencies between service migration and resource allocation, existing studies often overlook scenarios where tasks are incomplete before migration. Such oversight can result in data loss and significant deterioration in QoS. Addressing these critical aspects is essential for developing more effective migration and resource allocation strategies, ultimately enhancing both system performance and user experience.

To underscore the novelty and uniqueness of our work, we compare our study with existing research in the field and identify several key distinctions:

- **Acknowledgment of migration delays and impacts:** Unlike existing studies that often overlook the critical impact of the reduction in computational time caused by migration processes, our research takes these factors into account. We analyze the direct consequences of migration processes on the operational efficiency of systems, ensuring a more comprehensive understanding of the migration dynamics.
- **Exploration of service migration and resource allocation interplay:** The interaction between service migration strategies and resource allocation decisions has not been thoroughly examined in prior research. Our study delves into this interplay, aiming to establish a balanced approach that optimizes both elements to improve overall system performance.
- **Consideration of incomplete tasks during migration:** While a few studies have begun to address the interdependencies between service migration strategies and

resource allocation decisions, they rarely consider scenarios where tasks are not completed before migration. This oversight can lead to significant challenges, such as the need to migrate unfinished task data and service contexts together, which can further complicate resource allocation strategies. Our work addresses this gap by incorporating these scenarios into our optimization model, aiming to minimize disruptions and enhance service quality.

By addressing these aspects, our study not only contributes to the academic understanding of service migration dynamics but also offers practical insights that can be applied to improve the responsiveness and efficiency of IoT applications in MECC environments.

3. System Model and Problem Definition

As illustrated in Figure 1, we examine an edge–cloud collaboration system comprising ESs and a single cloud server. In this system, users continuously move and offload tasks generated by their mobile devices to the edge servers for execution. The ESs then send the task information and users' location data to a centralized scheduling and resource allocation system, which is responsible for making decisions regarding migration and resource allocation. By transmitting only the necessary information to the centralized system for decision making, this approach effectively reduces network load. At the same time, it allows for easier access to global network information, enabling more efficient and reasonable resource allocation strategies [30].

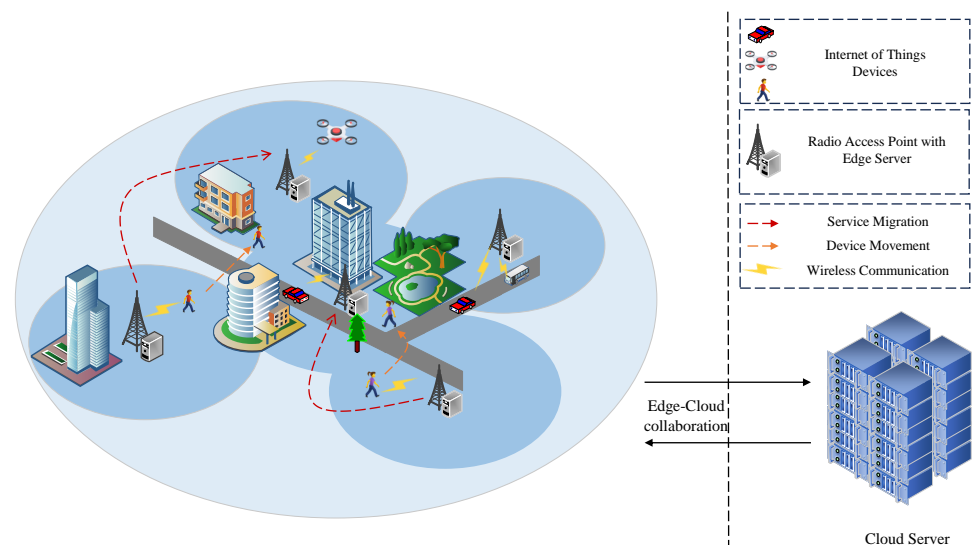


Figure 1. An example of an MECC environment.

3.1. System Model

We assume that each ES in the edge–cloud collaboration system is associated with a corresponding radio access network (RAN) node. Let $S = \{s_1, \dots, s_i, \dots, s_n\}$ denote the set of servers accessible to users via these RAN nodes. The required service functions (SFs) are deployed on the ESs to support emerging IoT applications. When users offload specific tasks to the ESs for processing, the corresponding SF creates a dedicated service instance (SI) to execute the task. The ESs support Λ types of application tasks, and their allocatable computational resources are represented as $\{p^1, \dots, p^i, \dots, p^m\}$, measured in GHz.

In the system, m users, denoted as $U = \{u_1, \dots, u_j, \dots, u_m\}$, are continuously active in the environment. Each user has installed Λ types of intelligent applications on their respective devices. It is assumed that ESs periodically update the information regarding the users they serve. Time is segmented into discrete intervals $T = \{1, \dots, t, \dots, T\}$, each with a duration of τ . After an ES updates the information of the users it serves in slot t , the user selects one of the Λ application types to generate a task for offloading to the ES. Then, the ES creates an SI for the offloaded task, uploads the task information at this

time slot, and inserts it into the task table $Task$ maintained by the control system, where $Task = \{Task_1, \dots, Task_t, \dots, Task_T\}$. Here, $Task_t = \{task_t^1, \dots, task_t^k, \dots, task_t^K\}$, where $Task_t$ represents the list of task information in time slot t . We define by $|task_t| = K_t$ the number of tasks in the system in time slot t . And $task_t^k$ defines $\{T_W, Rst_t^k, V_t^k, W, loc_t^k, X_t^k, Z_t^k\}$, which contains the response time constraints of the task type W , the remaining time of the task in time slot t , the remaining size of the task (measured in MB), the type of the task, the ESs associated with the user who offloaded the task, the location of the task, and $Z_t^k = 1$ indicates that the task is completed in time slot t ; otherwise, it remains unfinished.

When service migration is required, the corresponding SI, including task and context information, must be migrated from the origin server to the destination server. The model and assumptions for service migration are elaborated below.

3.2. Migration Model

For each task $task_t^k$, we use $X_t^k = [x_{k,t}^1, \dots, x_{k,t}^n]$ to denote the allocation of $task_t^k$ on ESs in time slot t , where $x_{k,t}^n = 1$ indicates that $task_t^k$ is allocated to server n ; otherwise, it signifies that $task_t^k$ is not on server n . In time slot t , $task_t^k$ needs to satisfy

$$C1 : \sum_{i=1}^n x_{k,t}^i = 1, \forall k \leq K_t, \quad \forall t \in T. \tag{1}$$

This stipulates that each task is allocated to exactly one ES. The ES hosting $task_t^k$ is denoted as follows:

$$\Psi_t^k = \arg \max_l X_t^k, \quad l \in \{1, \dots, n\}. \tag{2}$$

Migration occurs if $|\Psi_{t+1}^k - \Psi_t^k| > 0$, and we use Ψ_{t+1}^k to denote the location of the server where the task is located in the next time slot.

Whenever a migration occurs, the SI of $task_t^k$ must be migrated via the communication link of the RANs from the current ES to a new ES for continued processing. It is crucial to transfer the SI's state context during migration, which includes user-specific information, intermediate processing results, and more. Before resuming the task, the new ES must synchronize the SI's state context and restore the task's progress. As depicted in Figure 2, the migration process from suspension to restoration involves various time delays, including service suspension delay, synchronization delay, and service restoration delay. Thus, the total migration delay of SI can be expressed as follows:

$$D_t^k = h_t^k + w_t^k + r_{t+1}^k, \tag{3}$$

where h_t^k represents the service suspension delay associated with the SI context of $task_t^k$, w_t^k denotes the synchronization delay as the SI migrates from the source ES to the target ES, and r_{t+1}^k is the service restoration delay for restoring the task to its state before suspension.

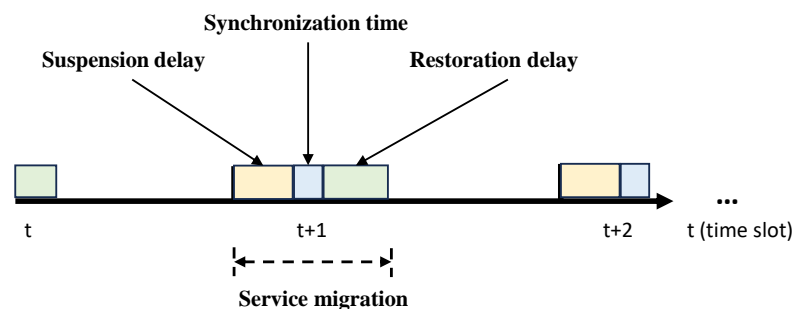


Figure 2. An example of the migration process.

During the service suspension and restoration processes, both the suspension and restoration delays are contingent upon the remaining size of the task, the processing inten-

sity required for either suspension or restoration, and the computing resources allocated to the task's SI. The service suspension delay can be articulated as

$$h_t^k = \frac{\rho_W^{sp}(V_{t+1}^k + V_W^m)}{p_{\Psi_t^k}}, \tag{4}$$

where ρ_W^{sp} represents the suspension processing intensity requirements of the application type W to which the $task_t^k$ belongs when the SI is paused (measured in CPU cycles required by processing per bit state context), V_W^m is the context size of the application type W (measured in MB), and $p_{\Psi_t^k}$ denotes the computational resources allocated to the SI on $s_{\Psi_t^k}$ (measured in cycles per second). And, the service restoration delay can be written as

$$r_{t+1}^k = \frac{\sigma_W^{sp}(V_{t+1}^k + V_W^m)}{p_{\Psi_{t+1}^k}}, \tag{5}$$

where σ_W^{sp} denotes the restoration processing intensity requirements of the application type to which $task_t^k$ belongs when the SI is restored, and $p_{\Psi_{t+1}^k}$ denotes the computational resources allocated to the SI on $s_{\Psi_{t+1}^k}$ in the next time slot. The term V_{t+1}^k is further discussed in Equation (14). The synchronization delay is the transmission delay determined by the remaining data size of the task and the bandwidth between the original server and the target server, which can be expressed as

$$w_t^k = \frac{V_{t+1}^k + V_W^m}{B_{\Psi_t^k, \Psi_{t+1}^k}}, \tag{6}$$

where $B_{\Psi_t^k, \Psi_{t+1}^k}$ is the link bandwidth between the source server $s_{\Psi_t^k}$ and the target server $s_{\Psi_{t+1}^k}$. Specifically, we have $D_t^k = 0$ if $|\Psi_{t+1}^k - \Psi_t^k| = 0$.

In time slot t , we consider the migration of SI into or out of ES s_i , where both service suspension and restoration demand computational resources. To avoid conflicts in resource usage, a period $\phi_{t,i}$ within each time slot is dedicated exclusively to these processes, expressed as

$$\phi_{t,i} = \text{Avg}\{C_{t,i}, V_{t,i}\}, \tag{7}$$

where $C_{t,i}$ represents the average delay for SIs migrating into target ES s_i , and $V_{t,i}$ represents the average delay for SIs migrating out of the original ES s_i .

When the remaining time $Rst_t^k < 0$ for $task_t^k$ and the task is incomplete, the SI of $task_t^k$ must migrate to the cloud server. Given the cloud server's distance from the ES, the synchronization delay for SI migration is $w_t^k = \frac{V_{t+1}^k + V_W^m}{B_{\Psi_t^k, \text{cloud}}} + \mu$, where μ represents the propagation delay. It is assumed that the cloud server possesses ample computational capacity to process the tasks and can return the results to the user in the subsequent time slot.

3.3. Computation and Communication Model

At the start of each time slot t , a user offloads a task from a specific application type to the associated ES s_i , where $loc_t^k = i$. Utilizing the Shannon formula, the maximum transmission rate for offloading the task can be expressed as

$$R_t^j = B \log_2 \left(1 + \frac{p_t^j g_t}{BN_0} \right), \tag{8}$$

where B denotes the wireless channel bandwidth between the user and the RAN, p_t^j is the transmission power of user u_j or the associated RAN node, g_t represents the free-space path loss, and N_0 is the noise power spectral density.

The uplink transmission delay of $task_t^k$ consists of two components: the delay for offloading the task to the nearest RAN node and the delay from this RAN node to ES s_i , which hosts the SI. This can be represented as

$$T_t^{tran,k} = \frac{V_t^k}{R_t^j} + \frac{V_t^k}{B_{loc_t^k,i}}. \tag{9}$$

Upon completion of $task_t^k$ after r_t^k time slots, we define the resultant data size as $V_{t+r_t^k}^{R,k} = \omega_{ret}^W V_W$, where V_W denotes the average task data size for the application type W of $task_t^k$, and ω_{ret}^W represents the ratio of the resultant data size to V_W . The return delay of the task result can be expressed as

$$T_{t+r_t^k}^{ret,k} = T_{t+r_t^k}^{back,k} + T_{t+r_t^k}^{down,k}, \tag{10}$$

where $T_{t+r_t^k}^{back,k}$ is the delay for returning the task result to the user's associated ES, and $T_{t+r_t^k}^{down,k}$ is the delay for transmitting the result to the user over the wireless channel, given by

$$T_{t+r_t^k}^{back,k} = \frac{V_{t+r_t^k}^{R,k}}{B_{\Psi_t^k,loc_{t+r_t^k}^k}}, \tag{11}$$

$$T_{t+r_t^k}^{down,k} = \frac{V_{t+r_t^k}^{R,k}}{R_{t+r_t^k}^j}. \tag{12}$$

Specifically, for results from the cloud, we have

$$T_{t+r_t^k}^{back,k} = \frac{V_{t+r_t^k}^{R,k}}{B_{cloud,loc_{t+r_t^k}^k}} + \mu. \tag{13}$$

The computational delay of a task on an ES depends on the task size, the computational intensity required for processing tasks, and the allocated computational resources. After $task_t^k$ has been executed for a period of time in time slot t , the remaining size of $task_t^k$ can be expressed as

$$V_{t+1}^k = V_t^k - \frac{p_{\Psi_t^k} \cdot (\tau - \phi_{t,\Psi_t^k})}{\kappa_W}, \tag{14}$$

where κ_W denotes the computational intensity required by processing $task_t^k$ of type W (measured in CPU cycles required per bit), and $p_{\Psi_t^k}$ denotes the computational resources allocated to $task_t^k$ on $s_{\Psi_t^k}$ in time slot t . The remaining time of $task_t^k$ after the execution of the task in time slot t is updated as $Rst_{t+1}^k = Rst_t^k - \tau$. The sum of the resources allocated to all tasks on s_i must satisfy its maximum resource constraint, i.e.,

$$C2 : \sum_{k=1}^{K_t} p_{\Psi_t^k} \leq p^i, \quad \forall i \in \{1, \dots, n\}, \quad \forall t \in T. \tag{15}$$

All mathematical symbols employed in this paper up to this point are systematically organized and presented in Table 1, following their order of introduction in the text.

Table 1. List of defined notation.

Symbol	Definition
S	The set of ESs
U	The set of users
T	The set of time slots
τ	The duration of the time slot t
Λ	Number of task types generated by users
$Task$	A list of tasks offloaded by users at different time slots
$Task_t^k$	List of offloaded task information for a single time slot t
$task_t^k$	The state about task k during time slot t
T_W	The response time constraints of application W
Rst_t^k	Remaining time of $task_t^k$ in time slot t
V_t^k	Remaining size of $task_t^k$ in time slot t
p^i	The total computing resources possessed by server s_i
loc_t^k	The ES associated with the user who offloaded $task_t^k$
X_t^k	The allocation of $task_t^k$ on ESs in time slot t
Z_t^k	Indicator of task completion
Ψ_t^k	ES where the $task_t^k$ is located
D_t^k	Migration delay of $task_t^k$ in time slot t
h_t^k	Suspension delay of $task_t^k$ in time slot t
w_t^k	Synchronization delay of $task_t^k$ in time slot t
r_{t+1}^k	Restoration delay of $task_t^k$ in time slot $t + 1$
$p_{\Psi_t^k}^k$	The computational resources allocated to $task_t^k$
ρ_{sp}^W	The computational intensity required to suspend service W
σ_{sp}^W	The computational intensity required to restore service W
V_W^m	The context size to be migrated for service W
$B_{\Psi_t^k, \Psi_{t+1}^k}$	The wired bandwidth between service $s_{\Psi_t^k}$ and target service $s_{\Psi_{t+1}^k}$
$C_{t,i}$	The average migration delay for services migrating to server s_i
$V_{t,i}$	The average migration delay for services migrating out of server s_i
$\phi_{t,i}$	The average migration delay of server s_i
μ	Propagation delay between ESs and cloud server
R_t^j	Wireless transmission rate of user j in time slot t
V_W	Average task data size for application W
$T_t^{tran,k}$	The uplink transmission delay of $task_t^k$
ω_{ret}^W	The ratio of the result data size to the V_W
$T_{t+r_t^k}^{back,k}$	The delay in returning task results to the ES $s_{loc_{t+r_t^k}}$
$T_{t+r_t^k}^{down,k}$	The delay in downlinking the task results to the user
κ_W	The computational intensity of the task generated by application W

3.4. Problem Formulation

With the above notation and modeling, we can obtain the response delay of the task as

$$T_t^k = \begin{cases} r_t^k \cdot \tau + \phi_{t+r_t^k, \Psi_{t+r_t^k}^k} + \frac{\kappa_W V_{t+r_t^k}^k}{p_{\Psi_{t+r_t^k}^k}^k} + T_{t+r_t^k}^{ret,k}, \\ T_W + T_{t+r_t^k}^{ret,k}, & \text{if result from cloud.} \end{cases} \quad (16)$$

In this paper, we minimize the average response delay of tasks by optimizing the service migration and resource allocation policies. The combined challenge of optimizing

service migration and resource allocation in the MECC environment can be expressed as follows:

$$\mathcal{P} : \min_{X,P} \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=1}^T \frac{1}{|task_t|} \sum_{k=1}^{|task_t|} T_t^k, \tag{17a}$$

$$\text{s.t. } \sum_{i=1}^n x_{k,t}^i = 1, \forall k \leq K_t, \forall t \in T, \tag{17b}$$

$$\sum_{k=1}^{K_t} p_{\Psi_t^k} \leq p^i, \forall i \in \{1, \dots, n\}, \forall t \in T. \tag{17c}$$

4. Proposed A2C-Based Algorithm

In this section, we detail our proposed Advantage Actor–Critic (A2C)-based dynamic migration and resource allocation approach. We discuss the transformation of problem \mathcal{P} into an MDP and describe the A2C-based algorithm.

4.1. Problem Transformation

In MECC systems, the ultimate goal is to improve the long-term QoS for users by implementing service migration and resource allocation policies. Given that tasks in time slot $t + 1$ depend on the execution status of tasks in time slot t , and considering the dynamic nature, heterogeneity, and complexity due to cross-time-slot execution characteristics, we utilize reinforcement learning methods to address this problem.

Prior to applying reinforcement learning, we first transform the problem into an MDP, represented by the tuple $M = \{S, A, P, R, \gamma\}$. Here, S denotes the state space, A represents the action space, P is the state transition probability, R is the reward function, and γ signifies the discount factor. Within this MDP framework, the agent perceives the state s_t of the environment at time slot t , selects an action a_t from the action space to execute migration and resource allocation, and receives a reward r_t . The environment then transitions to the next state s_{t+1} based on P [35]. Further details of the MDP are as follows.

State: For any time slot t , the system’s state is represented by an array reflecting the real-time status of tasks in the MECC system. This array includes information such as the remaining time for tasks, remaining task size, task type, the ES associated with the user, and the ES where the task is located, i.e.,

$$s_t = [Rst_t^k, V_t^k, W, loc_t^k, X_t^k]_{|task_t^k|*5}. \tag{18}$$

Each row represents the state of a task in the system, with a total of $|task_t^k|$ rows.

Action: In time slot t , actions for state s_t are represented by an array that includes migration decisions and resource allocation strategy, i.e.,

$$a_t = [\Psi_t^k, p_{\Psi_t^k}]_{|task_t^k|*2}. \tag{19}$$

Each row indicates the migration target server $s_{\Psi_t^k}$ for the task $task_t^k$, and the computing resources assigned to $task_t^k$ on that server.

Reward: Reinforcement learning methods are typically employed to maximize the long-term reward of the system. By converting the objective of minimizing the average delay of all tasks into minimizing their cumulative response times, we define the reward for an action a_t as the negative value of the delay. Additionally, we assign specific negative rewards for tasks that fail to complete their upload to the cloud within the given time constraints.

$$r_t = \begin{cases} -\sum_{k=1}^{|task_t|} (Z_t^k T_t^k), & \text{if } Rst_t^k \geq 0 \text{ and } V_t^k \leq 0; \\ (-10^3) * \sum_{k=1}^{|task_t|} (Z_t^k T_t^k), & \text{if } Rst_t^k \leq 0 \text{ and } V_t^k > 0. \end{cases} \quad (20)$$

4.2. Dynamic Migration and Resource Allocation Algorithm Based on A2C

Reinforcement learning is a robust approach for addressing dynamic programming challenges. Recent advancements in this field involve the integration of deep neural networks to represent both policy and value functions effectively. In this paper, we utilize the A2C method to optimize our objective. As illustrated in Figure 3, A2C not only refines the policy to select the optimal action for each state but also develops a value function that supports policy optimization.

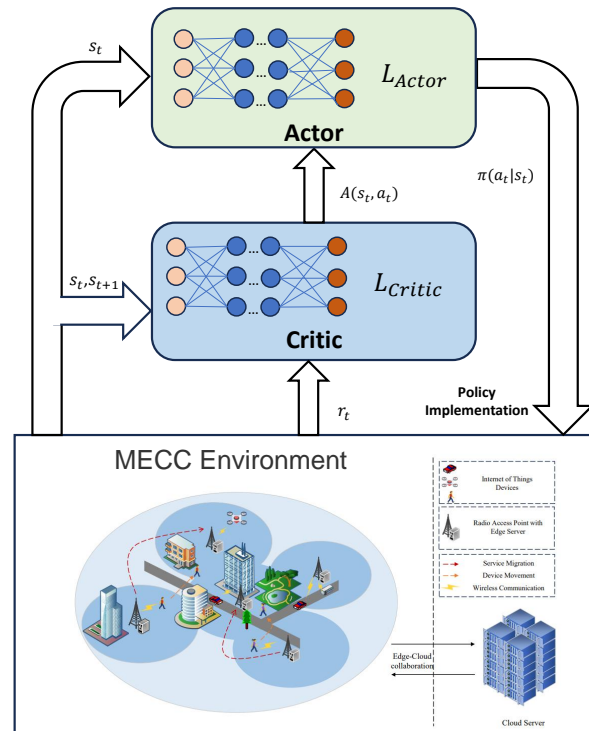


Figure 3. Training of A2C-based dynamic migration and resource allocation algorithm.

In DRL, for the given state s_t the expected reward obtained by selecting action a_t according to policy π is defined as the action value $Q_\pi(s_t, a_t)$, which is defined as follows:

$$Q_\pi(s_t, a_t) = E_{\{S, \pi\}} [R_t | s_t, a_t], \quad (21)$$

where R_t represents the expected sum of reward under the strategy π . R_t can be expressed as

$$R_t = \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t), \quad (22)$$

where γ^t represents the discount factor used to discount future rewards. According to Bellman's equation, the action value $Q_\pi(s_t, a_t)$ can be re-expressed as

$$Q_\pi(s_t, a_t) = r(s_t, a_t) + \gamma E_{S \sim S} [V_\pi(s_{t+1})], \quad (23)$$

and $V_\pi(s_{t+1})$ is the expected reward obtained by following policy π , called the state-value function, i.e.,

$$V_\pi(s_{t+1}) = E_{a_{t+1} \sim \pi} [Q_\pi(s_{t+1}, a_{t+1})]. \quad (24)$$

In the A2C framework, there are two primary components: the actor and the critic. The actor is responsible for maintaining a policy π , guiding action selection, and interacting with the environment. The critic, on the other hand, learns the state-value function based on rewards derived from the interactions between the actor and the environment, and aids the actor in policy updates.

The actor employs a neural network to model the policy function, generating action probabilities from the observed state. Its objective is to identify the optimal policy that maximizes the expected reward in the environment. We can define the objective function as

$$J(\pi) = E_{s_0} [V_\pi(s_0)]. \quad (25)$$

We use θ_a to denote the parameters of the actor's network and θ_c to denote the parameters of the critic's network. We can rewrite the function $J(\pi)$ as

$$J(\pi_{\theta_a}) = E[A(s_t, a_t) \cdot \log \pi_{\theta_a}(a_t | s_t)], \quad (26)$$

where $A(s_t, a_t)$ is advantage function. It indicates whether the reward obtained by choosing action a_t is higher than the average reward in s_t . We define the advantage function as

$$A(s_t, a_t) = Q_{\theta_c}(s_t, a_t) - V_{\theta_c}(s_t) \approx r(s_t, a_t) + \gamma V_{\theta_c}(s_{t+1}) - V_{\theta_c}(s_t). \quad (27)$$

Then, the loss function of the actor's network is

$$L_{Actor} = - \left[\log \pi_{\theta_a}(a_t | s_t) A(s_t, a_t) \right]. \quad (28)$$

Hence, we have

$$\theta_a \leftarrow \theta_a + l_a (A(s_t, a_t) \frac{\partial \log \pi_{\theta_a}(a_t | s_t)}{\partial \theta_a}), \quad (29)$$

where l_a is the learning rate of the actor network.

As with the actor network, we use the critic network to estimate the state-value function $V_{\theta_c}(s_t)$, and its loss function can be expressed as

$$L_{Critic} = (r + \gamma V_{\theta_c}(s_{t+1}) - V_{\theta_c}(s_t))^2. \quad (30)$$

Similarly, we have

$$\theta_c \leftarrow \theta_c + l_c (A(s_t, a_t) \frac{\partial V_{\theta_c}(s_t)}{\partial \theta_c}), \quad (31)$$

where l_c is the learning rate of the critic network.

Algorithm 1 outlines the A2C-based approach for migrating services and allocating resources described in this paper. Initially, we initialize the algorithm parameters, as well as the actor and critic networks, and set up the MECC system based on input values (lines 1–4). At the beginning of each episode, the system's state is reset and transmitted to the agent (lines 6–8). The agent then interacts with the environment: it inputs the observed state into the actor network, which returns the corresponding action distribution (line 9). The agent samples an action from this distribution (line 10), executes it, and receives the resultant reward and the next state (lines 11–12). This process repeats until the predefined update frequency is met. When it is time to update the networks, the observed states at time step $t + 1$ are fed into the critic network to obtain the average reward for that state (line 16). We then backtrack to time step $t - t_g + 1$, compute the advantage function using (27), and update the parameters of the actor and critic networks according to (29) and (31) (lines 18–21). The interaction with the environment continues until the termination of the algorithm. Ultimately, the agent consistently produces actions that yield favorable

rewards, demonstrating that the A2C-based approach has effectively optimized migration and resource allocation strategies in the MECC system. This system is now capable of executing pre-trained policies for efficient management.

Algorithm 1: Training of A2C-based dynamic migration and resource allocation algorithm.

Input : The number of ESs m , the number of users n
Output: Trained policy with parameter θ_a

- 1 Randomly set the initial weights for both the Actor and Critic networks;
- 2 Initialize the MECC system;
- 3 Set global counter i and step counter $t = 1$;
- 4 Set t_g, γ, EP_num, T , learning rate l_a, l_c ;
- 5 **for** episode $i \leftarrow 1$ **to** EP_num **do**
- 6 Reset the system state ;
- 7 **for** $t \leftarrow 1$ **to** T **do**
- 8 $s_t \leftarrow$ Observed system state;
- 9 The Actor network takes s_t as input and outputs the policy distribution of action $\pi(a_t|s_t)$;
- 10 Agent samples the action a_t according to $\pi(a_t|s_t)$;
- 11 Execute the action a_t ;
- 12 Obtain the reward r_t , get new state s_{t+1} ;
- 13 **if** $t \% t_g == 0$ **then**
- 14 // t_g is the frequency of update neuronal network
- 14 $R \leftarrow V_{\theta_c}(s_{t+1})$;
- 15 **for** $q \leftarrow t$ **to** $t - t_g + 1$ **do**
- 16 $R \leftarrow r_q + \gamma R$;
- 17 $A(s_q, a_q) \leftarrow R - V_{\theta_c}(s_q)$;
- 18 Calculate the Loss of Actor by (28);
- 19 Update θ_a of the Actor network according to (29);
- 20 Calculate the Loss of Critic by (30);
- 21 Update θ_c of the Critic network according to (31);
- 22 **end**
- 23 **end**
- 24 Change environment's state by s_{t+1} ;
- 25 **end**
- 26 **end**

5. Performance Evaluation

In this section, we conduct a series of simulations to evaluate the performance of the A2C algorithm against four alternative schemes across various environments. The results indicate that the algorithm proposed in this paper outperforms the comparative approaches in terms of efficiency.

5.1. Simulation Settings

In this simulation, we configure a randomly generated MECC system where the computing power and link bandwidth of each ES are randomly determined within specified parameter ranges. The system supports five types of applications, with parameters for each also randomly generated. The distance from the user to the RAN nodes is fixed at 10 m. Users can randomly select from one of the five application types to generate tasks for offloading. The wireless communication bandwidth between the user and the RANs is set to $B = 10$ MHz, the transmission power to $p_t^j = 10$ dBm, the channel power at the reference distance of 1 m to -50 dB, and the Gaussian noise power spectral density is established at $N_0 = -170$ dBm/Hz.

In our simulations, all deep neural networks (DNNs) are structured as four-layer fully connected networks, consisting of an input layer, two hidden layers, and an output layer. Each hidden layer in both the actor and critic networks contains 256 neurons. The learning rate for the networks is set at 0.001. As the objective is to minimize the average response delay across all tasks over a prolonged period, we set the discount factor for reinforcement learning to 0.9. Additional details on the experimental simulation parameters are provided in Table 2.

Table 2. Experiment parameters.

Parameters	Value
Bandwidth between ESs and cloud	[20, 100] Mbps
Total computing capacity in ES	[1, 3] GHz
Suspension processing intensity	[1.0, 2.0] cycles/bit
State context size	[100, 200] KByte
Bandwidth between ESs	[100, 200] Mbps
The ratio of the result data size to initial size	[0.001, 0.005]
Propagation delay from ESs to the cloud	400 ms
Computational intensity requirements	[20, 40] cycles/bit
Average task data size for W	[2.0, 3.0] MByte
The bandwidth between user and RANs	10 MHz
Transmission power	10 dBm
Noise power spectrum density	−170 dBm/Hz

5.2. Comparison Experiments

To evaluate our solution in a dynamic migration and resource allocation environment, we benchmarked it against four distinct schemes.

- **Follow-Avg scheme:** This scheme targets the user’s current location for migration if a task remains incomplete and there is residual time. It then allocates computing resources equally among all tasks on the same server.
- **PSO scheme:** In this scheme, migration targets and resource allocation decisions are treated as particles within a particle swarm optimization (PSO) algorithm, using average response delay as the fitness function. Decisions are made for each time slot state.
- **PPO scheme:** Proximal policy optimization (PPO) is employed, a method from online reinforcement learning within the DRL spectrum, to determine service migration and resource allocation.
- **DDPG:** Deep deterministic policy gradient (DDPG) utilizes the actor–critic framework of DRL to derive migration and resource allocation strategies.

5.3. Simulation Results

In all simulations, the parameters for the ESs were randomly generated within the ranges specified in Table 2. At each simulation step, new values were randomly selected from these ranges to update the system state. To assess performance, we calculated the average response delay and the average service failure rate across 10,000 episodes. Each episode consisted of 100 steps, during which we recorded the average service delay, the number of tasks generated by users, and the number of service timeouts.

The impact of the number of ESs: In this section, we investigate the effects of varying the number of ESs on the average service response delay and task failure rate. We fixed the number of users at 10, the duration of each time slot at 0.8 s, and the size of tasks generated by applications between 2 and 3 MB. Figure 4 shows the reduction in average response delay as the number of ESs increases from 4 to 8, incrementing by one each time. As the number of ESs grows, more computational resources become available, facilitating better task resource allocation and reducing service response times. Although the Follow-Avg, PPO, and DDPG methods exhibit minor differences in response delays across various

ES counts, they do not match the efficiency of the PSO and A2C methods. Notably, the A2C-based scheme consistently outperforms the PSO scheme by reducing the average response delay by 0.15 s across all evaluated numbers of ESs.

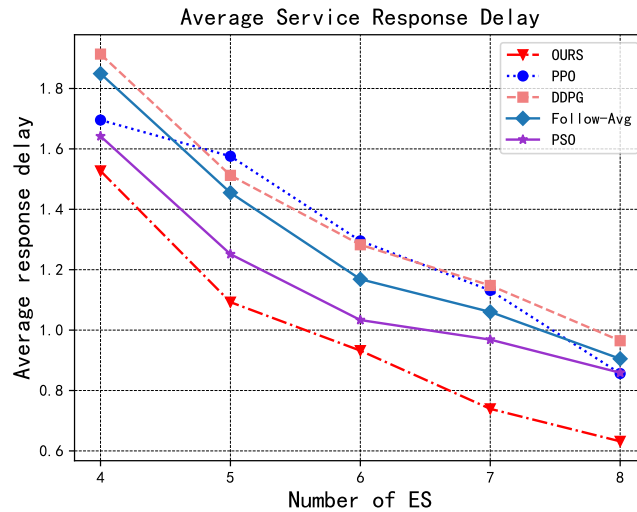


Figure 4. The impact of the number of ESs on average response delay.

Figure 5 demonstrates that our A2C-based approach achieves the lowest task failure rate, outperforming PSO, while PPO and DDPG record the highest failure rates. Under conditions of abundant resources, other methods experience failure rates as high as 20%, whereas our approach maintains a rate below 10%. Analyzing both Figures 4 and 5, with six or more ESs present, the Follow-Avg scheme shows a lower failure rate than PSO but suffers from longer response delays. This difference stems from the system’s dual focus on migration and resource allocation decisions. While abundant resources can mitigate response delays through efficient migration, leading to reduced failure rates, the generic resource allocation strategy of the Follow-Avg scheme, which does not account for task-specific requirements, results in prolonged average response delays. We conclude that the A2C-based algorithm excels over the other four schemes, offering superior service migration and resource allocation policies that effectively minimize both the average service response delay and failure rate, particularly in unpredictable environments.

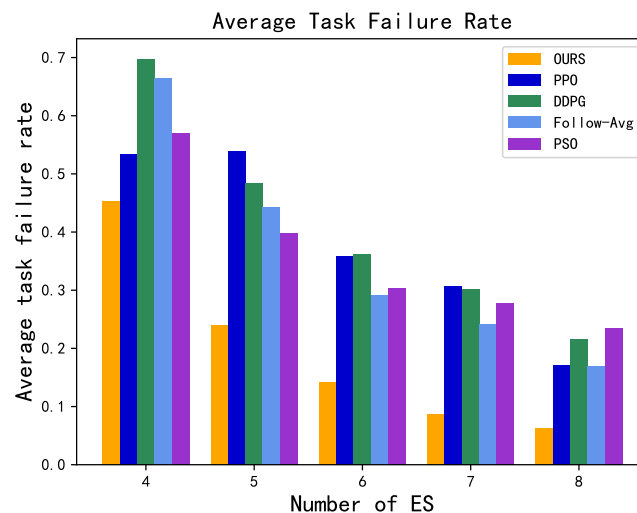


Figure 5. The impact of the number of ESs on failure rate.

Impact of time constraints: In our system, the time constraints for applications are defined by the number of time slots, and adjustments to the duration of these slots influence

the time constraints for each application. For this analysis we fixed the number of ESs at five while maintaining the number of users and task sizes consistent with previous comparisons. Figure 6 illustrates the average service response delays for varying time slot durations, from 0.6 s to 1.0 s, with an increment of 0.1 s. As shown in Figure 6, excluding PSO and our method, the differences in average response delays among the other three methods are minimal.

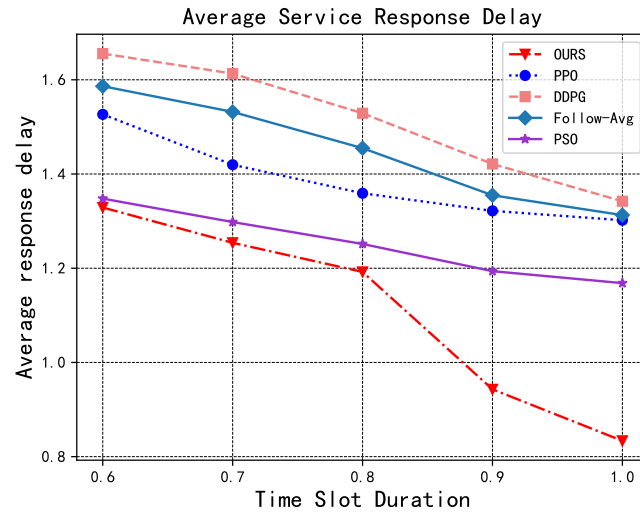


Figure 6. The impact of the time constraint on average response delay.

Our method consistently achieves the lowest average service response delay. As the time constraints are relaxed with a constant computational workload, the average service response delay decreases. Shorter time slot durations impose stricter constraints on each task, leading to an increased number of tasks that fail to complete within the designated time. Consequently, these tasks are offloaded to the cloud for continuation, introducing significant propagation delays during transmission and increasing the average service response delay. According to Figure 7, as time constraints become less stringent the task failure rate diminishes. Notably, under tighter time constraints, simple follow-migration methods exhibit higher failure rates, whereas our method consistently outperforms others. With more lenient time constraints, our method maintains a task failure rate below 10%. These observations highlight the importance of intelligently performing dynamic migration and resource allocation according to the current state in dynamic environments.

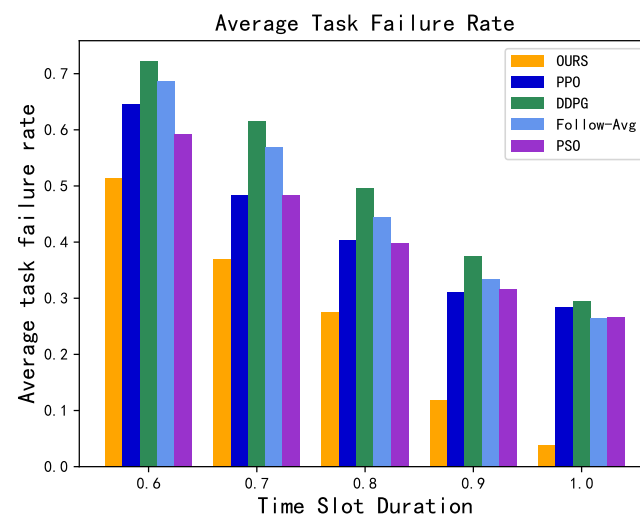


Figure 7. The impact of the time constraint on failure rate.

Impact of the number of users: The number of users in the system directly impacts the quantity of tasks, influencing the QoS for users. In this comparison, the number of ESs, task sizes, and time slot durations are consistent with previous experiments. We increment the number of users from 8 to 12, adjusting by one each time. As demonstrated in Figure 8, the average service response delay for all methods increases as the number of users rises due to more intense competition for resources among tasks. Notably, when the user count reaches 11, the average response delays for our method and the PSO approach converge. However, our approach consistently maintains a lower average response delay across various scenarios compared to other benchmarks. Furthermore, as shown in Figure 9, the PSO method requires significantly more time to make migration and resource allocation decisions than other methods, which may be impractical in real-world settings. Our method, however, delivers optimal results more efficiently. Figure 10 illustrates that task failure rates increase with the number of users. Despite this, our method outperforms others even under intense resource competition, achieving failure rates approximately 10% lower than PSO and 30% lower than the Follow-Avg method.

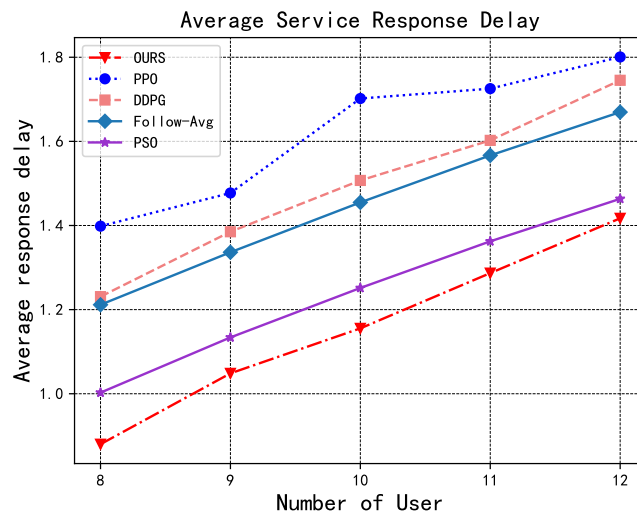


Figure 8. The impact of the number of users on average response delay.

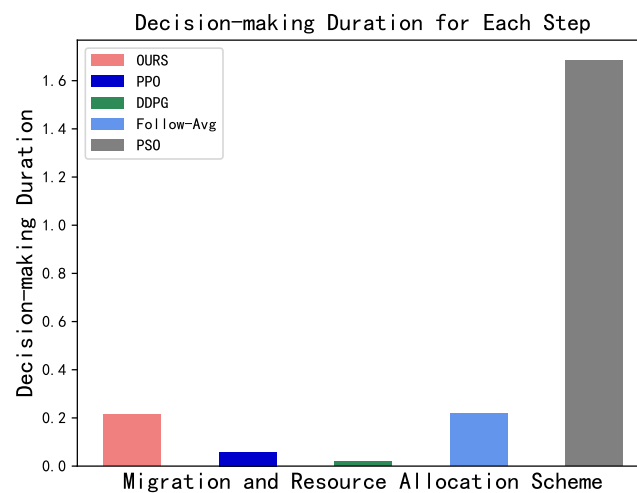


Figure 9. Decision-making duration for each step.

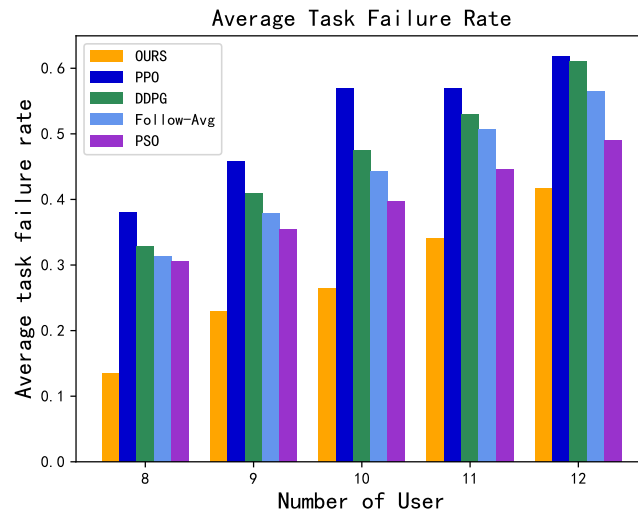


Figure 10. The impact of the number of users on failure rate.

Impact of task data size: In this section, we compare the impact of different task sizes on the results by varying the task sizes generated by the applications while keeping the number of ESs, time slot duration, and number of users consistent with the previous experiments. Regarding the task sizes generated by the applications, we set the application data size range from 2 ± 0.5 MB to 3 ± 0.5 MB, with an increment of 0.25 MB each time. Figure 11 demonstrates the impact of different application data sizes on the decisions made by various methods, with average response delay as the metric. It can be observed that as the task data size increases, the average response delay gradually rises. One reason for this is that the task’s data size affects migration delay, thereby reducing the time available for computation. Additionally, although the computational resources in the system remain unchanged, an increase in data size also contributes to an increase in average response delay. Figure 12 illustrates the variation in task failure rates under different data sizes. We observe that as the data size increases, the increase in failure rate becomes more pronounced for the Follow-Avg scheme. In the scenario with the highest data size, the failure rate of the Follow-Avg scheme exceeds that of the PPO and PSO schemes, while our approach ensures the lowest failure rate in all scenarios. Overall, our method achieves lower response delay while maintaining a low failure rate.

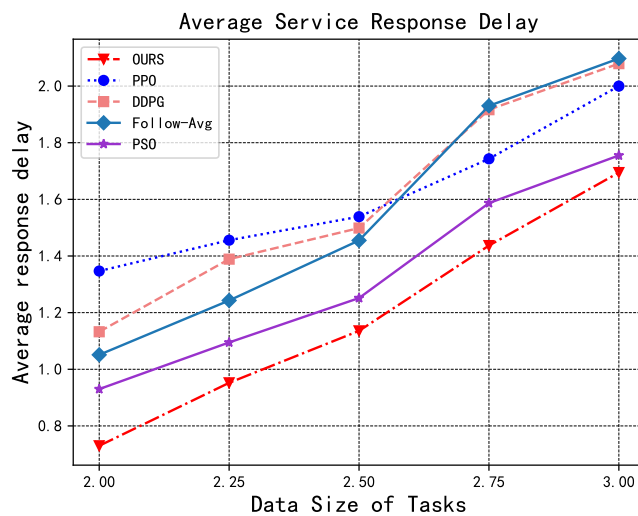


Figure 11. The impact of data size on average response delay.

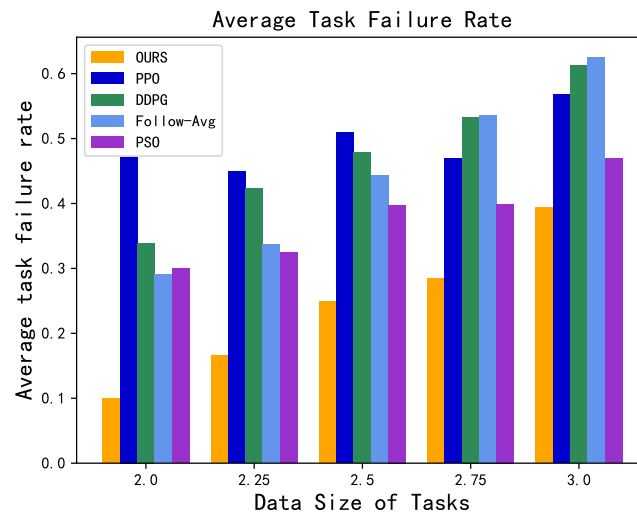


Figure 12. The impact of data size on average failure rate.

Impact of network scale expansion: To evaluate the effectiveness of the algorithm in a real-world scenario, we increased the number of users to 40 and the number of ESs to 20. As shown in Figure 13a, our method achieved the best performance in terms of average response delay, reducing it by 0.1 s compared to other strategies. Additionally, Figure 13b shows that the average failure rate was reduced by 10% compared to other methods. Overall, in a large-scale network environment, our method outperforms other approaches in dynamic migration and resource allocation strategies, maintaining lower average response delays and failure rates.

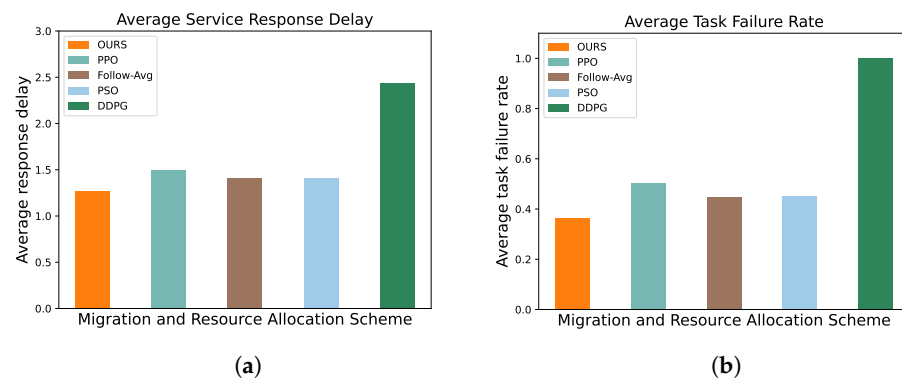


Figure 13. The impact of network scale expansion in an environment with 40 users and 20 ESs. (a) Average response delay. (b) Average failure rate.

5.4. Evaluation of Algorithm Overhead

In our simulation, we measured the overhead of the algorithm under different network scales, including memory usage, the number of iterations, and the training time per iteration. In the experiments, we used a 13th Gen Intel® Core™ i9-13900K CPU without utilizing a GPU. For network scales involving 10 and 40 users, the CPU usage for a single training did not exceed 10%. As shown in Table 3, the differences in the number of training iterations and the time required for each iteration across different network scales were minimal. However, when the number of users increased to 40, memory usage rose by approximately 30 MB compared to when there were 10 users. This indicates that as the problem scale increases, the overhead of the algorithm increases only slightly. We also observed that the duration of each iteration increased from 0.24 s to 0.938 s, due to the expansion of the neural network size caused by the increase in users and edge servers, which in turn increased the computational load. To address this growth, GPU acceleration could be considered in practical applications to effectively reduce the training time.

Table 3. Evaluation of algorithm overhead.

Network Scale (Number of Users)	Memory Usage (MB)	Number of Iterations	Training Duration per Iteration (s)
10	282.5	40,000	0.240
40	312.8	40,000	0.938

6. Conclusions and Future Work

In this paper, we investigate the dynamic migration and resource allocation issues in MECC systems. We emphasize the importance of service migration in networks characterized by dynamic features. Our investigation covers both the migration process and its performance, and we conduct a modeling analysis of computational performance post-migration. To tackle the challenges posed by dynamic computational demands and user mobility, we propose a method based on the Advantage Actor–Critic framework. This method determines migration and resource allocation operations for each time slot, based on observed states, aiming to minimize the average task response delay. The simulation results demonstrate that our A2C-based approach consistently reduces the average task response delay across various scenarios and ensures the lowest task failure rate compared to benchmark methods.

However, several aspects require further research. While our primary focus has been on the impact of migration on average response delay, the migration process also leads to additional network effects, including migration costs. Future research should explore strategies that simultaneously reduce migration costs and average response times. Additionally, our study mainly examines the impact of migration and computing resource allocation decisions. However, in real-world scenarios, the task offloading strategy significantly influences these migration and resource allocation strategies due to varying communication conditions associated with user mobility. This complex interplay between the offloading strategy and system performance in realistic settings demands more detailed investigation to optimize both cost and efficiency effectively. Another important issue to consider is that the centralized decision-making process may pose a risk of user data leakage. Therefore, finding ways to ensure data security while minimizing the impact of privacy protection mechanisms on system decisions has become a critical challenge.

Author Contributions: Conceptualization, Z.H.; Methodology, L.L. and Z.L.; Software, L.L.; Validation, Z.H. and L.L.; Formal analysis, J.Q.; Investigation, Z.H., L.L. and K.L.; Resources, K.L.; Data curation, Y.D.; Writing—original draft preparation, Z.H. and L.L.; Writing—review and editing, Z.H. and J.Q.; Visualization, L.L. and Y.D.; Supervision, K.L.; Project administration, K.L.; Funding acquisition, Z.H. and K.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the National Natural Science Foundation of China under Grant No. 62362068, in part by the Applied Basic Research Foundation of Yunnan Province under Grant Nos. 202201AT070156 and 202301AT070194, in part by the Open Foundation of Yunnan Key Laboratory of Software Engineering under Grant No. 2023SE208.

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

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

References

- Chen, W.; Wang, D.; Li, K. Multi-User Multi-Task Computation Offloading in Green Mobile Edge Cloud Computing. *IEEE Trans. Serv. Comput.* **2019**, *12*, 726–738. [[CrossRef](#)]
- Chen, Z.; Zhou, Z.; Chen, C. Code Caching-Assisted Computation Offloading and Resource Allocation for Multi-User Mobile Edge Computing. *IEEE Trans. Netw. Serv. Manag.* **2021**, *18*, 4517–4530. [[CrossRef](#)]
- Alkhalailah, M.; Calheiros, R.N.; Nguyen, Q.V.; Javadi, B. Data-intensive application scheduling on Mobile Edge Cloud Computing. *J. Netw. Comput. Appl.* **2020**, *167*, 102735. [[CrossRef](#)]
- Kong, X.; Wu, Y.; Wang, H.; Xia, F. Edge Computing for Internet of Everything: A Survey. *IEEE Internet Things J.* **2022**, *9*, 23472–23485. [[CrossRef](#)]

5. Siriwardhana, Y.; Porambage, P.; Liyanage, M.; Ylianttila, M. A Survey on Mobile Augmented Reality With 5G Mobile Edge Computing: Architectures, Applications, and Technical Aspects. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 1160–1192. [[CrossRef](#)]
6. Liu, S.; Yu, Y.; Lian, X.; Feng, Y.; She, C.; Yeoh, P.L.; Guo, L.; Vucetic, B.; Li, Y. Dependent Task Scheduling and Offloading for Minimizing Deadline Violation Ratio in Mobile Edge Computing Networks. *IEEE J. Sel. Areas Commun.* **2023**, *41*, 538–554. [[CrossRef](#)]
7. Liu, S.; Liu, L.; Tang, J.; Yu, B.; Wang, Y.; Shi, W. Edge Computing for Autonomous Driving: Opportunities and Challenges. *Proc. IEEE* **2019**, *107*, 1697–1716. [[CrossRef](#)]
8. Shakarami, A.; Ghobaei-Arani, M.; Shahidinejad, A. A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective. *Comput. Netw.* **2020**, *182*, 107496. [[CrossRef](#)]
9. Duan, S.; Wang, D.; Ren, J.; Lyu, F.; Zhang, Y.; Wu, H.; Shen, X. Distributed Artificial Intelligence Empowered by End-Edge-Cloud Computing: A Survey. *IEEE Commun. Surv. Tutor.* **2023**, *25*, 591–624. [[CrossRef](#)]
10. Chen, X.; Bi, Y.; Chen, X.; Zhao, H.; Cheng, N.; Li, F.; Cheng, W. Dynamic Service Migration and Request Routing for Microservice in Multicell Mobile-Edge Computing. *IEEE Internet Things J.* **2022**, *9*, 13126–13143. [[CrossRef](#)]
11. Li, X.; Chen, S.; Zhou, Y.; Chen, J.; Feng, G. Intelligent Service Migration Based on Hidden State Inference for Mobile Edge Computing. *IEEE Trans. Cogn. Commun. Netw.* **2022**, *8*, 380–393. [[CrossRef](#)]
12. Wang, P.; Ouyang, T.; Liao, G.; Gong, J.; Yu, S.; Chen, X. Edge intelligence in motion: Mobility-aware dynamic DNN inference service migration with downtime in mobile edge computing. *J. Syst. Archit.* **2022**, *130*, 102664. [[CrossRef](#)]
13. Liu, C.F.; Bennis, M.; Debbah, M.; Poor, H.V. Dynamic Task Offloading and Resource Allocation for Ultra-Reliable Low-Latency Edge Computing. *IEEE Trans. Commun.* **2019**, *67*, 4132–4150. [[CrossRef](#)]
14. Wang, X.; Ye, J.; Lui, J.C. Decentralized Task Offloading in Edge Computing: A Multi-User Multi-Armed Bandit Approach. In Proceedings of the IEEE INFOCOM 2022—IEEE Conference on Computer Communications, London, UK, 2–5 May 2022; pp. 1199–1208. [[CrossRef](#)]
15. Yang, J.; Yuan, Q.; Chen, S.; He, H.; Jiang, X.; Tan, X. Cooperative Task Offloading for Mobile Edge Computing Based on Multi-Agent Deep Reinforcement Learning. *IEEE Trans. Netw. Serv. Manag.* **2023**, *20*, 3205–3219. [[CrossRef](#)]
16. Liang, J.; Xing, H.; Wang, F.; Lau, V.K.N. Joint Task Offloading and Cache Placement for Energy-Efficient Mobile Edge Computing Systems. *IEEE Wirel. Commun. Lett.* **2023**, *12*, 694–698. [[CrossRef](#)]
17. 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**, 1403–1415. [[CrossRef](#)]
18. Zhu, Y.; Mao, B.; Kato, N. A Dynamic Task Scheduling Strategy for Multi-Access Edge Computing in IRS-Aided Vehicular Networks. *IEEE Trans. Emerg. Top. Comput.* **2022**, *10*, 1761–1771. [[CrossRef](#)]
19. Liu, L.; Zhao, M.; Yu, M.; Jan, M.A.; Lan, D.; Taherkordi, A. Mobility-Aware Multi-Hop Task Offloading for Autonomous Driving in Vehicular Edge Computing and Networks. *IEEE Trans. Intell. Transp. Syst.* **2023**, *24*, 2169–2182. [[CrossRef](#)]
20. Dang, X.; Su, L.; Hao, Z.; Shang, X. Dynamic Offloading Method for Mobile Edge Computing of Internet of Vehicles Based on Multi-Vehicle Users and Multi-MEC Servers. *Electronics* **2022**, *11*, 2326. [[CrossRef](#)]
21. Liao, L.; Lai, Y.; Yang, F.; Zeng, W. Online computation offloading with double reinforcement learning algorithm in mobile edge computing. *J. Parallel Distrib. Comput.* **2023**, *171*, 28–39. [[CrossRef](#)]
22. Huang, J.; Wan, J.; Lv, B.; Ye, Q.; Chen, Y. Joint Computation Offloading and Resource Allocation for Edge-Cloud Collaboration in Internet of Vehicles via Deep Reinforcement Learning. *IEEE Syst. J.* **2023**, *17*, 2500–2511. [[CrossRef](#)]
23. Xu, X.; Liu, K.; Dai, P.; Jin, F.; Ren, H.; Zhan, C.; Guo, S. Joint task offloading and resource optimization in NOMA-based vehicular edge computing: A game-theoretic DRL approach. *J. Syst. Archit.* **2023**, *134*, 102780. [[CrossRef](#)]
24. Liu, L.; Feng, J.; Mu, X.; Pei, Q.; Lan, D.; Xiao, M. Asynchronous Deep Reinforcement Learning for Collaborative Task Computing and On-Demand Resource Allocation in Vehicular Edge Computing. *IEEE Trans. Intell. Transp. Syst.* **2023**, *24*, 15513–15526. [[CrossRef](#)]
25. Liang, Z.; Liu, Y.; Lok, T.M.; Huang, K. A Two-Timescale Approach to Mobility Management for Multicell Mobile Edge Computing. *IEEE Trans. Wirel. Commun.* **2022**, *21*, 10981–10995. [[CrossRef](#)]
26. Bozkaya, E. Digital twin-assisted and mobility-aware service migration in Mobile Edge Computing. *Comput. Netw.* **2023**, *231*, 109798. [[CrossRef](#)]
27. Peng, Y.; Tang, X.; Zhou, Y.; Li, J.; Qi, Y.; Liu, L.; Lin, H. Computing and Communication Cost-Aware Service Migration Enabled by Transfer Reinforcement Learning for Dynamic Vehicular Edge Computing Networks. *IEEE Trans. Mob. Comput.* **2024**, *23*, 257–269. [[CrossRef](#)]
28. Xu, M.; Zhou, Q.; Wu, H.; Lin, W.; Ye, K.; Xu, C. PDMA: Probabilistic service migration approach for delay-aware and mobility-aware mobile edge computing. *Softw. Pract. Exp.* **2022**, *52*, 394–414. [[CrossRef](#)]
29. Wang, H.; Li, Y.; Zhou, A.; Guo, Y.; Wang, S. Service migration in mobile edge computing: A deep reinforcement learning approach. *Int. J. Commun. Syst.* **2023**, *36*, e4413. [[CrossRef](#)]
30. Li, C.; Zhang, Y.; Gao, X.; Luo, Y. Energy-latency tradeoffs for edge caching and dynamic service migration based on DQN in mobile edge computing. *J. Parallel Distrib. Comput.* **2022**, *166*, 15–31. [[CrossRef](#)]
31. Chen, W.; Chen, Y.; Wu, J.; Tang, Z. A multi-user service migration scheme based on deep reinforcement learning and SDN in mobile edge computing. *Phys. Commun.* **2021**, *47*, 101397. [[CrossRef](#)]

32. Chen, W.; Chen, Y.; Liu, J. Service migration for mobile edge computing based on partially observable Markov decision processes. *Comput. Electr. Eng.* **2023**, *106*, 108552. [[CrossRef](#)]
33. Liang, Z.; Liu, Y.; Lok, T.M.; Huang, K. Multi-Cell Mobile Edge Computing: Joint Service Migration and Resource Allocation. *IEEE Trans. Wirel. Commun.* **2021**, *20*, 5898–5912. [[CrossRef](#)]
34. Liu, F.; Yu, H.; Huang, J.; Taleb, T. Joint Service Migration and Resource Allocation in Edge IoT System Based on Deep Reinforcement Learning. *IEEE Internet Things J.* **2024**, *11*, 11341–11352. [[CrossRef](#)]
35. Sutton, R.; Barto, A. *Reinforcement Learning, Second Edition: An Introduction*; Adaptive Computation and Machine Learning Series; MIT Press: Cambridge, MA, USA 2018.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.