scientific reports

Check for updates

OPEN

Latency and energy-aware adaptive service migration in mobile edge computing

Lina Li[1], Junan Lv[1], Shuxin Wang[1], Chuncheng Hu[1], Guodong Sun[2] & Nianfeng Li[1]✉

Mobile Edge Computing (MEC) is a technology that provides computing services to mobile users by deploying edge servers at base stations, and it has become an important tool for enhancing user service experience. Due to limited resources and user mobility, service migration among edge servers becomes an effective way to maintain user service latency. However, frequent service migrations and resource competition among users result in additional energy consumption, significantly increasing the operational costs of service providers. In complex dynamic and resource-competitive scenarios, considering migration energy consumption as an independent core optimization objective comparable to service latency and making adaptive online migration decisions pose significant challenges and remain underexplored. In this paper, we propose a systematic new approach to address the service migration challenges in MEC from the perspectives of theoretical modeling, algorithm design, and mechanism innovation, namely a delay- and energy-aware adaptive service migration method. Firstly, we model service migration as a constrained (resource competition) bi-objective optimization problem (minimizing service latency and migration energy consumption), which can be formulated as a mixed-integer nonlinear programming problem. Then, given the NP-hard nature of this problem, network dynamics, differences in user mobility patterns, and the real-time requirements of migration decisions, we design a migration model based on Deep Q-Network (DQN), referred to as NPER-D3QN. This model integrates elements of D3QN, Noisy Net, and prioritized experience replay, enabling the generation of adaptive migration strategies with low latency and low energy consumption according to real-time network states, user mobility patterns, and server loads. Finally, we conduct simulation experiments comparing our proposed service migration method with existing typical and advanced methods. The results demonstrate that our method exhibits superior performance in terms of latency and energy efficiency, showing certain advantages in dynamic adaptation and multi-objective collaborative optimization.

Keywords Mobile edge computing, Adaptive service migration, Deep Q networks, Latency awareness, Energy awareness

The explosive growth of mobile internet and Internet of Things (IoT) has intensified the demand for massive data generation and real-time processing, which urgency has led to the emergence of Mobile Edge Computing (MEC)[1]. In the MEC architecture, edge servers located at base stations provide proximity-based computing services to users, thereby enhancing user experience[2]. Due to limited coverage ranges and computational resources of base stations, service migration becomes necessary when users move out of the coverage area or resource contention occurs, aiming to reduce service latency[3]. However, frequent service migrations increase communication energy consumption of edge servers, thereby raising operational costs for service providers[4]. Meanwhile, service redeployment also incurs additional migration delays. Therefore, how to achieve adaptive online service migration decisions that jointly optimize service latency and system energy consumption under complex scenarios characterized by diverse user mobility patterns, dynamic network conditions, and resource contention remains a challenging and pressing issue requiring comprehensive solutions.

Researchers have advanced various solutions to address the issue of service migration in Mobile Edge Computing (MEC). The majority of research on service migration addresses it as a multi-objective optimization problem. Specifically, they focus on service delay as the primary objective. To determine the optimal migration strategy, various methods have been employed, including dual-layer graph neural network architecture[5], Lyapunov optimization algorithms[6], and deep reinforcement learning algorithms[7–10]. While the aforementioned

[1]School of Computer Science and Technology, Changchun University, Changchun 130022, China. [2]Department of Internet-of-Things, Beijing Forestry University, Beijing 100083, China. ✉email: linf@ccu.edu.cn

techniques consider the optimized migration cost, they neglect the system's energy consumption, particularly the energy used in migration. This oversight impacts the eco-friendly utilization of system resources. Meanwhile, some research are oriented towards user mobility and design migration methods based on the premise of user location prediction using Lyapunov optimization algorithms[11], game theory[12], and reinforcement learning techniques[13], etc. However, they do not consider the uncertainty of resources and loads and are insufficiently adaptive to dynamic scenarios. In addition, some works do not consider resource constraints and design optimal migration strategies based on Lyapunov optimization methods[14], deep reinforcement learning methods[15,16], etc., which are insufficiently scalable and adaptive in practical applications. In summary, existing studies have less joint optimization of delay and energy consumption, and do not comprehensively consider the dynamics of users, resources, and load, which affects the adaptivity of migration methods.

In the complex dynamic scenarios of MEC, taking into account the user service latency and system migration energy consumption, the design of adaptive migration methods faces two main challenges: how to construct a system model and a problem model that support the joint optimization of service latency and migration energy consumption, and how to design an adaptive migration model that supports the dynamics of the users, resources, and loads. Some researchers model service migration as a multi-objective optimization problem, employing Deep Reinforcement Learning (DRL) techniques to address adaptive service migration challenges, including DQN algorithms[17,18], Deep Recursive Actor-Critic Model[19], etc. However, these studies only consider user mobility without considering user competition for computing resources, which leads to service migration queuing phenomenon and increases service delay, thus decreasing user QoS. To ensure high QoS, some studies[20,21] optimize the migration by introducing resource-awareness and mobility prediction modules into the DRL model, which take into account both user mobility and resource competitiveness strategies. Although deep reinforcement learning performs well in terms of adaptivity, existing studies fail to effectively balance user QoS and migration cost, resulting in wasted system resources or excessive migration frequency. Therefore, the exploration of how to use deep reinforcement learning to design high-quality and low-cost adaptive migration strategies is still the focus of service migration research.

In this paper, our primary focus is on the energy cost of service migration (i.e., communication energy consumption) and studies the service migration problem that guarantees high service quality with low energy overhead under resource constraints and user mobility. Due to the diversity of user mobility patterns and the fluctuation of base station bandwidth, service migration needs to be adaptive to provide a low-cost and high-quality service experience. Consequently, we suggest a sophisticated, deep learning-driven approach for service migration, designed to deduce the optimal migration strategy through ongoing environmental interaction. The main contributions of this paper are as follows.

- We propose a delay and energy-aware adaptive service migration method to systematically address the service migration problem in MEC from the perspectives of theoretical modeling, algorithm design, and mechanism innovation. Firstly, we model the migration problem as a constrained bi-objective optimization problem, aiming to minimize both latency and energy consumption while satisfying constraints on computational and communication resources. This problem belongs to the category of nonlinear mixed integer programming problems, where traditional optimization methods fail to adapt to complex environments and cannot meet real-time requirements.
- To address this NP-hard problem, considering the dynamic changes in the network, variations in user mobility patterns, and the real-time nature of migration decisions, we propose a DQN-based service migration model called NPER-D3QN to generate adaptive migration strategies. We introduce double Q-learning and dueling network architectures to resolve the issue of overestimation of Q-values, thereby improving the accuracy of the model. Additionally, by employing prioritized experience replay and noisy networks, we enhance its training efficiency and accelerate convergence.
- We also conduct time complexity analysis and runtime analysis for NPER-D3QN. A series of simulation experiments were conducted to validate the effectiveness of the proposed method. The results demonstrate that compared to typical and advanced baseline methods, NPER-D3QN achieves faster convergence speed and significant improvements in terms of average latency, migration cost, and service rejection rate. Furthermore, we explore the stability and scalability of our approach under different scenarios.

The paper is structured as follows: "Related work" provides a review of recent research. "System model and problem formulation" presents the system model and problem formulation. "Adaptive service migration model based on DRL" provides a detailed description of the proposed method and its implementation. The performance evaluation results are presented in "Performance evaluation". The paper concludes with "Conclusion".

Related work

In mobile edge computing, 5G communication technologies have created a demand for timeliness and resource availability of emerging IoT applications. While user applications or services are deployed at the edge servers to meet specific latency and resource requirements[22], mobility of users requires involvement of multiple edge servers. This is essential to offer continuous on-demand services, thereby ensuring a seamless and superior user experience. In recent years, researchers have proposed numerous service migration schemes in response to the evolving service demands of users. These schemes prioritize low latency during service migration, taking into account changes in users' geographic locations. Additionally, they consider the computational and storage resources of edge servers.

User location-based migration techniques typically assume the accurate prediction of a user's movement trajectory. They utilize this assumption as a foundation to instigate service migration operations either proactively or in real-time, thereby ensuring the uninterrupted provision and availability of services. Anwar

et al.[23] developed a dynamic shortest path selection algorithm and a dynamic multipath search algorithm to address the service disruption and Quality of Service (QoS) degradation. These algorithms enable autonomous network boundary discovery and path optimization while maintaining critical node connectivity. Xu et al.[11] introduced a mobile-aware service migration method rooted in a probabilistic model, which adeptly minimizes both service latency and migration expenses. Chen et al.[24] proposed a dynamic transfer framework based on real-time trajectory prediction and multi-agent deep reinforcement learning to address the challenges of task transfer decision-making in vehicular metaverse for everything, which effectively reduces the execution latency of tasks and improves user experience. Zhao et al.[25] proposed a mobile aware proactive service migration method for MEC. By integrating recurrent neural network with geo-embedded Markov chain predictor through ensemble learning, it predicts users' next location and determines target edge server for service migration according to these predictions, thereby reducing migration time and end-to-end latency. Li et al.[12] introduced a dynamic service migration approach grounded in a non-cooperative game theory to mitigate the decline in service quality resulting from user mobility. Cao et al.[13] proposed a path optimization technique based on deep reinforcement learning, aiming to concurrently minimize communication cost and latency while satisfying the service requirements. These location-based migration schemes often overlook the dynamic nature of the network environment and the unpredictability of the system load. In real MEC environments, triggering migrations based only on location changes may lead to wasted resources or too frequent migrations, making it difficult to balance the quality of service and system efficiency.

Some approaches typically minimise end-to-end delay by optimising task allocation, link selection, and migration path planning to improve the performance of interactive applications and real-time services. Chen et al.[26] decomposed the mixed-integer nonlinear programming problem involving service migration and resource allocation into two subproblems. They employed a deep reinforcement learning approach based on actor-critic asynchronous updates to address service migration, thereby optimizing service quality in multi-edge IoT systems. Kang et al.[8] conceptualized the vehicle-to-base-station task migration dilemma as a partially observable Markov decision process. Consequently, they introduced a dynamic migration system, grounded in multi-intelligent body deep reinforcement learning. Li et al.[27] proposed an intelligent service migration method that leverages hidden state inference, which markedly enhances the quality of service and reduces latency. Shi et al.[18] proposed a two-time scale resource management framework for MEC by trade-off between service migration and task redirection to achieve seamless and cost-efficient MEC services. Chen et al.[10] integrated task assignment and service migration within IoT systems, modeling them as Markov decision-making processes, and using a training architecture anchored in a dual-delay deep deterministic policy gradient. Chen et al[28] proposed a data-driven mobile-aware service migration scheme in MEC to decrease system latency and enhance the efficiency of service migration. Tsourdinis et al[7] introduced a DRL-driven service migration mechanism to effectively address the communication delay problem caused by user mobility and external factors. Although such approaches have achieved some success in reducing service latency, they tend to ignore energy consumption constraints. In situations where a choice must be made between energy consumption and delay, an exclusive focus on delay can lead to increased system energy usage and decreased overall resource utilization. This limits the practical value of such applications in energy-sensitive environments.

Some studies focus on the logical design of the migration decision under the assumption of infinite resources to verify the effectiveness and feasibility of the scheme at the theoretical level. Peng et al[29] proposed a service migration mechanism grounded in migration reinforcement learning, which enhances the satisfaction of service migration by taking into both computational and communication cost facts. Wang et al[14] formulated the migration problem as a multi-objective optimization challenge, employing the Lyapunov optimization method to derive utility-optimal migration decisions to ensure service reliability and reduce energy consumption. Xu et al[5] designed a task migration strategy, utilizing the intensity Pareto evolutionary algorithm, to strike an optimal balance between energy consumption and time cost. Han et al[6] introduced a service migration optimization algorithm, leveraging a deep Q-learning network, designed to minimize the service access delay and energy consumption. Fan et al.[30] introduced a service migration scheme tailored for backhaul-less vehicular networking, which achieves backhaul network-independent service migration by taking into account task dependencies. Zheng et al[15] proposed a dynamic and energy-efficient service migration framework for 3-D networks, which leverages deep reinforcement learning to optimize migration decisions, aiming to reduce communication delays and energy consumption. Zhang et al[16] introduced a mobility-aware service migration mechanism, grounded in deep reinforcement learning, designed to mitigate service delay and network migration delay in multi-access edge computing environments. In real deployment environments, edge nodes have limited computation and resource competition becomes an unavoidable key issue. As a result, such migration schemes often face performance degradation or even unavailability in real environments, and lack practical scalability and adaptability.

In contrast to previous work, we propose an adaptive service migration algorithm that integrates the multi-user mobility and resource competition to achieve service migration with high service quality and low system energy consumption. Furthermore, our method is scalable and can adapt to the dynamic changes in the number of computing nodes and users in the MEC network. This ensures that optimal migration performance is consistently maintained, even within the intricate context of multi-user and high-density environments.

System model and problem formulation

System overview

The Mobile Edge Computing system in focus is depicted in Figure 1. This architecture features base stations, situated at various locations, interconnected to the core network via wired links thereby facilitating service migration. Each base station not only serves as a wireless access point but is also equipped with edge servers to execute users' computational tasks. Users are sporadically distributed within the system and possess the ability to move arbitrarily at any given moment. Typically, users establish connections to the nearest base station through

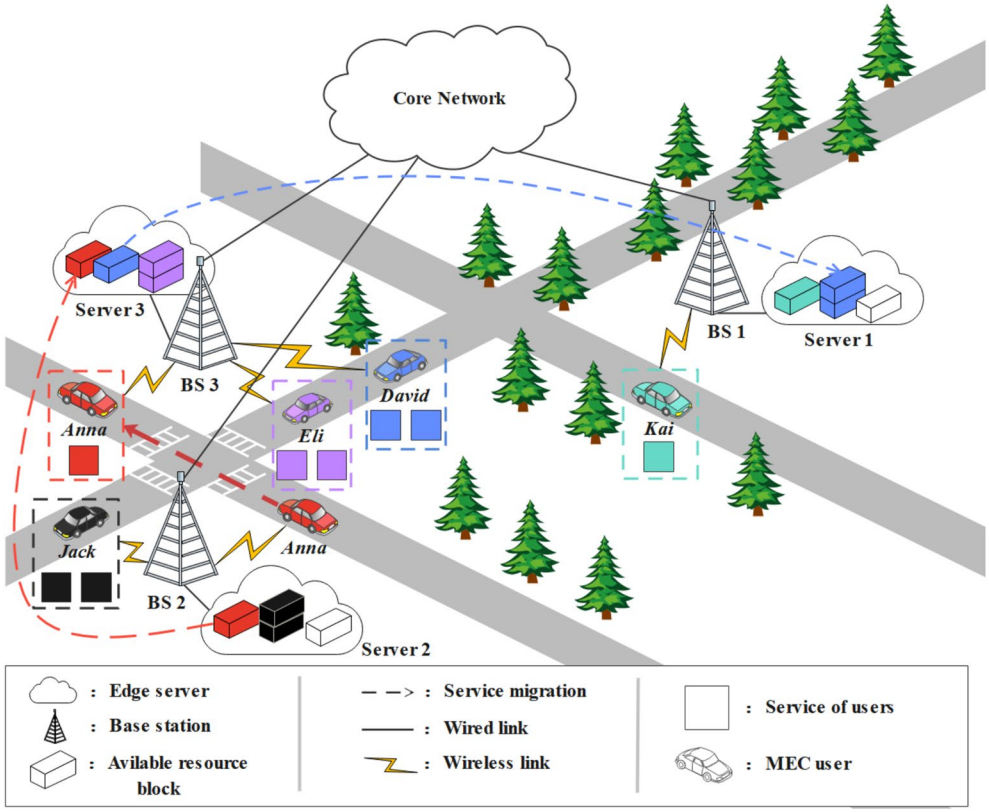


Fig. 1. Example of user service migration in MEC system.

wireless signals to maintain low communication latency[31]. When the locations of users change, the system is tasked with determining whether to migrate services. Although service migration can potentially decrease latency, it may also result in a corresponding increase in the overall energy consumption of the system[32].

In this system, each user has different computing tasks (also called services) and communicates with the nearest base station through wireless links to maintain low communication latency. The base stations (BS) serve as both access points and hosts for edge servers (ES) that handle users' computing tasks. Different base stations are interconnected via wired links to enable computing resource sharing[33]. Users are randomly distributed and can move to any possible location. When user locations change, their connected base stations may switch frequently, requiring the system to consider migrating user services to other nearby edge servers. Such migration typically reduces service latency but increases system energy consumption[34].

In the MEC architecture, each ES usually consists of multiple virtual machines with computing resources equally allocated among them[35]. User services are deployed on different virtual machines, which we abstract as resource blocks. Owing to the constrained computational resources of each Edge Server (ES), resource blocks can only be allocated to users in a sequential manner. This limitation inevitably results in service queuing. When an ES lacks sufficient computing resources, the MEC system may migrate services to other resource-sufficient ESs to reduce queuing delays.

Figure 1 illustrates a service migration example. At one moment, user Anna (red car) is within BS 2's coverage, with her service (red rectangle) allocated one resource block (red cube) on Server 2. Later, when Anna moves into BS 3's coverage, her service migrates from Server 2 to Server 3, allocating a new resource block (red cube). Meanwhile, Eli (purple car) and David (blue car) are also within BS 3's coverage. Since Server 3 only has 4 resource blocks - after allocating 2 to Eli (purple cubes) - only one remains (blue cube), insufficient for David's service needs (blue rectangle). Consequently, David's service migrates to the nearest resource-sufficient Server 1, allocating 2 resource blocks (blue cubes).

Service migration model

We examine a service migration model within the context of a cellular network. The model under consideration consists of a set of geographically distributed base stations L (indexed by l), interconnected via wired links for data transmission. A collection of edge servers S (indexed by s) are co-located with the base stations to provide computational resources for highly mobile users U (indexed by u), processing their offloaded service requests. The system functions through discrete time slots, segmenting time into T slots, each enduring ω seconds. Within a given time slot $t(t = 0, 1, 2, ..., T)$, a user set U produces a service set ϕ (indexed by φ). This set is subsequently implemented on edge servers via wireless links. For the convenience of the reader, we have gathered all relevant notations in Table 1.

Notation	Description
$x_{\varphi,t}(u,s)$	The connectivity between server s and user u in time slot t
K	Number of orthogonal resource blocks
W	Bandwidth of resource blocks
E_l	Number of users associated with BS l
P	The baseline transmit power of users
h_0	Channel power gain
$dist_{u,l}$	Distance from u to its serving BS
N_0	Noise Power
$data_{u,t}$	Offloaded task size of u at time slot t
$R_{u,l}$	The uplink wireless transmission rate of u
$f_{\varphi,u}$	The number of CPU cycles required by φ
F_s	The computing capacity of s
$n_{s,t}$	The number of virtual machines started by s
v_{φ}^{size}	Service instance size
B	Wired link bandwidth
ρ	Backhaul delay factor
$hop(s_1, s_2)$	The hop distance between MEC server s_1 and s_2
D_{φ}	Deadline for service φ
κ	The energy consumption of a CPU cycle
η	The migration cost consumption factor

Table 1. Definitions of notations.

We assume that the service deployment location of a user, denoted by φ, remains unchanged within one time slot. At time slot t, if service φ is processed on edge server s, we denote this as $x_{\varphi,t+1}(u,s)=1$. Otherwise $x_{\varphi,t+1}(u,s)=0$. At time slot $t+1$, if service φ remains on server s, then $x_{\varphi,t+1}(u,s)=x_{\varphi,t}(u,s)$. Otherwise, service migration occurs, then $x_{\varphi,t+1}(u,s)\neq x_{\varphi,t}(u,s)$.

Service delay model

Service latency is a term used to denote the time required for various operations including service communication, computation, migration, and backhaul[36]. The communication delay refers to the latency experienced by user u when communicating with base station l, which is primarily determined by the delay in task offloading. The computation delay represents the time necessary for the ES to process tasks. The migration delay is the latency caused by the migration of services through wired links. Finally, backhaul delay refers to the time needed for the edge server to send computational results back to the user, encompassing transmission, propagation, processing, and queuing. It should be noted that the data volume of these computational results is typically minimal[37] and, therefore, considered negligible in this study.

Communication delay. The offloading speed of user tasks to BS mainly depends on their wireless channel capacity. This study assumes that every base station utilizes an Orthogonal Frequency Division Multiple Access (OFDMA) model to manage communication resources[7]. Specifically, base station l provides K mutually orthogonal resource blocks, each with bandwidth W. The uploading speed of tasks from users to the BS is predominantly determined by the uplink. The rate of uplink transmission denoted as $R_{u,l}$, from user u to base station l can be quantified using the following formula.

$$R_{u,l}=W\cdot[\frac{K}{E_l}]\cdot log_2(1+SNR_u). \tag{1}$$

where E_l represents the quantity of users linked to BS l, and SNR_u denotes the received signal-to-noise ratio. The calculation of SNR_u is demonstrated in the following manner.

$$SNR_u=\frac{Ph_0dist_{u,l}^{\varpi(\zeta-1)}}{N_0}, \tag{2}$$

Here, P is the constant baseline transmit power of the user. h_0 is the channel power gain, $dist_{u,l}$ is the distance between user u and base station l. ϖ is the path loss exponent and ζ is the fractional order channel inversion power control component. Lastly, N_0 signifies the noise power. As a result, the communication delay, represented as $\tau_{\varphi,t}^{com}(u)$, experienced by user u when offloading the service in time slot t can be articulated through the equation given in[38].

$$\tau_{\varphi,t}^{com}(u) = \frac{data_{u,t}}{R_{u,l}}, \tag{3}$$

where $data_{u,t}$ is the size of user *u*' offloading service at time slot *t*.

Computation delay. The computational capacity of Edge Servers (ES) is notably constrained in comparison to cloud data centers. In this work, we measure ES computing capability by CPU cycles per second, where the computational capacity of ES *s* is denoted as F_s. During time slot *t*, when the total requested resources $F_s(t)$ from services deployed on s exceeds F_s, the system incurs additional queuing delay. Furthermore, each service is allocated a virtual machine on the ES, with computational resources equally distributed among these virtual machines. For each user, the available computing capacity of ES dynamically varies according to the number of services being processed concurrently in the same time slot. The computation delay $\tau_{\varphi,t}^{exe}(u)$ that a user's service experiences during time slot *t* depends on the computation capacity F_s of Edge Server *s* and the number of active virtual machines $n_{s,t}$ on Edge Server *s*[39]. The calculation formula is the following.

$$\tau_{\varphi,t}^{exe}(u) = \frac{f_{\varphi,u}}{F_s/n_{s,t}}, \tag{4}$$

where $f_{\varphi,u}$ denotes the quantity of CPU cycles required for processing service φ as submitted by user *u*. Additionally, $n_{s,t}$ quantifies the number of VMs initiated by ES *s* within time slot *t*.

Migration delay. In a wired link, the migration delay is predominantly influenced by three factors: the bandwidth *B* of the wired backhaul link between base stations, the size v_{φ}^{size} of data volume to be migrated by the service φ, and the hop distance, denoted as $hop(s_1, s_2)$, refers to the distance between the source server s_1 and the target server s_2. These parameters collectively dictate the time required for data transfer, impacting overall network efficiency. Specifically, it is calculated by the following formula.

$$\tau_{\varphi,t}^{mig}(u) = \begin{cases} 0, & x_{\varphi,t+1}(u,s) = x_{\varphi,t}(u,s), \\ \frac{v_{\varphi}^{size}}{B} + 2\rho hop(s_1,s_2), & x_{\varphi,t+1}(u,s) \neq x_{\varphi,t}(u,s) \end{cases}, \tag{5}$$

where ρ represents the return delay factor. It is important to mention that, in instances where the service has not been migrated, the migration delay linked to the service is deemed to be 0.

The variable $\tau_{\varphi,t}(u)$ characterizes the cumulative delay in processing the service φ for user *u* in the time slot *t*. More specifically, $\tau_{\varphi,t}(u)$ is calculated as sum of the three distinct delays: communication, computation delay, and migration delay.

$$\tau_{\varphi,t}(u) = \tau_{\varphi,t}^{com}(u) + \tau_{\varphi,t}^{exe}(u) + \tau_{\varphi,t}^{mig}(u), \tag{6}$$

Service energy model

When the system processes user services, data migration will generate corresponding energy consumption. In the migration process, the user service is transitioned from the source server to the target server via a wired connection. The primary factor influencing its energy consumption is the volume of data, denoted as v_{φ}^{size}, which the service φ requires for migration. This can be quantified using the following calculation.

$$C_{\varphi,t}^{mig}(u) = \begin{cases} 0, & x_{\varphi,t+1}(u,s) = x_{\varphi,t}(u,s), \\ \eta v_{\varphi}^{size}, & x_{\varphi,t+1}(u,s) \neq x_{\varphi,t}(u,s) \end{cases} \tag{7}$$

where η is the energy consumption factor associated with migration. In instances where the service remains unmigrated, the system's migration energy consumption is consequently zero.

In our work, $C_{\varphi,t}^{mig}(u)$ specifically models the migration-related communication energy consumption incurred when transferring service states to the wired backhaul through migration, aiming to quantify the direct energy cost of service migration. This formulation does not account for other components contributing to system-wide energy consumption, including idle/active power consumption of edge servers, cooling costs, and carbon footprint.

Problem formulation

This paper addresses the multi-user service migration problem in mobile edge computing. The objective is to minimize overall system latency and total migration cost by jointly determining an optimal service migration policy for each user, subject to their respective latency requirements and the resource constraints of the edge server. This problem can be formally defined as follows.

$$P1: min \sum_{t=1}^{T} \sum_{\varphi=1}^{\phi} \tau_{\varphi,t}(u), \forall u \in U, \tag{8}$$

$$P2: min \sum_{t=1}^{T} \sum_{\varphi=1}^{\phi} C_{\varphi,t}^{mig}(u), \forall u \in U, \tag{9}$$

$$s.t. \sum_{\varphi=1}^{\phi} \sum_{s=1}^{S} x_{\varphi,t+1}(u,s) = 1, \forall u, t, \tag{10}$$

$$\sum_{\varphi=1}^{\phi} \sum_{u=1}^{U} f_{\varphi,u} x_{\varphi,t}(u,s) \leq F_s, \forall t, s, \tag{11}$$

$$\tau_{\varphi,t}(u) \leq D_\varphi, \forall u, \varphi, t. \tag{12}$$

Constraint (10) indicates that user *u* can only upload service φ to the same edge server *s*. Constraint (11) ensures that the migration decision conforms to the computational resource constraints of the edge server. Constraint (12) ensures that every service is completed within its deadline, where D_φ represents the maximum acceptable delay for service φ.

Adaptive service migration model based on DRL

Overview of model design

In this section, we design a high-performance service migration model based on deep reinforcement learning to address the specific problem of joint latency-energy consumption optimization in complex dynamic scenarios of mobile edge computing (MEC). This approach aims to overcome the limitations of traditional optimization methods that cannot adapt online and in real-time to complex environments, while extending the application scope of existing deep reinforcement learning methodologies. First, our reward function (balancing latency and energy consumption), state representation (including resource utilization rates and user-server distances), and action space (associated with edge servers) collectively support dynamic and competitive MEC service migration scenarios with dual-objective optimization. Subsequently, we integrate three mechanisms - D3QN, prioritized experience replay, and noisy networks - into a unified framework. These three components respectively contribute to stable policy training, adaptive and efficient policy exploration, and accelerated convergence of policy learning, thereby enabling holistic online adaptive decision-making for service migration.

Markov decision process

Within the MEC system, the Deep Q-Network DQN agent mitigates latency via strategic service migration decisions. The system state, comprising user locations and resource utilization of edge servers, is solely determined by the state in the preceding time slot. This prior state is ascertained by the migration decision enacted by the agent. Consequently, we conceptualize the service migration process using a MDP. In this MDP framework, the state space, action space, and reward function pertaining to the agent are defined as follows.

State space

Since the migration decisions of agents depend on the utilization of edge servers by geographical location, we define the state at time slot *t* as follows.

$$s(t) = \{dist_{u,l}, n_{s,t}, F_{s,t}^{ava}, R_{u,l}, f_{\varphi,u}, v_\varphi^{size}\}, \tag{13}$$

Here, $dist_{u,l}$ is the distance between user *u* and its attached BS *l*, $n_{s,t}$ is the number of VMs created by ES *s* at time slot *t*. $F_{s,t}^{ava}$ is the available computational resources for ES *s* at time slot *t* (in percentage), where 0 means no computational resource available for *s*. $R_{u,l}$ is the uplink rate from user *u* to base station *l*, $f_{\varphi,u}$ is the number of CPU cycles needed to execute the service φ uploaded by user *u* and v_φ^{size} is the size of data to be migrated for the service φ.

Action space

Upon receiving information regarding the state of the environment, the agent executes a sequence of actions to engage with it. In this study, it is assumed that each time slot service can be relocated to any server within the base station's coverage area. Consequently, the action space is defined by the servers located within this specified coverage area, denoted as $a(t) \in S$.

Reward function

The aim of this study is to simultaneously minimize service delay and system energy consumption. To this end, we have devised a reward function that aggregates the negated values of the user's quality of service and the system's energy consumption. This formula is detailed below.

$$r(t) = (1 - \frac{\tau_{\varphi,t}(u)}{D_\varphi}) - C_{\varphi,t}^{mig}(u), \tag{14}$$

where $(1 - \frac{\tau_{\varphi,t}(u)}{D_\varphi})$ represents the quality service of the user, $\tau_{\varphi,t}$ signifies the time required for time slot *t* to process the user's service φ, and D_φ denotes the maximum acceptable delay for service φ.

Network architecture based on noisy-dueling DQN

In the DQN model, the neural network can take the state *s*(*t*) of each time slot as input and map the action *a*(*t*). The proposed NPER-DQN comprises three distinct network layers: an input layer, a fully connected layer,

and an output layer. Initially, the state vector $s(t)$ is fed into the neural network via the input layer, where state features are subsequently extracted by the fully connected layer. It is important to note that the Dueling DQN algorithm was employed in this process to compute both state and action values. Subsequently, the output layer determines the Q-value for each action by amalgamating the aforementioned state and action values. In the following sections, we provide a detailed elucidation of the Dueling DQN methodology.

In the standard DQN, the value function $Q_\pi(s(t), a(t); \theta)$ is approximated by the whole neural network directly. That is to say, the Q value for all actions are outputted in the output layer of the network without distinguishing the state value from the action advantage. In this case, it may happen that the agent only cares about the value of the state and ignores the difference caused by different actions. To avoid this problem, we decompose the Q-value function into a state value function V(s(t)) and an action value function A(s(t),a(t))[40]. Correspondingly, the neural network is split into a value branch for estimating V(s(t)) and an advantage branch for estimating A(s(t),a(t)). The final Q value is obtained by combining the outputs of both branches, as shown in Eq. (15).

$$Q_\pi(s(t), a(t); \theta, \alpha, \beta) = V(s(t); \theta, \beta) + (A(s(t), a(t); \theta, \alpha) - \frac{1}{|A|} \sum_a A(s(t), a(t); \theta, \alpha)), \tag{15}$$

where α represents the parameter associated with the dominance branch, while β corresponds to the parameter linked with the value branch. Furthermore, $|A|$ denotes the size of the action space.

In addition, we introduce noise in each layer of the network to replace the action exploration mechanism of traditional DQN. The fundamental premise of the Noisy Net approach entails the noisy parameterisation of weights and biases associated with the ith neuron within the neural network[41].

$$\omega_{ij} = \mu_{\omega ij} + \sigma_{\omega ij} - \epsilon_{\omega ij}, \tag{16}$$

$$b_{ij} = \mu_{bij} + \sigma_{bij} - \epsilon_{bij}, \tag{17}$$

where $\mu = \{\mu_{\omega ij}, \mu_{bij} | \forall i, j\}$ is a learnable mean parameter. $\sigma = \{\sigma_{\omega ij}, \sigma_{bij} | \forall i, j\}$ is a learnable noise amplitude parameter that controls the magnitude of the noise. $\epsilon = \{\epsilon_{\omega ij}, \epsilon_{bij} | \forall i, j\}$ samples the noise from a standard normal distribution.In this way, the output of the neural network is not only dependent on the input, but is also affected by the noise, thus introducing randomness in the selection of actions. Thus, in each fully connected layer (or other applicable layer), for input x_i, the output y_j is:

$$y_j = w_{ij} \cdot x_i + b_{ij}. \tag{18}$$

During training, the mean parameter μ and noise magnitude σ are learned simultaneously. This allows the model to automatically control the amount of exploration. Noisy Nets can increase their exploration at the beginning of training. As training progresses, the effect of noise decreases, allowing the model to gradually exploit its knowledge. At this juncture, the parameter θ in the value function $Q_\pi(s(t), a(t); \theta)$ is supplanted by an expression that amalgamates both the mean and the noise magnitude.

$$\theta = \{\mu, \sigma\}. \tag{19}$$

Framework of NPER-D3QN

Combined with the neural network structure given in "Markov decision process", we will give a specific framework of NPER-D3QN as shown in Figure 2. The NPER-D3QN incorporates three primary modules: D3QN, Noisy Net, and Prioritized Experience Replay. (1) D3QN module. The D3QN algorithm comprises a Q-network and a target Q-network. The Q-network selects the optimal action $a(t)$ based on the state-value function $V(s)$ and the action-value function $A(s, a)$, while the target Q-network is employed to evaluate the value of $a(t)$. During training, the agent computes the loss using outputs from both the Q-network and its target counterpart. Subsequently, it updates the parameters of the Q-network and intermittently copies them to the target network to ensure stable training. (2) Noisy Net module. The Noisy Net incorporates learnable parameterized noise (μ, σ) into the weights w and biases b of both the Q-network and the target Q-network. This approach replaces traditional ϵ-greedy exploration methods, enabling adaptive and efficient exploration. (3) Prioritized Experience Replay (PER) module. In PER, the quadruple $< s(t), a(t), r(t), s(t+1) >$ is stored as experience in memory. Mini-batches of high-priority experiences are then selected in a fair manner to train the D3QN module, thereby breaking data correlations.

Agent configuration based on double neural networks

Traditional DQN algorithms use a maximum value operation to select actions and estimate future values in a way that can lead to overestimation. In an effort to counter the problem of overestimation during the Q-value updating process, this study utilizes two neural networks to ensure stable training: namely, the action selection network and the target network[42]. Specifically, the role of the action selection network is to identify and select the action that corresponds to the maximum Q value, given the state of the environment.

$$a^*(t+1) = \underset{a}{argmax}\, Q_\pi(s(t), a(t); \theta, \alpha, \beta). \tag{20}$$

The evaluation of a selected action by the target network is conducted using the subsequent equation.

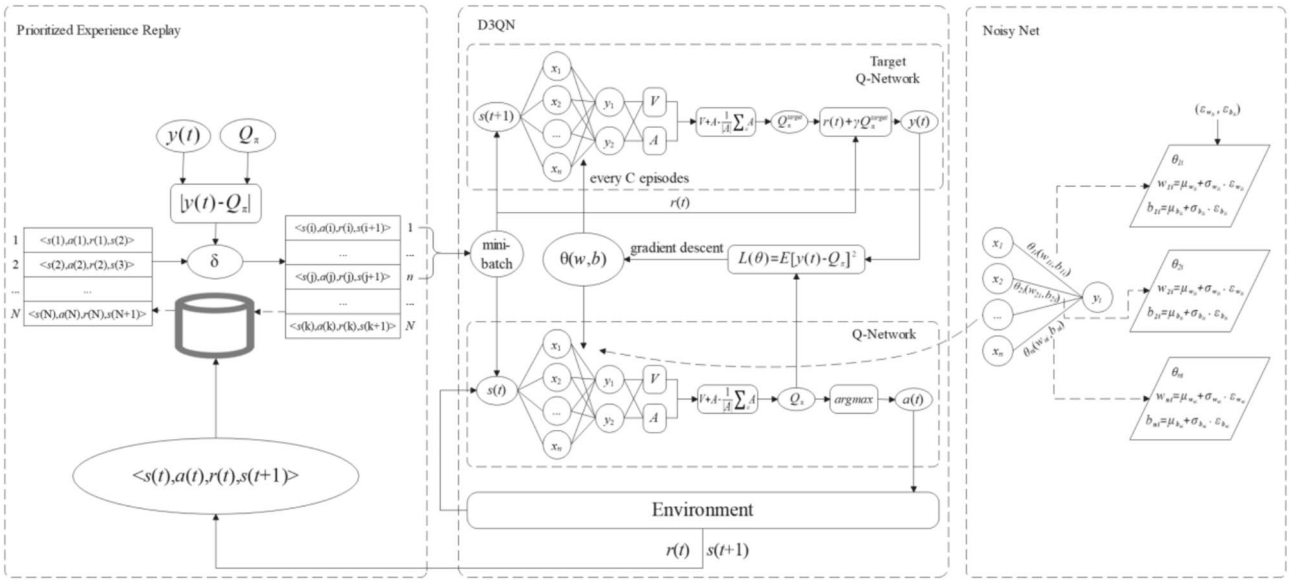


Fig. 2. A framework for adaptive service migration model.

$$y(t) = r + \gamma Q_{\pi}^{target}(s(t+1), a^{*}(t+1); \overline{\theta}, \alpha, \beta). \tag{21}$$

The target network parameters are periodically aligned with those of the action selection network, ensuring minimal variance in target values. During training, the D3QN iteratively adjusts the parameter θ to minimize the loss function. When samples are introduced into the existing network, a Q value is determined via forward propagation. The mean square error (MSE) loss is then calculated between the derived Q value and the intended target Q value as specified subsequently.

$$L(\theta) = E[(y(t) - Q_{\pi}(s(t), a(t); \theta, \alpha, \beta))^2]. \tag{22}$$

Subsequently, the network computes the gradient of the loss function with respect to the weights.

$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial Q_{\pi}(s(t), a(t); \theta, \alpha, \beta)} \cdot \frac{\partial Q_{\pi}(s(t), a(t); \theta, \alpha, \beta)}{\partial \theta}, \tag{23}$$

Here, θ_i represents any specific parameter within the network. Following this, the network parameters are updated using the gradient descent algorithm.

$$\theta \leftarrow \theta - \psi \frac{\partial L}{\partial \theta}, \tag{24}$$

where ψ denotes learning rate.

Noisy net-based action selection

Traditional DQN algorithms predominantly employ $\epsilon - greedy$ exploration techniques. The fundamental principle is that, when choosing an action, the current optimal action is predominantly selected. However, there exists a specific probability of choosing a random action (referred to as exploration). This probability is governed by the ϵ value.

$$a(t) = \begin{cases} Uniform(A), & with\ probability\ p, \\ argmax Q_{\pi}(s(t), a(t+1); \theta), & with\ probability\ 1-p. \end{cases} \tag{25}$$

While $\epsilon - greedy$ is simple and easy to implement, there are some obvious drawbacks, such as the possibility of insufficient exploration in the early stages and the possibility of wasting time on unnecessary exploration in the later stages. Specifically, the Noisy Net introduces noise terms with learnable variance to the weights and biases, enabling the network to automatically control the balance between exploration and exploitation during training. Given an input state s, the network yields Q-values for all actions via forward propagation. The incorporation of noise introduces a degree of randomness to each forward propagation, facilitating exploration at the strategy level. Consequently, the action selection procedure is streamlined to: choose the action corresponding to the maximum value within the output Q-value vectors.

$$a(t) = argmax\, Q_{\pi}(s(t), a(t); \theta, \alpha, \beta). \tag{26}$$

Priority-based experience replay

In the conventional DQN, the Experience Replay mechanism functions by storing experience fragments, denoted as $s(t+1)>$, acquired from the agents' interactions with the environment into a consolidated pool of experiences. Subsequent training involves randomly sampling from this pool. This process effectively decouples the correlation among samples, thereby enhancing data utilization. However, one problem with random sampling is that all experiences have the same probability of being selected, leading to underutilisation of samples with high learning value (e.g. samples with large TD errors). For this reason, we have introduced a Priority-based Experience Replay in D3QN. This mechanism records quaternions $< s(t), a(t), r(t), s(t+1) >$ as experiences in memory, increasing the probability that high-priority experiences are used for network training. The priority of experience *i* is determined by its temporal difference error δ_i[43], calculated as shown in (27).

$$\delta_i = |y(t) - Q(s(t), a(t); \theta, \alpha, \beta)|, \tag{27}$$

Experiences with larger δ are prioritised higher as they contribute more to the update strategy. To ensure fairness in sampling, experiences are selected using a priority sampling strategy. The probability of sampling experience *i* is represented as *P*(*i*) and is defined in equation (28).

$$P(i) = \frac{p_i^{\vartheta}}{\sum_k p_k^{\vartheta}}, p_i = |\delta_i + \zeta|, \tag{28}$$

where ϑ represents the degree of bias that controls the probability of being sampled, and ζ is a minimum positive value to ensure that samples with δ equal to zero are not completely ignored.

When the memory bank stores sufficient experiences, the agent samples a mini-batch according to *P*(*i*) and feeds them into both current and target networks for training. After training, the Q-network evaluates the longterm cumulative reward of each state-action pair.The action, denoted as *a*(*t*), that maximizes the *Q* value is chosen deterministically as the migration strategy $\pi(s(t))$.

$$\pi(s(t)) = \underset{a}{argmax}\, Q_\pi(s(t), a(t); \theta, \alpha, \beta). \tag{29}$$

Training of NPER-D3QN

Training algorithms

Training algorithms: Algorithm 1 outlines the training procedure for NPER-D3QN. Initialize an experience replay buffer *D* with a capacity of *N* , and randomly initialize the weight parameters of Q-network and target Q-network θ and θ^- (lines 1-3). In each training episode, we set the initial time slot *t* and the initial state *s*(*t*) (lines 5-6). At each time slot *t*, we feed *s*(*t*) into Q-network and select an action a(t) according to the current policy. Then we get a new state $s(t+1)$, and a reward *r*(*t*) by taking action *a*(*t*), and save the tuple $< a(t), r(t), s(t+1) >$ to buffer *D* (lines 8-13). The priority levels of the experience samples housed in *D* are adjusted utilizing the temporal-difference error, represented by δ (lines 14-15). Upon accumulation of a substantial volume of experiences, a priority-based mini-batch of data, denoted by $< s(j), a(j), r(j), s(j+1) >$, is sampled from the dataset *D*. The Q-network processes *s*(*j*) as input, while the target Q-network uses *r*(*j*) and $s(j+1)$, generating outputs Q_π and *y*(*j*), respectively. The model parameters, θ, are subsequently refined using gradient descent, informed by these outputs (lines 16-21). The migration policy, π, then derived as Eq. (29) (line 24). To ensure learning stability, the parameters θ of the target Q-network are methodically reset at each fixed time interval *C* (line 27).

Input: $N, M, T, n, C, dist_{u,l}, n_{s,t}, F_{s,t}^{aya}, R_{u,l}, f_{\varphi,u}, v_{\varphi}^{size}, \forall t \in T$.
Output: Migration policy π.
Initialize replay memory D with capacity N.
Initialize Q network with random weights θ.
Initialize target network Q^{-} with random weights θ^{-}.
for $episode = 1, M$ **do**
  Set time slot $t \leftarrow 1$.
  Initialize state $s(t) = \{dist_{u,l}, n_{s,t}, F_{s,t}^{aya}, R_{u,l}, f_{\varphi,u}, v_{\varphi}^{size}\}$.
  while $t \leq T$ **do**
    Input state $s(t)$ into Q network.
    Compute Q_{π} according to formula (15).
    Select action $a(t)$ using Q_{π} according to formula (26).
    Execute action $a(t)$ on environment, observe new state $s(t+1)$.
    Compute $r(t)$ according to formula (14).
    Store experience tuple $< s(t), a(t), r(t), s(t+1) >$ in D.
    Calculate δ of tuple $< s(t), a(t), r(t), s(t+1) >$ according to formula (27).
    Update priority of tuples in D based on δ.
    if $size(D) \geq n$ **then**
      Sample n tuples from D as minibatch d according to priority.
      for each $< s(i), a(i), r(i), s(i+1) >$ in minibatch d **do**
        Input $s(i)$ into Q network, and compute Q_{π} according to formula (15).
        Input $s(i+1)$ into target network Q^{-}, compute Q_{π}^{target} and $y(i)$ according to formulas (15) and (21).
        Execute a gradient descent step according to formula (23) with respect to the Q network parameter θ.
      end for
    end if
    Generate policy $\pi(s(t))$ according to formula (29).
    $t \leftarrow t+1$.
  end while
  Reset $\theta^{-} = \theta$ every C episodes.
end for

Algorithm 1. Training algorithm of NPER-D3QN.

Complexity analysis
The D3QN embedded with Noisy Net with PER still maintains a superior scalability in terms of overall complexity. Specifically, Noisy Net only adds noise parameter sampling and computation to forward propagation, assuming that the number of network parameters is Y. The time and space complexity remains on the same order as the size of the network parameters $O(Y)$. It does not bring about an extra burden. Then the replay of the priority experience introduces data structures such as Sum-Tree when sampling and updating priorities, increasing the time complexity from $O(1)$ to $O(logN)$. N represents the capacity of the experience pool. Consequently, the time complexity associated with a training update is $O(Y + logN)$. Since Y is typically much larger than $logN$ in practical applications, the main computational overhead of the algorithm remains concentrated on the forward and backward propagation through the network, with an overall complexity that increases only slightly compared to original D3QN, while space complexity is maintained at $O(Y + N)$.

Performance evaluation

Experimental setup

Experimental environment
We employ a 6×6 grid environment as the simulation setup for mobile edge computing, with specific parameters listed in Table 2. In this environment, each grid cell deploys one BS, and we assume each BS covers its respective grid cell area, enabling communication with users within the cell. To better approximate real-world conditions, we randomly deploy 6 to 36 MEC servers among these BSs. Each server has a total CPU capacity of 7 billion instructions per second, 16GB memory, and service instance size of 0.5GB. Meanwhile, each server can host up to 10 virtual machines, with each VM having 0.5GB memory. We assume the number of users varies dynamically between 20 and 120. During each time slot, users move randomly in four directions (up, down, left, right) within the grid and offload tasks to ES. Each user u offloads tasks sized between 2-5 MB at time t, with all tasks having equal priority. The rate of task arrivals follows a Poisson distribution with the parameter λ having a range of $\lambda = [1, 2]$. Meanwhile, the CPU cycles required for each task are uniformly distributed, ranging from 50 to 100 million instructions. The orthogonal resource blocks are allocated a bandwidth of 180kHz and a quantity of 50, with the user transmission power adjusted to 200 mW. The noise power during transmission is assessed at -174dBm/Hz, the path loss exponent is calculated at 3.75, and the power control factor is established at 0.25. Additionally, the backhaul delay coefficient for task migration is set to 0.02s/hop, with wired backhaul link bandwidth of 1Gbps and migration energy coefficient of 2. Specifically, we suppose that all the tasks can be

Parameter	Symbol	Value
Total BS locations	$\|L\|$	36
Number of MEC servers range	$\|S\|$	$6 \sim 36$
Number of UEs	$\|U\|$	$20 \sim 120$
Service instance size	v_{φ}^{size}	0.5 GB
Average task arrival rate	λ	$P \sim [1, 2]$
Number of CPU cycles required by φ	$f_{\varphi,u}$	$50 \sim 100$ MI
Offload task size of u at time slot t	$data_{u,t}$	$2 \sim 5$ Mb
Resource block bandwidth	W	180 kHz
Number of orthogonal resource blocks	K	50
Baseline transmit power of users	P	200 mW
Noise power	N_0	-174 dBm/Hz
Path loss exponent	ϖ	3.75
Power control factor	ς	0.25
Wired link bandwidth	B	1 Gbps
Migration cost consumption factor	η	2
Backhaul delay factor	ρ	0.02 s/hop

Table 2. Parameters of the simulated environment.

Parameter	Symbol	Value
Episode	M	500
Time slots	T	80
Replay Memory	N	10000
Batch size	n	64
Reward discount factor	γ	0.99
Initial value of noise parameter	σ	0.017
Target network update frequency	C	50
Learning rate	ψ	0.001

Table 3. Hyperparameters of NPER-D3QN.

completed in a single time slot t, and we also assume that at the beginning of each time slot t, migration happens on the granularity of user-configured virtual machine level instead of the task level.

Model parameter

Table 3 specifies the training parameters for all models. The model undergoes 500 episodes, with each episode comprising 80 time slots. The experience replay buffer has a capacity of 10,000 samples, with a mini-batch size of 64 for random sampling. The noise parameters in the network are initialized to 0.017. The reward discount is fixed at 0.99, while the learning rate is configured to 0.001. We implement updates to the target network parameters on a schedule of every 50 episodes.

Evaluation metrics

For compute-intensive tasks, CPU performance directly affects application response time and processing capability, while memory and bandwidth have relatively minor impacts. Similarly to, we use the CPU as a metric to evaluate server resources and assess the efficacy of our solution through cumulative reward, average service latency, average migration energy consumption, and average migration rejection rate[20]. The cumulative reward refers to the total long-term return obtained by the agent during reinforcement learning, reflecting the model's convergence and the effectiveness of policy optimization. A higher cumulative reward indicates that the model can gradually develop better resource scheduling strategies through training. Average service latency is defined as the mean service latency across all time slots. A smaller value indicates that the system provides more efficient service migration and a better user experience. The calculation process is shown in Equation (30). The average migration energy consumption is defined as the mean energy consumed in service migrations between servers across all measured time slots. This provides an insight into the overall cost associated with system services. The calculation process is shown in Equation (31). The average migration rejection rate is defined as the ratio of service migration requests declined due to resource scarcity to the total number of such requests. This metric measures the effectiveness of successful migrations. This rate is calculated as shown in Equation (32).

$$\bar{\tau} = \frac{\sum_{t=1}^{T} \sum_{\varphi=1}^{\phi} \tau_{\varphi,t}(u), \forall u \in U}{T}, \tag{30}$$

where $\tau_{\varphi,t}(u)$ represents the delay encountered by user u in the processing of service request φ within the time frame of slot t, while T indicates the total number of such time slots.

$$\bar{C} = \frac{\sum_{t=1}^{T} \sum_{\varphi=1}^{\phi} C_{\varphi,t}^{mig}(u), \forall u \in U}{T}, \tag{31}$$

where $C_{\varphi,t}^{mig}(u)$ represents the energy consumption associated with migration during time slot t to address the service request φ for user u.

$$r_{rej} = \frac{n_{rej}}{n_{mig}}, \tag{32}$$

where n_{rej} represents the count of instances, while n_{mig} denotes the overall quantity of migration requests.

Baseline methods

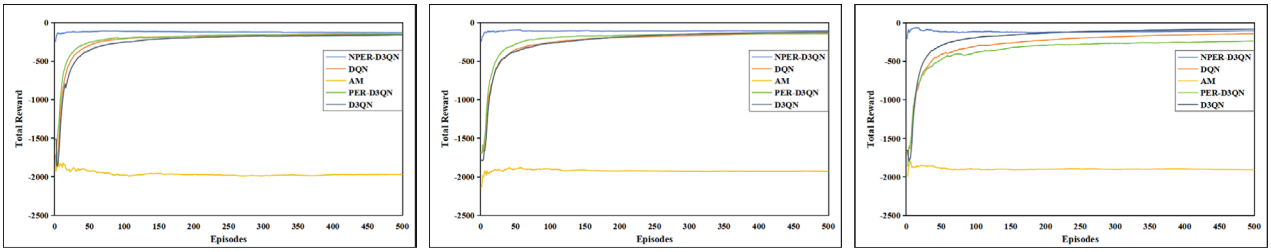
We compared our proposed method with six recently updated baseline methods, which are detailed below.

- DQN[20]: This scheme uses the traditional DQN algorithm for service migration.
- D3QN[44]: This scheme is an ablation of our proposed method, targeting latency and energy consumption, and using the D3QN algorithm for service migration.
- PER-D3QN[45]: This scheme is an ablation of our proposed method, targeting latency and energy consumption, and using the PER-D3QN algorithm for service migration.
- Delay[46]: The scheme aims to reduce latency by integrating DDQN with LSTM, thus enhancing the long-term memory of the model.
- Dist[18]: The proposed scheme is an adaptation of the traditional DQN, specifically engineered to facilitate service migration based on terminal location. This consistently ensures a minimal distance between the user and the edge server (ES).
- AM: This approach is rooted in a conventional DQN, which facilitates the migration of services to the most suitable server within each time slot.

Experimental results and analysis

Convergence analysis

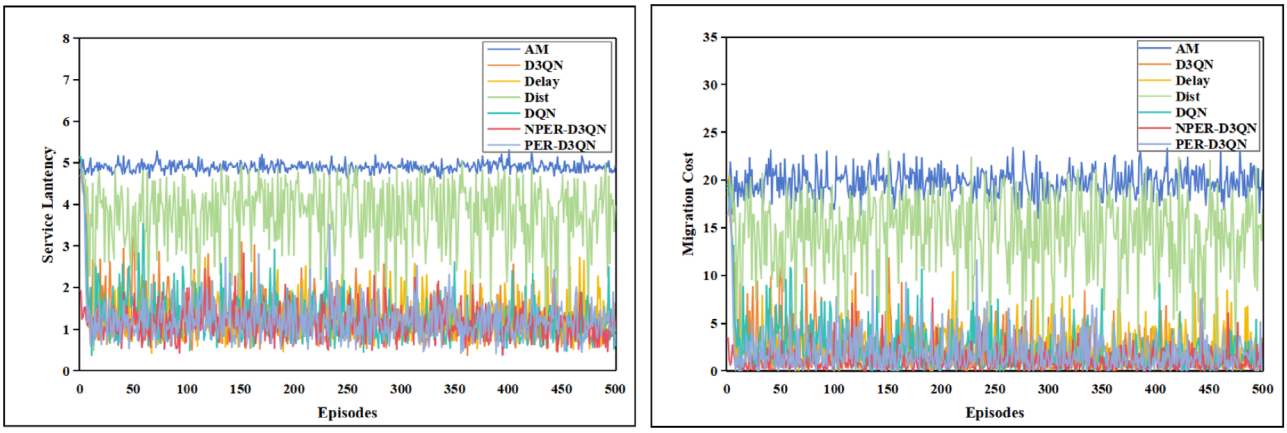
To thoroughly assess the efficacy of the proposed NPER-D3QN algorithm, we perform a comparative analysis against six benchmark schemes. This analysis considers various parameters, including convergence speed, service delay, migration energy consumption, and migration rejection rate. We first analyze the convergence speed of different solutions by quantifying their cumulative rewards. Figure 3(a)-(f) shows the cumulative rewards of different solutions under varying numbers of MEC servers when there are 60 users. Since the Delay and Dist solutions have different reward functions, they are not discussed here. In the initial stage, all solutions are in the



(a) $|S| = 6$ MEC servers (b) $|S| = 12$ MEC servers (c) $|S| = 18$ MEC servers

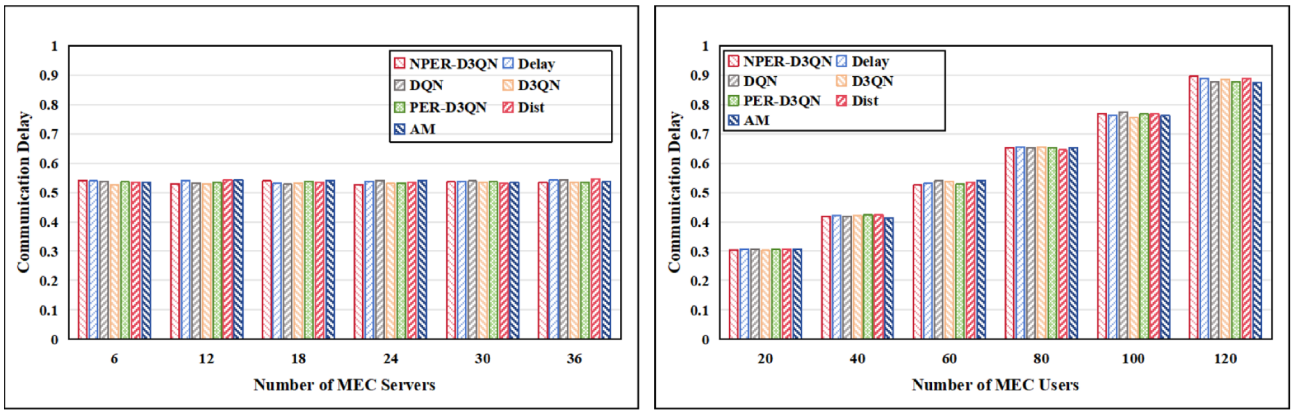
(d) $|S| = 24$ MEC servers (e) $|S| = 30$ MEC servers (f) $|S| = 36$ MEC servers

Fig. 3. Total reward depicting NPER-D3QN convergence rate with $|U| = 60$.



(a) Service delays during training (b) Migration energy consumption during training

Fig. 4. Latency and energy consumption during training for different scenarios when $|U| = 60$ and $|S| = 24$.



(a) Impact of MECs on communication delay(60 Users). (b) Impact of Users on communication delay(24 MECs).

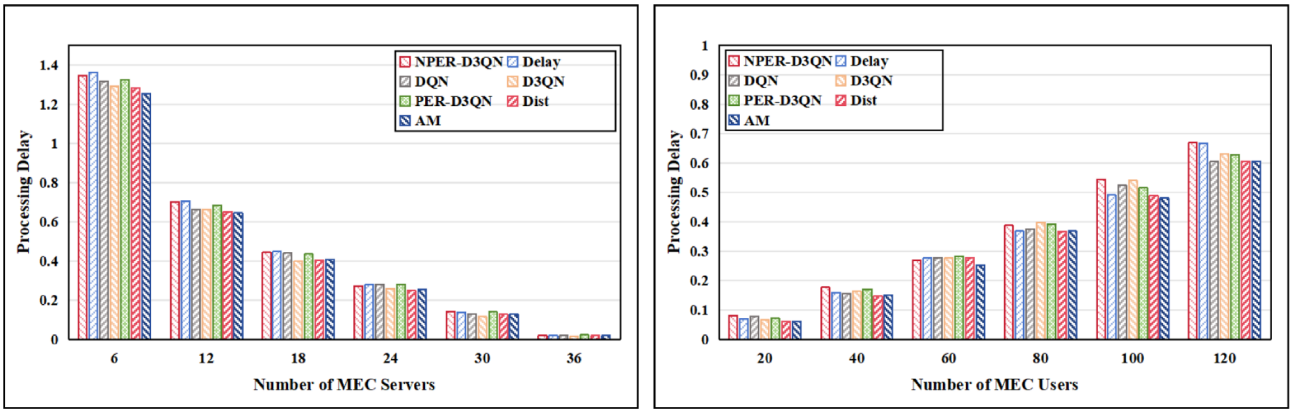
Fig. 5. Communication delay performance of schemes with different numbers of MEC servers and users.

exploration phase and need to continuously optimize migration strategies through trial and error. Among them, DQN, D3QN and PER-D3QN requires 100-150 episodes to converge, while both NPER-D3QN and AM can converge within 50 episodes. Due to its constant migration behavior, AM achieves the lowest cumulative reward. DQN, D3QN and PER-D3QN have the second-lowest cumulative reward because of its lower training efficiency. In contrast, NPER-D3QN can learn better strategies to make optimal trade-offs between QoS and migration energy consumption, consistently maintaining the highest cumulative reward. Specifically, NPER-D3QN is capable of making migration decisions based on dual criteria: the utilization of resources within the MEC servers and the geographical distance between the users and the servers. This approach allows it to minimize service latency and migration energy consumption respectively, thereby maximizing long-term rewards. Consequently, under resource-constrained conditions, NPER-D3QN achieves higher cumulative rewards than other solutions.

Figure 4 illustrates the delay and energy consumption during the training phase for various schemes. It is evident that the D3QN, enhanced by the incorporation of Noisy Net and the preferred experience replay mechanism, exhibits superior performance. Notably, its delay performance markedly surpasses that of the traditional D3QN. Concurrently, due to a more streamlined strategy selection, there is a discernible reduction in redundant exploration and unnecessary computations, leading to overall lower energy consumption. In comparison, the remaining six baseline methodologies exhibit shortcomings in both delay and energy consumption management, underscoring that the approach presented in this paper optimizes energy usage while simultaneously enhancing training efficiency.

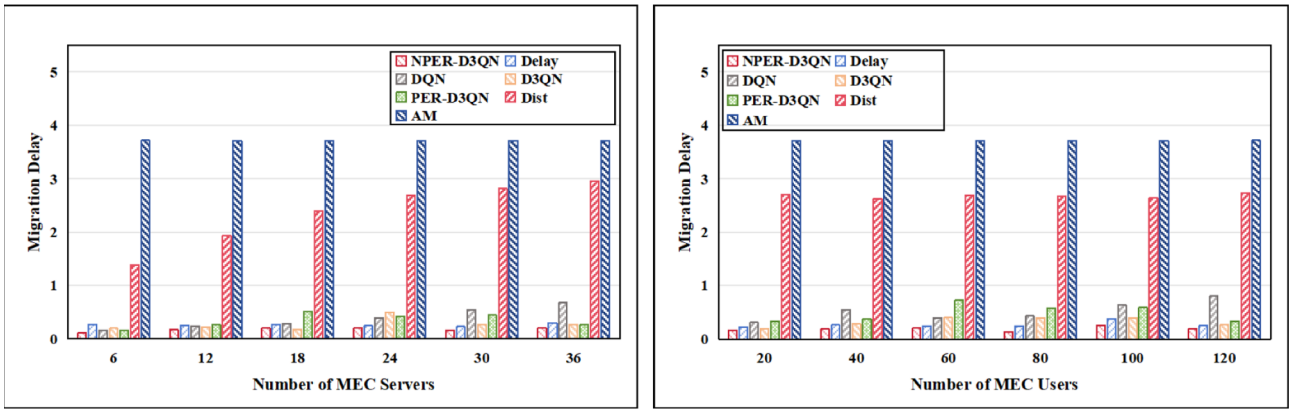
Delay performance analysis

The aim of service migration is to enhance the user experience by diminishing service latency, which includes communication latency, computation latency, and migration latency. The primary factor influencing communication latency is the number of users, as it determines the volume of tasks. Figure 5(a) shows that communication latency remains relatively stable, even with an increase in MEC servers, and different schemes



(a) Impact of MECs on processing delay(60 Users). (b) Impact of Users on processing delay(24 MECs).

Fig. 6. Processing delay performance of schemes with different numbers of MEC servers and users.



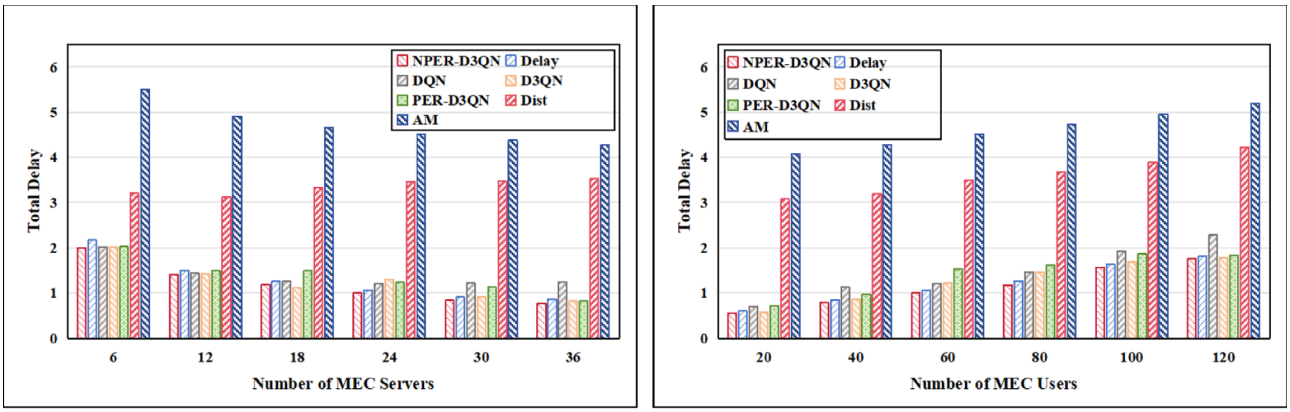
(a) Impact of MECs on migration delay(60 Users). (b) Impact of Users on migration delay(24 MECs).

Fig. 7. Migration delay performance of schemes with different numbers of MEC servers and users.

exhibit similar communication latencies. However, when the number of users increases, as shown in Figure 5(b), all schemes display a rising trend in communication latency. This can be attributed to the increase in users leading to a higher volume of tasks being uploaded, thereby increasing the communication latency. It is evident that the difference in communication delay under the seven scenarios is minimal. This is because the communication delay is only influenced by the task size and the transmission speed of the wireless link, and does not depend on the enhancement of the migration programme.

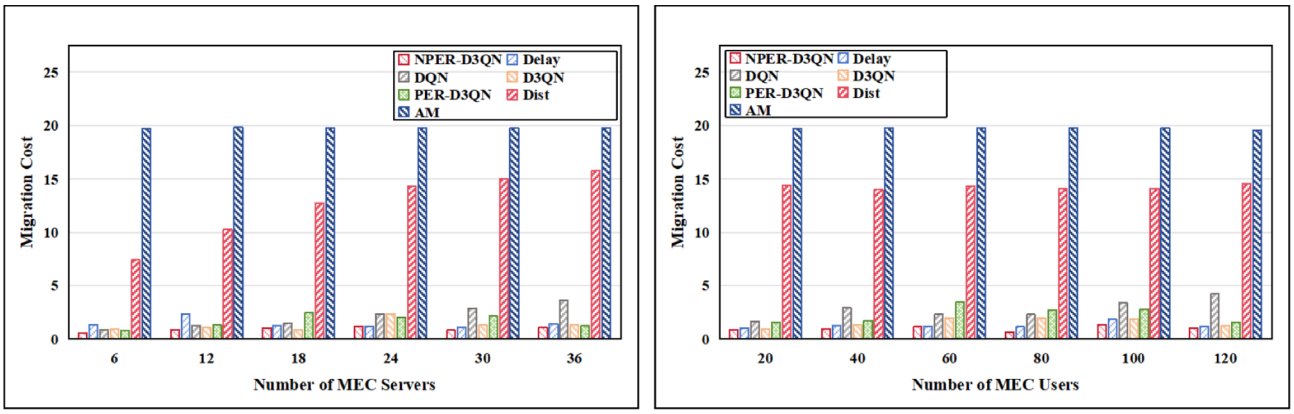
Figure 6 presents the performance of communication latency for all schemes under different conditions. As shown in Figure 6(a), the processing latency of each scheme decreases with the increase of MEC servers. The reason is that more servers provide more computing resources, so the schemes can offload tasks from busy MEC servers to idle ones. Consequently, this reduces task queuing duration and enhances resource utilization. Figure 6(b) demonstrates the processing latency performance with varying numbers of users. The results indicate that processing latency increases with more users, since the user count determines the number of tasks and consequently affects their processing time. Moreover, a higher task volume leads to insufficient computational resources, resulting in longer queuing times. Among the seven schemes, the AM scheme achieves the lowest processing latency. This is because AM minimizes queuing delays by frequently migrating tasks to idle servers. Our proposed scheme, however, must balance various latencies and migration energy consumption, and therefore avoids service migration when queuing delays are minor. As a result, while our solution shows slightly inferior processing latency performance compared to others, the difference remains small.

Figure 7 presents the migration latency performance of each scheme under different conditions. As depicted in the figure, an increase in either the number of servers or users leads to an upward trend in the migration latency of each scheme. This is due to the fact that as the number of servers increases, each scheme intensifies the migration frequency to reduce the processing latency of tasks. Conversely, when the number of users rises, the limited computing resources of servers require each scheme to frequently migrate services to less occupied servers, thereby minimizing queuing time. It is evident that our scheme consistently exhibits the lowest migration latency under various conditions. This is attributed to the fact that NPER-D3QN can optimally balance service



(a) Impact of MECs on service delay(60 Users). (b) Impact of Users on service delay(24 MECs).

Fig. 8. Service latency performance of scenarios with different number of MEC servers and users.



(a) Impact of MECs on migration cost(60 Users). (b) Impact of Users on migration cost(24 MECs).

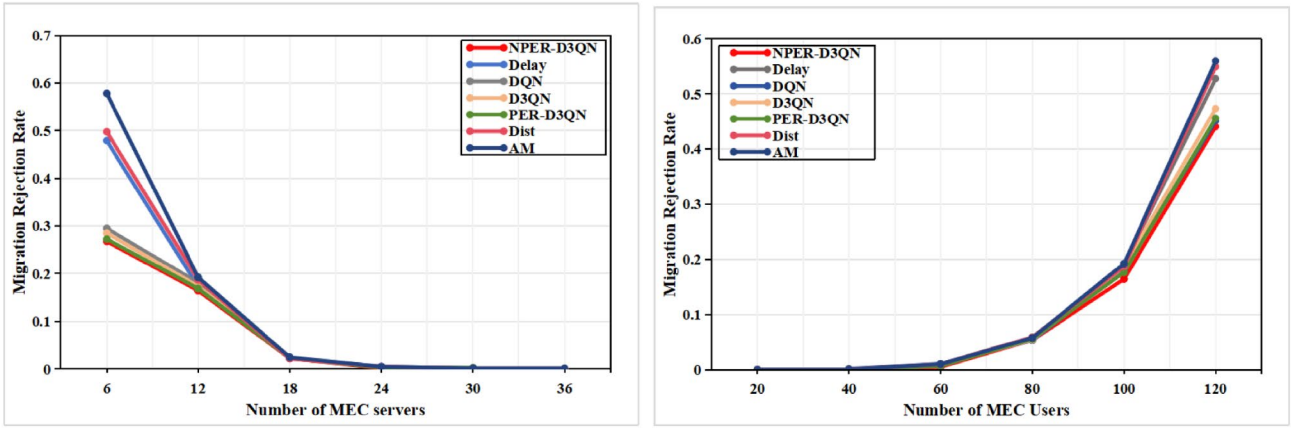
Fig. 9. Migration energy performance of schemes with different MEC servers and number of users.

latency and migration latency, and can make migration decisions based on the difference in total latency before and after migration. The AM scheme constantly migrates services, resulting in a relatively high migration latency.

Figure 8(a) and (b) describe the average service latency of the seven schemes in terms of different numbers of MEC servers and users, respectively. As shown in Figure 8(a), the service latency decreases with the increase in the number of MEC servers. This is because more resources can be provided by more servers so that the tasks can be processed faster. The average service delay of NPER-D3QN is reduced by 7.4%, 14.2%, 5.3%,12.2%,64.3% and 74.6% compared to Delay, DQN, D3QN, PER-D3QN, Dist and AM, respectively. In Figure 8(b), service latency shows an upward trend with increasing user numbers. This occurs because more tasks lead to the creation of more virtual machines, resulting in higher processing latency. Therefore, appropriate migration is essential for reducing service latency. NPER-D3QN produces the lowest service latency, with reductions of 5.1%, 21.5%, 9.8%, 20.1%, 68.4%, and 75.4% compared to Delay, DQN, D3QN, PER-D3QN, Dist, and AM schemes, respectively. This advantage stems from NPER-D3QN's ability to migrate services to suitable MEC servers for processing. In contrast, Delay, DQN, D3QN and PER-D3QN exhibit lower learning efficiency and consequently make poorer migration decisions, while Dist and AM incur higher migration latency due to their frequent migration behavior.

Migration energy performance analysis

Another goal of service migration is to reduce operator cost by reducing the amount of energy consumed during migration. The amount of energy consumed during migration depends on the frequency and distance of migrations, i.e., the more frequently services migrate and the longer the migration distance, the more energy will be consumed. Figure 9(a) and (b) show the average migration energy consumption of the seven schemes in terms of different numbers of MEC servers and users, respectively. As shown in Figure 9(a), the migration energy consumption increases with the increase of the number of available MEC servers. The reason is that when there are more servers available, each scheme performs more migrations to reduce service latency. NPER-D3QN produces lower migration energy consumption compared to Delay, DQN, D3QN, PER-D3QN, Dist, and AM schemes by 34.9%, 54.3%, 28.0%, 43.0%, 92.5%, and 95.2%, respectively. Similarly, Figure 9(b) shows



(a) Impact of MECs on migration rejection rate(100 Users). (b) Impact of Users on migration rejection rate(12 MECs).

Fig. 10. Migration rejection rate performance of schemes with different MEC servers and number of users.

Methods	Service delay	Migration energy	Migration rejection rate	Mean runtime
AM	6.56	36.14	0.170	92.84
Dist	5.61	29.45	0.142	115.42
Delay	3.320	11.61	0.161	116.12
DQN	4.49	18.48	0.154	106.39
D3QN	3.17	6.51	0.148	107.67
PER-D3QN	2.824	4.12	0.151	105.1
NPER-D3QN	2.434	2.24	0.131	107.5

Table 4. Large-scale performance and runtime of different methods.

that migration energy consumption also increases with more users. This occurs because more users lead to more tasks, which require more frequent migrations to servers with sufficient computing resources. Among the seven scenarios, NPER-D3QN produced the least migration energy consumption, which was reduced by 21.6%, 63.8%, 35.0%, 56.5%, 92.9%, and 94.9% over Delay, DQN, D3QN, PER-D3QN, Dist, and AM, respectively. This is because NPER-D3QN fully considers migration distance and minimizes unnecessary migrations. In contrast, Delay only considers service latency without accounting for migration energy. DQN, D3QN and PER-D3QN cannot make optimal migration decisions due to its learning inefficiency. Dist only considers the distance between users and MEC servers but ignores migration distance. AM results in high migration energy consumption due to its frequent migration behavior.

Migration rejection rate performance analysis

In light of the constrained resources within MEC servers, service migration does not always proceed successfully. When a migration attempt is thwarted, the system is compelled to either reassess its migration targets or altogether abort the migration, resulting in additional waiting delays that degrade user QoS. Figure 10 presents the service migration rejection rates for all schemes. In our experiments, if a server already hosts 10 services, it will reject additional services. As shown in Figure 10(a), when the number of servers is small, the service rejection rate is relatively high. Compared with DQN, Delay, D3QN, PER-D3QN, Dist and AM, our proposed scheme NPER-D3QN has the lowest service denial rate, with average reductions of 9.1%, 32.7%, 5.9%, 2.2%, 35.8% and 42.9%, respectively. Figure 10(b) illustrates a direct correlation between an increase in users and the rate of migration rejection. This occurs because more users occupy more computing resources, leaving most servers unable to accept additional services. In contrast, the migration rejection rate of NPER-D3QN was significantly lower than that of the other six options, with reductions of 4.8%, 15%, 8.2%, 4.5%, 17.6%, and 19%, respectively. This superior performance stems from NPER-D3QN's comprehensive consideration of server resource utilization when making migration decisions. While DQN, D3QN and PER-D3QN considers resources but fails to learn optimal migration strategies, resulting in suboptimal performance, Delay, Dist and AM schemes completely ignore resource utilization, leading to higher rejection rates.

Large-scale performance analysis

To evaluate the performance of our proposed method in a large-scale scenario[47], we set the number of users to 720 and the number of servers to 96 to test service latency, migration energy consumption, migration rejection rate, and system response speed. As shown in Table 4, for service latency, we record the average delay from when a user initiates a service request until it is completed and compare it with existing benchmark methods. The

experimental results show that the proposed method can effectively maintain low service latency under high user load, thus protecting the user experience. Compared with six baseline methods (AM, Dist, Delay, DQN, D3QN, and PER-D3QN), the maximum reduction in service latency achieved by the proposed method is 62.9% (AM), the minimum is 13.8% (PER-D3QN), and the average is 38.2%, which indicates significant improvement. Similarly, as the number of users increases, our scheme still maintains the lowest energy consumption at only 2.24. When compared with the other six benchmark schemes, this represents maximum energy savings of 93.8% (AM), minimum savings of 45.6% (PER-D3QN), and an average reduction of 77.6%, demonstrating significant efficiency improvements. In addition, our scheme also maintains the lowest migration rejection rate as the number of users increases, achieving reductions of 22.9%, 7.8%, 18.6%, 14.9%, 11.5%, and 13.2% compared to the above six baseline schemes, respectively, with an average reduction of 32.4%, which remains significantly effective. This shows that as the user scale continues to expand, our method can always maintain its advantages of low latency, low energy consumption, and low rejection rate, indicating excellent scalability and robustness.

We implement the decision agent in our simulations as a logically centralized controller that collects required system state at the beginning of each time slot and outputs migration actions. This design facilitates benchmarking under unified policies, but for large-scale deployments, wide-area state collection may introduce communication overhead and raise concerns about scalability and privacy. Practical deployment favors hierarchical or divide-and-conquer implementations, where large-scale MEC networks are partitioned into subregions with local controllers and lightweight cross-region coordination can support decentralized alternatives such as multi-agent DRL or federated DRL frameworks to enable distributed privacy-preserving decision-making.

Runtime analysis

We also evaluate the average runtime (s)[47] of each method to analyze computational complexity and show the feasibility of real-time deployment. As shown in Table 4, we present the runtime per iteration for each method. Each iteration contains 80 time slots during which user positions may change. It can be observed that AM exhibits the shortest runtime, but its average service delay, migration energy consumption, and migration rejection rate are all the highest, indicating an inverse relationship between runtime and method performance. Delay demonstrates the longest runtime, followed by Dist, yet the overall performance of the former surpasses the latter. The performances of both methods (except for Delay's migration rejection rate) remain inferior to the other four approaches. Our proposed method shows similar execution time compared to the other three methods, positioned at an intermediate level. Compared with the shortest-runtime method among the three (PER-D3QN), our approach increases runtime by 1.3% (1.4 s), while achieving over 15.3% improvements in average service delay, migration energy consumption, and migration rejection rate. The results demonstrate that our model achieves a favorable trade-off between computational efficiency and performance improvement, i.e., it can significantly enhance performance with minimal training overhead, which holds positive implications for practical real-time decision-making under the modeled assumptions. However, the end-to-end feasibility in actual deployment remains contingent on system integration factors (such as state acquisition costs and network dynamics), which require further validation in future work.

Conclusion

This paper addresses the service migration problem for multi-users in a Mobile Edge Computing (MEC) environment. Considering users' mobility patterns, resource contention relationships, as well as the joint optimization of service latency and system energy consumption, we propose a novel systematic approach tailored to this specific complex problem. In theoretical modeling, we formulate the multi-user service migration problem as a constrained multi-objective optimization framework that minimizes both service latency and total energy consumption under resource competition. For algorithm design, we develop an enhanced Deep Reinforcement Learning (DRL)-based service migration model named NPER-D3QN. This model improves Q-value estimation accuracy through dueling Q-learning architecture and competitive networks, while incorporating prioritized experience replay mechanisms and Noisy Networks modules to accelerate convergence, thereby generating high-performance service migration policies. Regarding migration mechanisms, we transform multi-user service migration into sequential single-user migrations, which eliminates inter-user resource contention and avoids predicting user mobility trajectories, enabling adaptive real-time migration decisions. Simulation results demonstrate that compared with six baseline methods, our proposed approach achieves significant reductions in service latency, system cost, and migration rejection rate, with average performance improvements of 31.53%, 59.39%, and 16.48% respectively, indicating the superior performance of the NPER-D3QN model. Furthermore, experimental results on large-scale scenarios and algorithm runtime show that our proposed method exhibits excellent scalability and low computational overhead. Despite the encouraging results, there are several limitations to this work. First, our evaluation is based on simulations and abstracted physical network infrastructure and protocol-level effects; thus, practical aspects such as signaling overhead, packet loss, and noisy/delayed state observations are not explicitly captured. Second, the energy model focuses only on migration-related communication energy consumption and does not include full system-level energy components such as idle/active power of edge servers, cooling costs, or carbon footprint, which could impact absolute energy savings in practice. Third, we employ a logically centralized agent with wide-area state information access rights, which, while amenable to divide-and-conquer approaches for large-scale deployments, raises privacy concerns. In future work, we plan to validate the proposed approach using real mobility traces and/or an edge testbed, extend the modeling framework to partially observable settings, and explore the integration of decentralized implementation schemes (e.g., hierarchical, multi-agent, or federated DRL) with more detailed system-level energy consumption modeling.

Data availability

All data generated or analysed during this study are included in this published article and its supplementary information files.

Received: 3 October 2025; Accepted: 14 January 2026
Published online: 24 January 2026

References

1. Ning, Z. et al. Mobile edge computing and machine learning in the internet of unmanned aerial vehicles: A survey. *ACM Comput. Surv.* **56**, 13:1-13:31 (2024).
2. 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.* **11**, 11341–11352 (2024).
3. Yuan, Y. et al. Service migration optimization for system overhead minimization in vecns via deep reinforcement learning. *IEEE Internet Things J.* **12**, 3905–3920 (2025).
4. Peng, Y. et al. Computing and communication cost-aware service migration enabled by transfer reinforcement learning for dynamic vehicular edge computing networks. *IEEE Trans. Mob. Comput.* **23**, 257–269 (2024).
5. Huang, X. et al. Optimizing task migration for public and private services in vehicular edge networks: A dual- layer graph neural network approach. *IEEE Trans. Mob. Comput.* **24**, 13191–13208 (2025).
6. Han, Y., Li, X. & Zhou, Z. Dynamic task offloading and service migration optimization in edge networks. *Int. J. Crowd Sci.* **7**, 16–23 (2023).
7. Tsourdinis, T., Makris, N., Fdida, S. & Korakis, T. Drl-based service migration for mec cloud-native 5g and beyond networks. In *Network Softwarization*. 62–70 (2023).
8. Kang, J. et al. UAV-assisted dynamic avatar task migration for vehicular metaverse services: A multi-agent deep reinforcement learning approach. *IEEE/CAA J. Autom. Sin.* **11**, 430–445 (2024).
9. Li, X., Zhou, Z., Wang, Y., Deng, S. & Hung, P. C. K. Service migration for delay-sensitive IOT applications in edge networks. *IEEE Trans. Serv. Comput.* **18**, 1782–1797 (2025).
10. Gao, Z., Yang, L. & Dai, Y. VRCCS-AC: Reinforcement learning for service migration in vehicular edge computing systems. *IEEE Trans. Serv. Comput.* **17**, 4436–4450 (2024).
11. Wang, C. et al. AI-enabled spatial-temporal mobility awareness service migration for connected vehicles. *IEEE Trans. Mob. Comput.* **23**, 3274–3290 (2024).
12. Li, C., Zhang, Q. & Luo, Y. A jointly non-cooperative game-based offloading and dynamic service migration approach in mobile edge computing. *Knowl. Inf. Syst.* **65**, 2187–2223 (2023).
13. Cao, B. et al. Smart: Cost-aware service migration path selection based on deep reinforcement learning. *IEEE Trans. Intell. Transp. Syst.* **25**, 12421–12436 (2024).
14. Wang, C. et al. AI-enabled spatial-temporal mobility awareness service migration for connected vehicles. *IEEE Trans. Mobile Comput.* **23**, 3274–3290 (2024).
15. Zheng, G., Navaie, K., Ni, Q., Pervaiz, H. & Zarakovitis, C. Energy-efficient secure dynamic service migration for edge-based 3-D networks. *Telecommun. Syst.* **85**, 477–490 (2024).
16. Zhang, X. et al. Optimization of Service Migration Decisions in Mobile Edge Computing Based on Markov Decision Processes. In *The 3rd International Joint Conference on Information and Communication Engineering*. 92–96 (2024).
17. Park, S. W., Boukerche, A. & Guan, S. A novel deep reinforcement learning based service migration model for mobile edge computing. In *IEEE/ACM International Symposium on Distributed Interactive Simulation and Real-Time Applications*. 1–8 (2020).
18. Shi, Y. et al. Service migration or task rerouting: A two-timescale online resource optimization for MEC. *IEEE Trans. Wirel. Commun.* **23**, 1503–1519 (2024).
19. Wang, J., Hu, J. & Min, G. Online service migration in edge computing with incomplete information: A deep recurrent actor-critic method. In *Computing Research Repository*. arXiv:2012.08679 (2020).
20. Mwasinga, L. J. et al. RASM: Resource-aware service migration in edge computing based on deep reinforcement learning. *J. Parallel Distrib. Comput.* **182**, 10745 (2023).
21. Zhou, X., Ge, S., Qiu, T., Li, K. & Atiquzzaman, M. Energy-efficient service migration for multi-user heterogeneous dense cellular networks. *IEEE Trans. Mobile Comput.* **22**, 890–905 (2023).
22. Kumar, P. S., Kumar, P. S. & Satyabrata, D. Dynamic service migration and resource management for vehicular clouds. *Ambient Intell. Hum. Comput.* **12**, 1227–1247 (2020).
23. Anwar, M. R., Wang, S., Akram, M. F., Raza, S. & Mahmood, S. 5G-enabled MEC: A distributed traffic steering for seamless service migration of internet of vehicles. *IEEE Internet Things J.* **9**, 648–661 (2022).
24. Chen, J. et al. Multi-agent deep reinforcement learning for dynamic avatar migration in AIOT-enabled vehicular metaverses with trajectory prediction. *CoRR* arXiv:2306.14683 (2023).
25. Zhao, X. et al. MAPSM: Mobility-aware proactive service migration framework for mobile-edge computing in consumer internet of vehicles. *IEEE Trans. Consumer Electron.* **71**, 3753–3766 (2025).
26. Chen, Z. et al. Mobility-aware seamless service migration and resource allocation in multi-edge IOV systems. *IEEE Trans. Mob. Comput.* **24**, 6315–6332 (2025).
27. Li, X., Chen, S., Zhou, Y., Chen, J. & Feng, G. Intelligent service migration based on hidden state inference for mobile edge computing. *IEEE Trans. Cognit. Commun. Netw.* **8**, 380–393 (2022).
28. Chen, W. et al. MSM: Mobility-aware service migration for seamless provision: A data-driven approach. *IEEE Internet Things J.* **10**, 15690–15704 (2023).
29. Peng, Y. et al. Computing and communication cost-aware service migration enabled by transfer reinforcement learning for dynamic vehicular edge computing networks. *IEEE Trans. Mobile Comput.* **23**, 257–269 (2024).
30. Fan, Q., Chen, L., You, C., Chen, Y. & Yin, H. Dependency-aware service migration for backhaul-free vehicular edge computing networks. *IEEE Trans. Veh. Technol.* **73**, 1337–1352 (2024).
31. Labriji, I. et al. Mobility aware and dynamic migration of MEC services for the internet of vehicles. *IEEE Trans. Netw. Serv. Manag.* **18**, 570–584 (2021).
32. Bozkaya-Aras, E. Optimizing service migration in IOT edge networks: Digital twin-based computation and energy-efficient approach. In *2025 IEEE Wireless Communications and Networking Conference (WCNC), Milan, Italy, March 24-27, 2025*. 1–6 (2025).
33. Yin, L., Li, P. & Luo, J. Smart contract service migration mechanism based on container in edge computing. *J. Parallel Distrib. Comput.* **152**, 157–166 (2021).
34. Cui, Y. et al. Multi-user reinforcement learning based task migration in mobile edge computing. *Front. Comput. Sci.* **18** (2024).
35. Haris, R. M., Khan, K. M., Nhlabatsi, A. & Barhamgi, M. A machine learning-based optimization approach for pre-copy live virtual machine migration. *Clust. Comput.* **27**, 1293–1312 (2024).
36. Miao, Y., Wu, G., Li, M., Ghoneim, A. & Hossain, M. S. Intelligent task prediction and computation offloading based on mobile-edge cloud computing. *Future Gener. Comput. Syst.* **102**, 925–931 (2020).

37. Cao, J., Yu, Z. & Xue, B. Research on collaborative edge network service migration strategy based on crowd clustering. *Sci. Rep.* **14**, 7207 (2024).
38. Liu, S. et al. Dependent task scheduling and offloading for minimizing deadline violation ratio in mobile edge computing networks. *IEEE J. Sel. Areas Commun.* **41**, 538–554 (2023).
39. Tang, Z., Zhou, X., Zhang, F., Jia, W. & Zhao, W. Migration modeling and learning algorithms for containers in fog computing. *IEEE Trans. Serv. Comput.* **12**, 712–725 (2018).
40. Wang, Z. et al. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning*. 1995–2003 (2016).
41. Fortunato, M. et al. Noisy networks for exploration. In *International Conference on Learning Representations* (2018).
42. van Hasselt, H., Guez, A. & Silver, D. Deep reinforcement learning with double q-learning. In *AAAI Conference on Artificial Intelligence*. 2094–2100 (2016).
43. Horgan, D. et al. Distributed prioritized experience replay. In *International Conference on Learning Representations* (2018).
44. Wu, H., Yang, X. & Bu, Z. Task offloading with service migration for satellite edge computing: A deep reinforcement learning approach. *IEEE Access* **12**, 25844–25856 (2024).
45. Chi, J., Zhou, X., Xiao, F., Lim, Y. & Qiu, T. Task offloading via prioritized experience-based double dueling DQN in edge-assisted IIOT. *IEEE Trans. Mob. Comput.* **23**, 14575–14591 (2024).
46. Tang, M. & Wong, V. W. S. Deep reinforcement learning for task offloading in mobile edge computing systems. *IEEE Trans. Mobile Comput.* **21**, 1985–1997 (2022).
47. Wang, Y., Ru, Z., Wang, K. & Huang, P. Joint deployment and task scheduling optimization for large-scale mobile users in multi-UAV-enabled mobile edge computing. *IEEE Trans. Cybern.* **50**, 3984–3997 (2020).

Author contributions

Conceptualization, L.L. and J.L.; methodology, L.L. and J.L.; software, J.L.; validation, S.W., C.H. and G.S.; formal analysis, L.L., N.L. and G.S.; investigation, S.W.; resources, C.H.; data curation, J.L; writing–original draft preparation, J.L. and L.L.; writing–review and editing, L.L. and N.L.; funding acquisition, L.L. and N.L.. All authors have read and agreed to the published version of the manuscript.

Funding

This research was funded by the Technology Innovation Guidance Project of Jilin Province, China (Grant No. 20240402083GH).

Declarations

Competing interests

The authors declare no competing interests.

Additional information

Correspondence and requests for materials should be addressed to N.L.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.