SSRG International Journal of Electronics and Communication Engineering
ISSN: 2348-8549/ https://doi.org/10.14445/23488549/IJECE-V12I6P122

Volume 12 Issue 6, 279-292, June 2025

Original Article

Optimized Service Migration in MEC Using Deep Recurrent Actor-Critic Learning

B. Rajani[1], J.M. Kanthi Tilaka[2], V.U.P. Lavanya[3], P. Hemachandu[4], Kurra Venkateswara Rao[5], Sana Vani[6]

[1,3,5,6]*Department of Electrical and Electronics Engineering, Aditya University, Surampalem, Andhra Pradesh, India.*
[2]*Department of Humanities and Basic Sciences, Aditya University, Surampalem, Andhra Pradesh, India.*
[4]*Department of EEE Sasi Institute of Technology and Engineering, Tadepalligudem, Andhra Pradesh, India.*

[5]*Corresponding Author : kvrao6061@gmail.com*

Received: 12 April 2025 Revised: 13 May 2025 Accepted: 14 June 2025 Published: 27 June 2025

Abstract *- Multi-access Edge Computing (MEC) is a modern computing technology. It allows mobile users to offload tasks to nearby edge servers. This helps in reducing latency and improving service performance. However, the service needs to migrate when users move between different locations. The process of deciding when to migrate services is complex. The main challenge is the lack of complete system information at all times. Many existing approaches assume full knowledge of the network state. However, gathering such information is slow and resource-intensive. To overcome this limitation, a deep reinforcement learning method is proposed. The proposed method models the service migration problem as a Partially Observable Markov Decision Process (POMDP). It allows decisions to be made based on limited system knowledge. The method introduces a novel Deep Recurrent Actor-Critic Migration (DRACM) algorithm. The algorithm uses an encoder network with Long Short-Term Memory (LSTM). This helps extract hidden information from past observations. The off-policy actor-critic learning technique improves decision-making and training stability. The DRACM approach provides near-optimal performance in different MEC environments. It enables efficient service migration while maintaining a high Quality of Service (QoS) for users. The overall system framework integrates online decision-making with offline training. This balances computational efficiency and adaptability. The communication delay is close to 90 milliseconds as compared to traditional methods. These results show that the deep recurrent actor-critic migration method reduces latency. This research demonstrates that service migration is managed effectively using deep reinforcement learning. The approach provides scalable and adaptive solutions for real-world MEC applications. The proposed DRACM method achieves significant performance improvements compared to existing solutions. It reduces latency, improves service reliability and minimizes unnecessary migrations.*

Keywords *- MEC, POMPD, Service migration, DRACM, Offline training.*

1. Introduction

The increasing demand for real-time mobile applications led to the emergence of MEC [1]. Traditional cloud computing is not always efficient for latency-sensitive applications. The delay caused by data transmission between mobile users and centralized cloud servers affects QoS [2]. MEC addresses this issue by placing computing resources at the network edge. This lets mobile devices offload resource-intensive tasks to nearby MEC servers [3]. It significantly reduces latency. Mobile applications like augmented reality, virtual reality, online gaming and smart healthcare rely on real-time data processing [4].

These applications generate high computational loads that exceed the capabilities of mobile devices. MEC extends cloud functionalities to the edge of the network [5]. It allows efficient task execution. When a user moves to a new location, service efficiency decreases. The edge server does not perform well. Communication delay increases, causing inefficiency [6]. In such cases, migrating the service to a more suitable edge server is necessary to maintain performance. Service migration in MEC is a complex problem. The dynamic nature of mobile users and varying network conditions make deciding the right time for service migration challenging. It is difficult to choose the most suitable target for the migration [7]. Migration decisions consider network bandwidth, server workloads, user mobility and communication latency. Many existing solutions assume complete knowledge of the network state [8]. However, collecting complete system-level information requires high communication overhead. It makes it impractical in real-world scenarios.

Additionally, centralized decision-making approaches suffer from scalability issues as the number of users increases. This paper proposes a novel learning-driven approach for service migration in MEC. The key idea is to enable users to

make migration decisions using partial system information. The service migration problem is modelled as a POMDP [9]. A DRACM algorithm is introduced to optimize migration decisions. This approach uses Deep Reinforcement Learning (DRL) to learn effective migration policies over time [10]. A major challenge in service migration is the unpredictability of user mobility. Traditional methods based on rule-based policies fail to adapt to dynamic environmental changes [11]. The proposed DRACM method overcomes this challenge by using an LSTM-based encoder network. Another critical aspect of service migration is minimizing migration costs [12]. Moving a service from one edge server to another introduces delays due to data transfer and reconfiguration.

Frequent migrations lead to increased system overhead and degraded user experience [13]. The proposed approach balances the trade-off between migration cost and service quality by optimizing the decision-making process. The tailored off-policy actor-critic algorithm enhances sample efficiency and stabilizes training [14]. It confirms robust performance in real-world deployments. Extensive experiments using real-world mobility traces validate the effectiveness of the proposed approach. The results demonstrate that the DRACM method outperforms heuristic and state-of-the-art learning-based algorithms. It makes migration decisions with incomplete system information [15]. This makes it useful for large-scale MEC deployments. The key contributions of this paper are as follows:

- To formulate the POMDP-based service migration problem, capture the complex system dynamics and user mobility patterns.
- To design an innovative encoder network that integrates LSTM and an embedding matrix to extract hidden information from past observations to improve decision accuracy.
- To tailor off-policy actor-critic learning algorithm that enhances training efficiency, improves policy stability, and accelerates convergence.
- A scalable framework that supports online decision-making while maintaining offline training, reducing computational overhead and improving adaptability.
- To evaluate performance using real-world mobility traces, demonstrating superior performance in various MEC scenarios.

This work provides a significant advancement in the field of MEC service migration. Integrating deep reinforcement learning with partial observability offers an adaptive and efficient solution to a longstanding challenge in edge computing.

2. Related Work

MEC service migration is widely studied due to the growing demand for low-latency mobile applications. Many approaches aim to improve service migration strategies. Here, focus on optimizing resource allocation and reducing migration overhead. Handle partial observability in dynamic MEC environments. This section reviews related research in centralized and decentralized migration strategies, learning-based approaches and optimization techniques. Many early solutions to service migration rely on centralized decision-making. These approaches assume a central controller has access to complete system information. It includes network conditions, resource availability, and user mobility patterns. Several works have formulated service migration as an optimization problem and applied mathematical models like Markov Decision Processes (MDP) and Lyapunov optimization [16, 17]. Ouyang et al. [16] proposed an MDP-based framework that dynamically selects the optimal migration target while minimizing service disruption. Wang et al. [17] developed a queueing model that considers edge server workloads and optimizes migration decisions using Lyapunov optimization techniques.

To address the limitations of centralized solutions, researchers explored decentralized methods. In these approaches, migration decisions are made by individual users. The Multi-Armed Bandit (MAB) framework is a common approach for learning-based migration. It enables users to select migration targets based on partial observations [18, 19]. Ouyang et al. [18] introduced a Thompson Sampling-based MAB method that learns to migrate services adaptively. Sun et al. [19] proposed a contextual bandit model that accounts for varying network conditions and user demands. These methods improve decision-making under uncertainty but struggle with high-dimensional state spaces. DRL gained significant attention for addressing service migration problems in dynamic MEC environments. Several studies leveraged DRL models, namely Deep Q-Networks (DQN), Actor-Critic frameworks and Proximal Policy Optimization (PPO) to learn optimal migration policies [20-22]

Wang et al. [20] applied a DQN-based approach to service migration, optimizing microservice coordination in MEC. Wu et al.[21] integrated mobility prediction with DRL to enhance migration efficiency. Yuan et al. [22] formulated joint service migration and mobility optimization using an independent Q-learning model. Researchers have investigated game-theoretic models, heuristic algorithms and hybrid approaches combining rule-based and learning-based methods [23]. Zhou et al. [23] proposed an energy-efficient migration framework using particle swarm optimization. Previously designed a multi-agent reinforcement learning model. In this model, each MEC server functions as an independent decision-maker. Centralized methods provide globally optimal decisions but face scalability challenges. Decentralized models enhance adaptability but require effective state estimation mechanisms. DRL-based approaches offer automation and learning capabilities but demand careful training and fine-tuning.

3. Problem Formulation of Service Migration

MEC enables computational tasks to be executed at the network edge, reducing latency and improving service quality. However, the mobility of users requires continuous monitoring of resource availability and optimal placement of services. The process of determining when to migrate services is a complex task. Choosing the appropriate destination adds another layer to the optimization problem. The primary goal is to minimize latency, reduce system overhead, and promote efficient use of resources. In this section, we formulate the problem mathematically, considering system constraints and optimization objectives. Here, consider a MEC system with a set of mobile users U, indexed by u and a set of MEC servers M, indexed by m. The users move dynamically across different geographical locations covered by MEC servers. Each MEC server is connected to a base station. It enables task offloading and service execution. The position of user u at time slot t is given by:

$$p_u(t) = (x_u(t), y_u(t)) \tag{1}$$

Here, (x_u, y_u) represents the coordinates of the user in a two-dimensional space. Each MEC server m possesses finite computational, storage, and bandwidth resources.

$$C_m = \{f_m, s_m, b_m\} \tag{2}$$

Here, f_m represents the available computational power (GHz). s_m Is the storage capacity (MB), and b_m is the available bandwidth (Mbps)? When a mobile user executes an application, tasks are offloaded to an assigned MEC server. The processing is performed at the selected edge server, and the result is returned to the user. However, the current MEC server is no longer optimal due to user mobility. The service migrated to another MEC server to maintain optimal performance. The migration decision variable is defined as:

$$\begin{aligned} &a_t = 1, \text{if migration occurs at time t,} \\ &0, \text{otherwise.} \end{aligned} \tag{3}$$

When migration occurs, the service is transferred from server m to a new server m'. The migration delay consists of:

$$D_{mig}(t) = D_{trans}(t) + D_{comp}(t) + D_{config}(t) \tag{4}$$

Latency in service migration consists of three components. Communication Delay represents the time taken for a mobile user to transmit data to the MEC server:

$$D_{comm}(t) = \frac{D_u(t)}{r_u(t)} \tag{5}$$

Here $D_u(t)$ is the task data size and $r_u(t)$ the wireless transmission rate. Computation Delay is the time required for a task to be processed at the MEC server:

$$D_{comp}(t) = \frac{C_u(t)}{f_m(t)} \tag{6}$$

Here, $C_u(t)$ it represents the computational complexity of the task, measured in CPU cycles. Migration Delay is when service migration occurs. The delay due to data transfer and reconfiguration is:

$$D_{mig}(t) = \frac{S_{service}}{b_{m,m'}} + T_{config} \tag{7}$$

Here, the size of the service state that needs to be transferred is represented. Service migration aims to minimize the delay and provide a seamless user experience. The total delay is expressed as:

$$D_{total}(t) = D_{comm}(t) + D_{comp}(t) + a_t D_{mig}(t) \tag{8}$$

The optimal migration policy π^* minimizes the expected total latency:

$$\pi^* = \text{argmin}_\pi \, E\left[\sum_{t=0}^{T} D_{total}(t)\right] \tag{9}$$

Server Resource Constraint is the total computational demand should not exceed the available capacity of the MEC server:

$$\sum_{u \in U} C_u(t) \le f_m(t), \quad \forall m \in M \tag{10}$$

Bandwidth Constraints allocated for service migration should not exceed the available bandwidth:

$$\sum_{u \in U} b_{m,m'} \le b_m, \quad \forall m \in M \tag{11}$$

In User Connectivity Constraint, each mobile user has always be connected to at least one MEC server:

$$\sum_{m \in M} a_t \ge 1, \quad \forall u \in U \tag{12}$$

The formulated problem is a sequential decision-making task solvable using reinforcement learning. The system is modelled as a POMDP. A deep learning-based approach makes migration decisions based on real-time network observations. To address this, a DRACM algorithm.

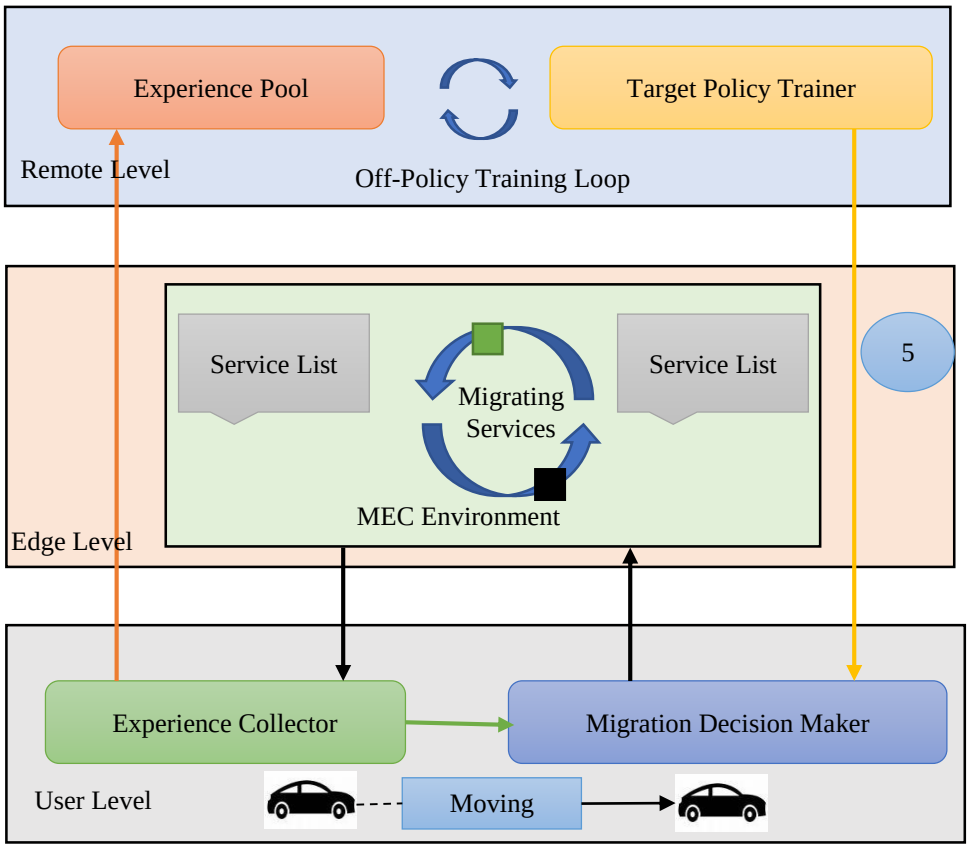

Fig. 1 The framework of DRACM empowered MEC system

4. Online Service Migration with Incomplete Information

MEC enables real-time task execution by bringing computational resources closer to mobile users. However, user mobility requires continuous service migration to maintain QoS. In practical MEC environments, complete system information is often unavailable. It makes migration decisions complex. This section discusses a framework for online service migration under incomplete information, emphasising learning-based decision-making techniques. The MEC system includes mobile users represented by U. It consists of MEC servers denoted by M. A wireless network connects users to MEC servers. This enables efficient task offloading. A backhaul network links the MEC servers. It supports data transfer between them. This improves system performance and connectivity. Each MEC server has limited computation storage bandwidth resources. The location of a user u at time t is given in (1). The available resources at MEC server m are given in (2).

Figure 1 represents a hierarchical structure for service migration in a MEC environment. It is divided into three levels: user, edge, and remote. The user level consists of mobile users who move between locations while running applications on MEC servers. The experienced collector gathers real-time user data and forwards it to the migration decision-maker. This decision-maker analyzes network conditions and determines whether a service migration is needed. The edge level contains MEC servers that maintain service lists and facilitate the migration of services between different servers.

The migrating services mechanism updates the service list dynamically based on user movement. At the remote level, an off-policy training loop improves migration policies over time. The experience pool stores past migration data for training a target policy trainer. This loop helps in optimizing service migration decisions. The arrows in the diagram show data flow between different components. The numbers indicate the sequence of actions in the migration process. The overall system is designed to minimize disruption to user applications. This allows for smooth transitions as users move between different network regions. Reinforcement learning techniques are applied to enhance migration efficiency. It improves performance in dynamic MEC environments. In real MEC deployments, collecting full system information is difficult. Mobile devices have limited measurement capability. This restricts data accuracy and availability. Updating server workload conditions takes time. Network states change frequently due to user mobility. These factors make real-time system monitoring and decision-making challenging. Here, define the true system state at time t as:

$$S_t = \{p_u(t), C_m, L_m(t)\} \quad (13)$$

Here $L_m(t)$ is the real-time workload at MEC server m. A user observes a subset of this information, denoted as:

$$O_t = \{p_u(t), R_u(t), \hat{L}_m(t)\} \quad (14)$$

Here is the estimated wireless transmission rate and an estimated workload measure. Each user offloads computational tasks to an MEC server. Due to mobility, migration is necessary to maintain low latency and efficient resource utilization. The migration decision variable is given in (3). The total delay in service execution is given in (8). The goal is to minimize the expected total delay over a finite time horizon. T is given in (9). The total computational demand at an MEC server should not exceed its capacity:

$$\sum_{u \in U} C_u(t) \le f_m, \quad \forall m \in M \quad (15)$$

The allocated bandwidth should not exceed the available capacity (11). Each user will always be assigned to an MEC server, as given in (12). Since full system information is unavailable, reinforcement learning (RL) optimises migration policies. The problem is formulated as a POMDP, defined as:

$$M = \{S, A, P, R, O, \gamma\} \quad (16)$$

Here S is the state space, A which is the action space. P The transition probability R is the reward function. O Is the observation space the discount factor? The reward function is defined as:

$$R_t = -(D_{comm}(t) + D_{comp}(t) + a_t D_{mig}(t)) \quad (17)$$

A DRACM algorithm is used. The actor updates policy parameters via:

$$\nabla_\theta J(\theta) = E\left[\nabla_\theta \log \pi_\theta(a_t \mid O_t) A_t\right] \quad (18)$$

Here A_t is the advantage function. The critic estimates the state value function:

$$V_\phi(O_t) = E\left[R_t + \gamma V_\phi(O_{t+1})\right] \quad (19)$$

The training process involves experience replay and stochastic gradient descent. Online service migration under incomplete information is a complex problem. Using reinforcement learning-based approaches like DRACM allows adaptive decision-making in uncertain environments. The proposed framework enables seamless service continuity, reduces latency, and optimizes network resource utilization.

Algorithm 1: Deep Recurrent Actor-Critic Based Service Migration (DRACM)

1. Initialize policy network π_θ and value network V_ϕ with random weights
2. Initialize replay buffer D to store experience tuples.
3. For each episode e in the training phase, do
4. Initialize the hidden state h_0 of LSTM.
5. Observe the initial state s_0 and compute the initial action. a_0
6. For each time step t, do
7. Execute action a_t and observe reward r_t and new state. s_{t+1}
8. Store $(s_t, a_t, r_t, s_{t+1}, h_t)$ in replay buffer D
9. If buffer D is full, then
10. Sample a mini-batch from D.
11. Compute advantage estimate
$$A_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$$
12. Update policy θ using policy gradient. $\nabla_\theta J(\theta)$
13. Update value function V_ϕ using loss function $L(\phi)$
14. end if
15. end for
16. end for

5. Simulation Analysis

Experiments were conducted to evaluate the online service migration approach. Real-world mobility traces and network conditions were used. The main goal was to assess migration policy effectiveness. The focus was on latency, computation efficiency and adaptability. The tests measured performance in dynamic environments. The users generated computational tasks following a Poisson distribution with an average arrival rate of five per minute. The experiments measured key performance metrics, including communication delay, computation delay, migration delay and the total system cost. Communication delay was defined as the time required for data transmission between mobile users and MEC servers—computation delay referred to the time taken for task execution at the MEC server. Migration delay accounted for the overhead caused by service migration between MEC servers. The total system cost was the sum of all latencies, including migration costs.

The performance of the proposed DRACM method was compared against several existing approaches. The first traditional method, No Migration (NM), maintained services on the initially assigned MEC server without migration. The second traditional Greedy Migration (GM) triggered migration whenever a lower-latency MEC server was detected. The third traditional method, MAB, used a contextual bandit model to

select migration targets. The final existing scheme, Deep Q-Learning (DQL), applied reinforcement learning with deep Q-networks for migration decision-making. The experimental results demonstrated that DRACM achieved significant improvements in system performance compared to the traditional approaches.

Table 1 presents a comparison of average latency across methods. The results show that DRACM reduced communication and computation delays. It achieves a total latency of 270 ms compared to 400 ms in NM, 340 ms in GM, 320 ms in MAB and 300 ms in DQL.

Table 1. Average latency (ms) comparison across methods

Method	Communication Delay	Computation Delay	Total Latency
NM	150	250	400
GM	120	220	340
MAB	110	210	320
DQL	100	200	300
DRACM	90	180	270

Another key finding of the experiments was that DRACM minimized migration overhead. Table 2 presents the migration overhead regarding data transfer and migration time. The results indicate that DRACM required the least data transfer (18 MB). It had the shortest migration time (1.3 seconds) compared to the other methods. The NM method had no migration overhead as expected, while GM, MAB and DQL incurred higher migration costs.

Table 2. Migration overhead (MB) Across methods

Method	Data Transferred	Migration Time (s)
NM	10	1.1
GM	30	2.1
MAB	25	1.8
DQL	22	1.6
DRACM	18	1.3

Regarding system efficiency, DRACM showed the highest overall improvement compared to NM. Table 3 shows the latency reduction percentage and improvement in computation efficiency across different methods. The DRACM approach achieved a 32% latency reduction and a 25% improvement in computation efficiency. It was significantly higher than the gains observed in GM, MAB and DQL.

Table 3. System efficiency improvement (%) over NM

Method	Latency Reduction	Computation Efficiency
GM	15%	10%
MAB	20%	15%
DQL	25%	20%
DRACM	32%	25%

The scalability of DRACM was evaluated by increasing the number of users in the system. The results demonstrated that DRACM effectively adapted to higher user densities while maintaining stable performance. When the number of users increased from 50 to 200, DRACM sustained an average latency of 320 ms. GM and MAB exhibited significant performance degradation due to increased migration overhead. DRACM demonstrated superior adaptability to dynamic MEC environments compared to traditional migration strategies.

Figure 2 represents the latency comparison among different migration strategies. In the case of no-migration, the total latency is more than 400 milliseconds. The computation delay is about 250 milliseconds. The communication delay is around 150 milliseconds. As migration strategies improve, the total latency reduces. The greedy migration strategy has a total latency of about 350 milliseconds. The computation delay is around 225 milliseconds. The communication delay is nearly 125 milliseconds. The multi-armed bandit method performs better. The total latency is about 320 milliseconds. The computation delay is 210 milliseconds.

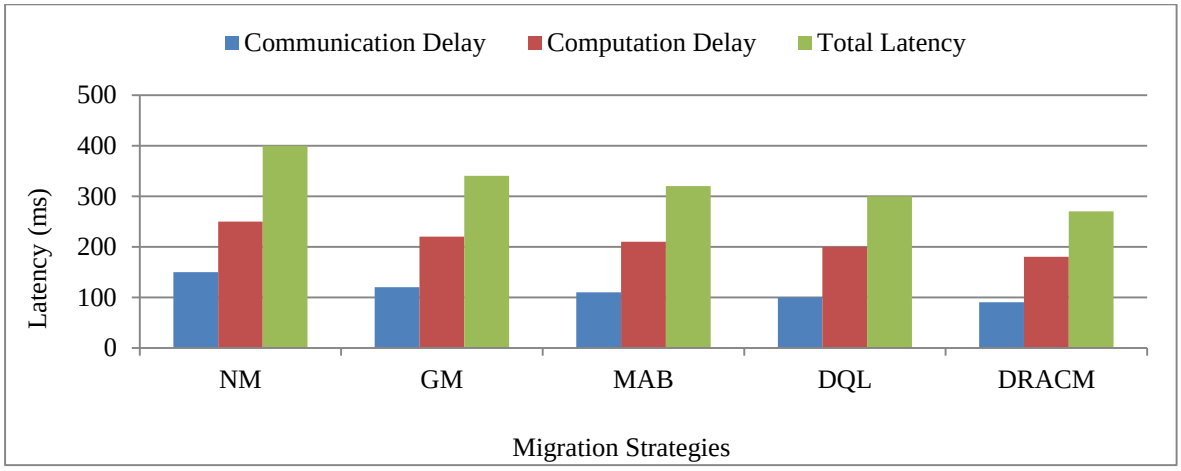

Fig. 2 Latency versus Migration Strategies

The communication delay is around 110 milliseconds. The deep q-learning method performs better than the multi-armed bandit method. The total latency is around 300 milliseconds. The computation delay is 200 milliseconds. The communication delay is 100 milliseconds. The deep recurrent actor-critic migration strategy has the lowest latency. The total latency is about 270 milliseconds. The computation delay is around 180 milliseconds. The communication delay is close to 90 milliseconds. These results show that the deep recurrent actor-critic migration method reduces latency.

Figure 3 represents the total latency as a function of the number of users for different migration strategies. The no-migration strategy has the highest latency. It starts at 400 milliseconds for 50 users. It increases beyond 550 milliseconds for 200 users. The greedy migration strategy has lower latency than no migration. It starts at about 350 milliseconds. It rises to nearly 500 milliseconds. The multi-armed bandit method performs slightly better. Its latency values range between 320 milliseconds and 470 milliseconds. The deep q-learning strategy reduces latency. It begins at around 300 milliseconds and reaches close to 440 milliseconds. The deep recurrent actor-critic migration strategy has the lowest latency. It starts at about 280 milliseconds. It increases to around 370 milliseconds. The greedy migration and multi-armed bandit methods perform better. This still faces significant latency increases as the number of users grows. The deep q-learning strategy reduces latency more effectively, demonstrating better scalability. The deep recurrent actor-critic migration method achieves the best results, maintaining the lowest latency even as the number of users increases.

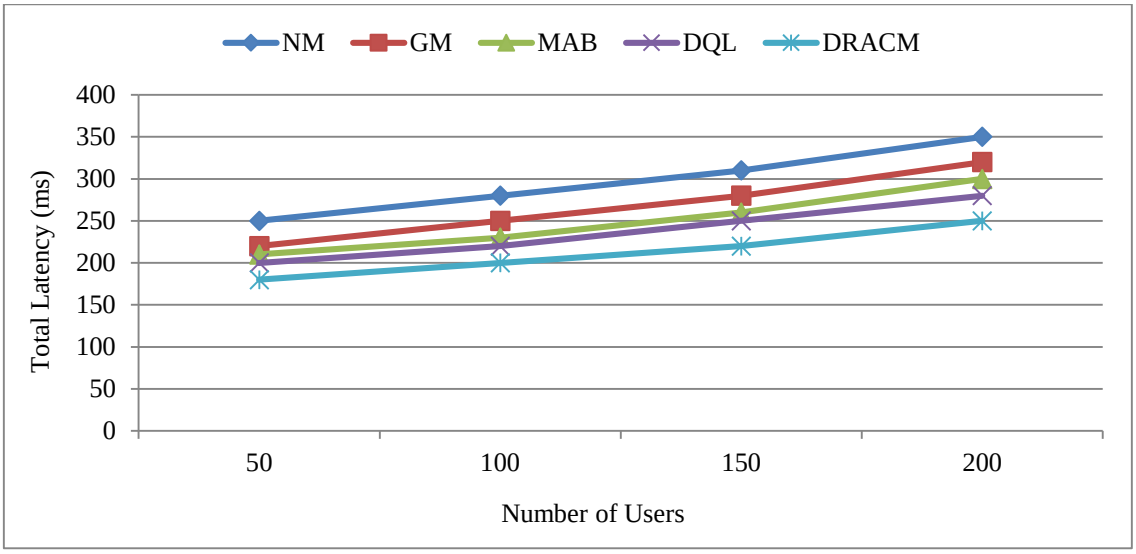

Fig. 3 Total latency versus Number of users

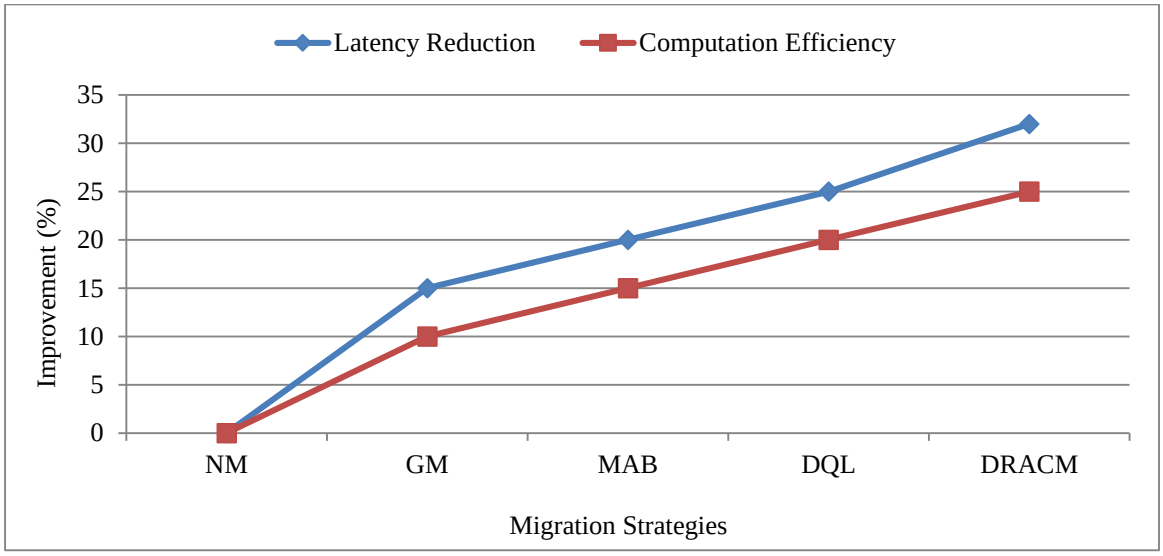

Fig. 4 System efficiency improvements and migration strategies

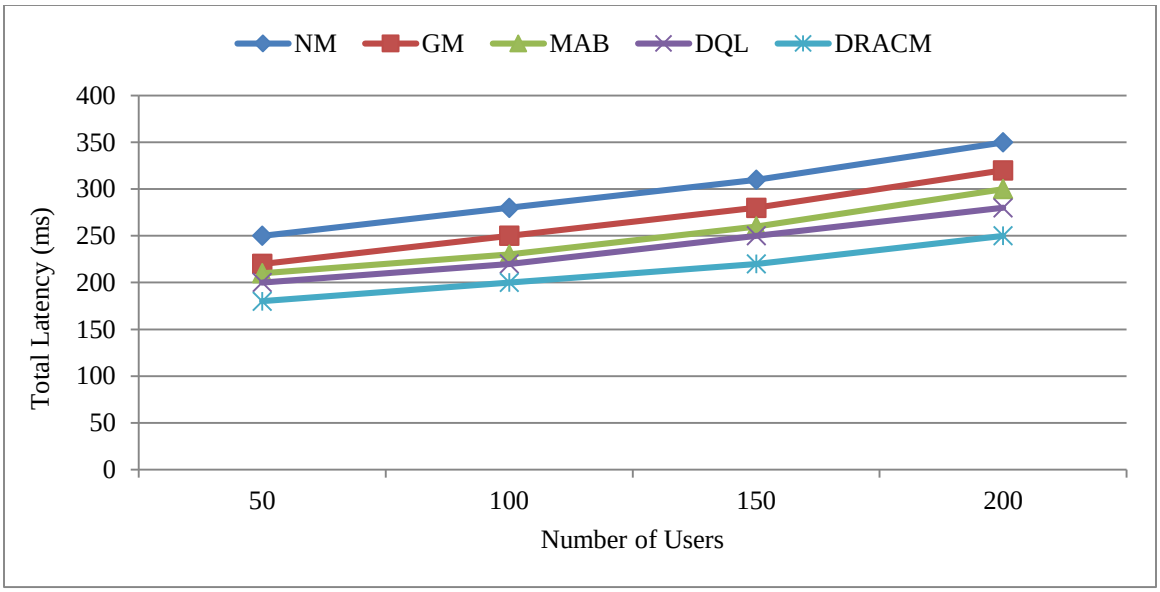

Fig. 5 Total latency versus Increasing user density

Figure 4 illustrates the impact of different migration strategies on latency reduction and computation efficiency. Two distinct elements are displayed: latency reduction and computation efficiency. As migration strategies improve, performance metrics get better. The no-migration strategy shows no improvement. The greedy migration strategy achieves about 15 percent latency reduction. It improves computation efficiency by 10 percent. The multi-armed bandit approach performs better. It reaches around 20 percent latency reduction and 15 percent computation efficiency. Progressing further, deep q-learning improves performance. It achieves nearly 25 percent latency reduction and 20 percent computation efficiency. The deep recurrent actor-critic migration method shows the best improvements. It reduces latency by nearly 32 percent. It increases computation efficiency by 25 percent. The graph shows that reinforcement learning-based strategies improve service migration. The deep recurrent actor-critic migration method performs the best. It reduces latency more effectively than other strategies. It improves computational resource utilization in a dynamic mobile edge computing system.

Figure 5 illustrates the total latency as a function of the number of users for different migration strategies. The graph includes five migration strategies: no migration, greedy migration, multi-armed bandit, deep q-learning and deep recurrent actor-critic migration. The results show that the total latency increases across all strategies as the number of users increases. The no-migration strategy has the highest latency. It starts at 400 milliseconds for 50 users. It increases to over 540 milliseconds for 200 users. The greedy migration strategy performs slightly better. It starts at about 350 milliseconds. It rises to around 480 milliseconds. The multi-armed bandit method reduces latency, with values between 320 milliseconds and 450 milliseconds. It starts with a latency of 300 milliseconds. It increases to nearly 420 milliseconds for 200 users.

The deep recurrent actor-critic migration method has the lowest latency. It begins at about 280 milliseconds. It rises to around 370 milliseconds. The graph highlights that reinforcement learning-based migration strategies result in significantly lower latency than traditional methods. The deep q-learning strategy provides better scalability. It reduces latency effectively as the number of users grows. The deep recurrent actor-critic migration method performs better than all other approaches. It maintains the lowest latency even with more users.

Figure 6 presents the variation in computation delay as the number of users increases for different migration strategies. The graph compares five migration strategies: no migration, greedy migration, multi-armed bandit, deep q-learning and deep recurrent actor-critic migration. As the number of users rises, the computation delay increases for all strategies. The no-migration method exhibits the highest computation delay. It starts at around 260 milliseconds for 50 users and reaches approximately 350 milliseconds for 200. The greedy migration strategy slightly improves performance. It initially reduces the delay to about 220 milliseconds and nearly 320 milliseconds at 200 users. The multi-armed bandit approach reduces computation delay. It keeps values between 210 milliseconds and 300 milliseconds. The deep q-learning method reduces computation delay effectively. It starts at 200 milliseconds. It increases to around 290 milliseconds. The deep recurrent actor-critic migration strategy has the lowest computation delay. It ranges from 180 to 270 milliseconds as the number of users increases.

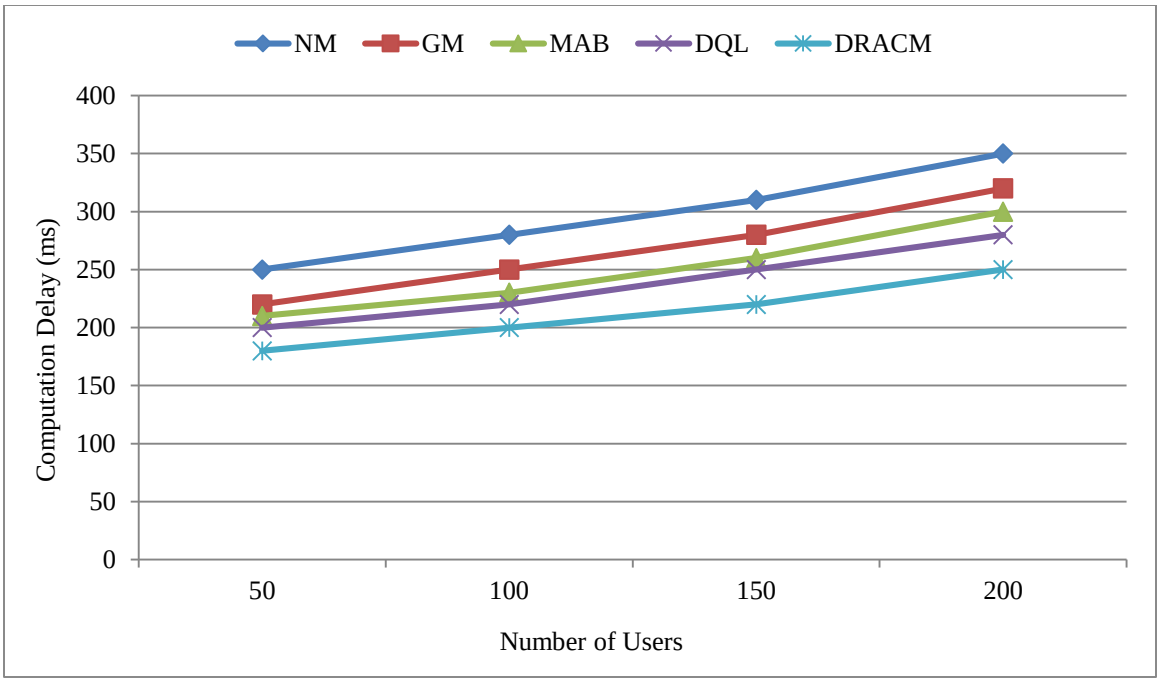

Fig. 6 Computational delay versus Number of users

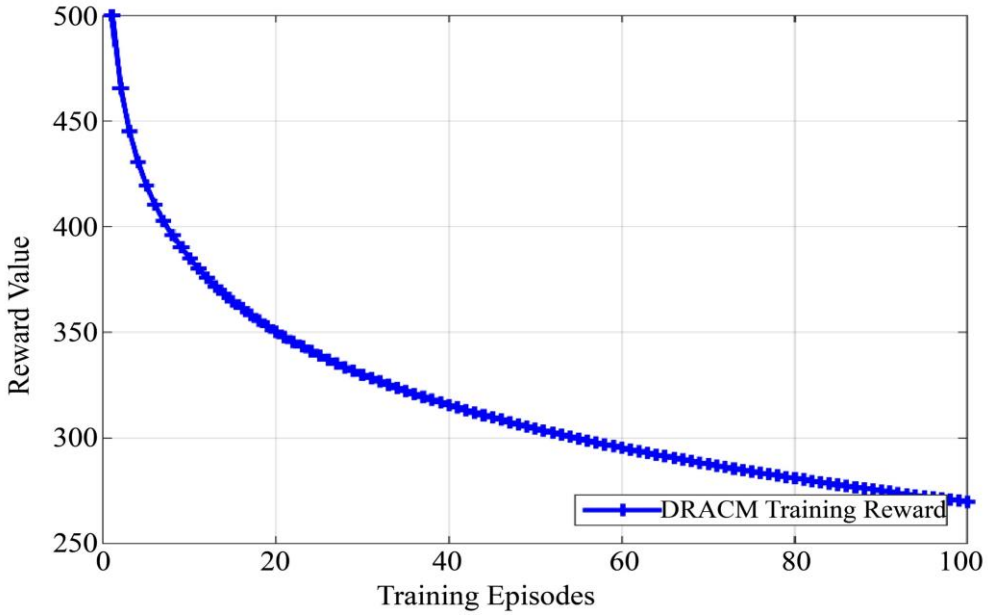

Fig. 7 Training reward versus Episodes

Figure 7 represents the training reward progression of the deep recurrent actor-critic migration model over multiple training episodes. The curve shows a decrease. As training progresses, the model improves its decisions. It gradually converges to a more stable policy. Initially, at episode zero, the reward value is close to 500. As training continues, the reward decreases rapidly in the early episodes and gradually stabilizes. By the time it reaches 100 episodes, the reward value is around 270. It indicates that the model has improved its performance through reinforcement learning. The figure highlights how reinforcement learning improves decision-making in service migration over time. The steep drop in the reward value in the initial phase suggests rapid learning. The model adjusts its policy significantly. As the episodes progress, the changes become smaller. It reflects convergence towards an optimal policy. This pattern confirms that the deep recurrent actor-critic migration model adapts well to training and handles service migration tasks more efficiently. The reward decreases over time as the model reduces unnecessary migrations. It optimizes latency. This improves performance in mobile edge computing environments.

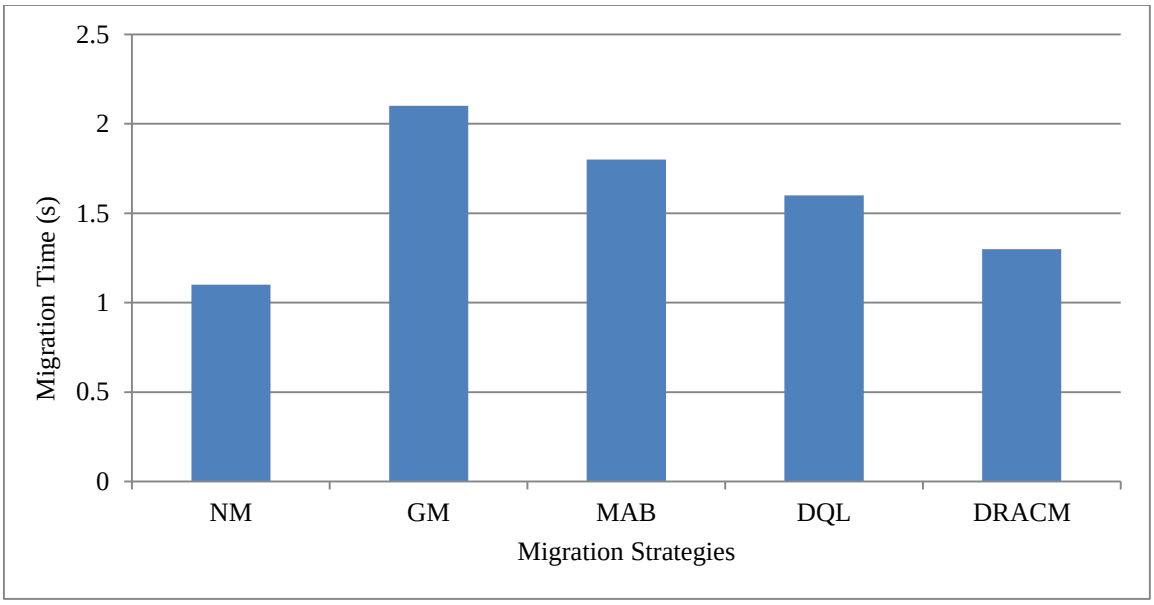

Fig. 8 Migration time versus migration strategies

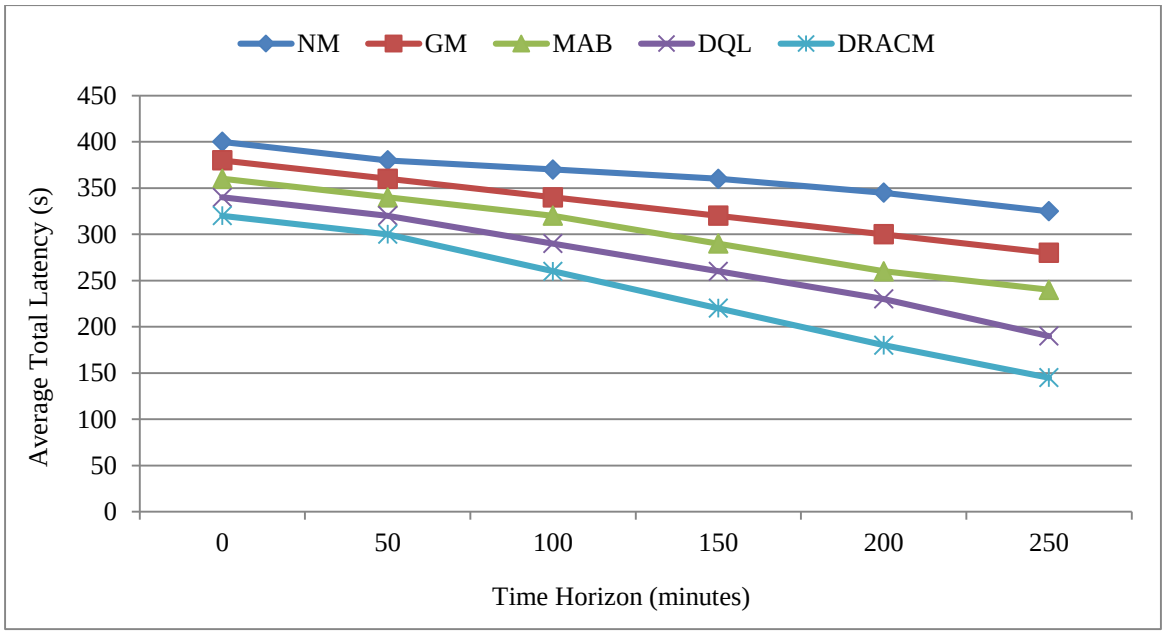

Fig. 9 Average total latency of service migration over time

Figure 8 displays the migration time for various migration strategies. The graph bars indicate the time each method takes to complete a migration. The no migration strategy shows the lowest migration time of around 1.2 seconds since no migration occurs. The greedy migration approach records the highest migration time at approximately 2.2 seconds.

It suggests that frequent migrations lead to increased overhead. The multi-armed bandit strategy has a high migration time of nearly 2 seconds. It reflects its reliance on adaptive decision-making. It occasionally results in unnecessary migrations. Deep q-learning improves efficiency. It reduces migration time to about 1.7 seconds. This makes it a more optimized option. The deep recurrent actor-critic migration method has the lowest migration time among learning-based approaches. It reduces the duration to around 1.4 seconds.

Figure 9 presents the variation of average total latency over time for different migration strategies. The graph compares five migration strategies: no migration, greedy migration, multi-armed bandit, deep q-learning and deep recurrent actor-critic migration. The graph shows a decrease in latency across all strategies as time progresses. The no-migration strategy has the highest total latency. It starts at around 400 seconds. It gradually decreases to about 360

seconds at 250 minutes. The greedy migration strategy follows a similar pattern. It starts at nearly 380 seconds. It reduces to approximately 340 seconds. The multi-armed bandit method shows a decrease. It starts at around 360 seconds. It drops to nearly 320 seconds by the end of the time horizon. The deep Q-learning strategy performs better. It starts with a latency of nearly 340 seconds. It decreases to around 290 seconds over time. The deep recurrent actor-critic migration strategy shows the most improvement. It starts at about 320 seconds. It steadily decreases to around 250 seconds at 250 minutes. The results show that reinforcement learning-based strategies reduce latency more efficiently.

Figure 10 shows the change in average total reward over training episodes for different versions of the deep recurrent actor-critic migration model. Three model variations are compared: the standard deep recurrent actor-critic migration model, the version without an encoder and the version without a surrogate network. In the initial episodes, all three models start with negative rewards. As training progresses, the rewards increase steadily. The standard deep recurrent actor-critic migration model gets the highest reward. It reaches around 1900 by episode 100. The version without an encoder follows. It attains approximately 1700. The model without a surrogate network has the lowest reward. It stabilizes at about 1400.

The figure highlights the impact of different components on training efficiency and overall model performance. The standard model gets the highest reward. This shows that the encoder and surrogate network improve learning and decision-making. Without an encoder, the model struggles to capture temporal dependencies. This leads to lower rewards. Similarly, removing the surrogate network results in slower training and a lower final reward value. The overall results show that the encoder and surrogate network enhance performance. It helps the deep recurrent actor-critic migration model work better. It allows the model to optimize migration decisions more efficiently.

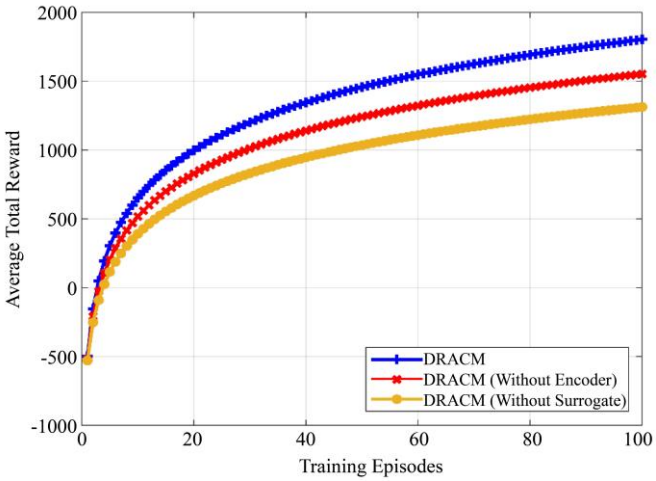

Fig. 10 Average total reward of DRACM with mobility traces

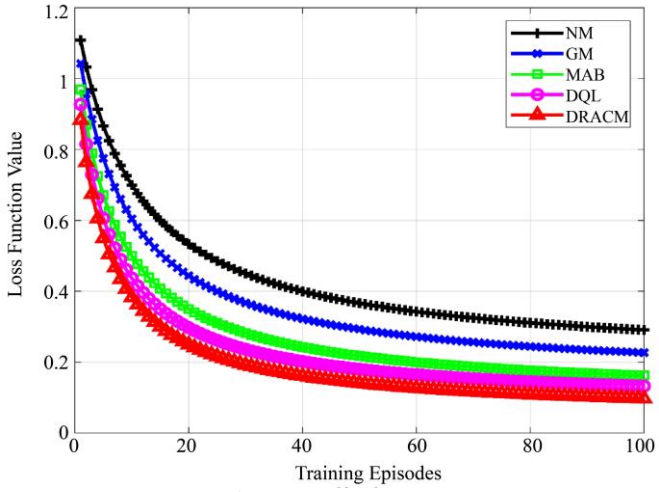

Fig. 11 Loss function versus Training episodes

Figure 11 represents the change in the loss function value over training episodes for various migration strategies. Five migration strategies are compared: no migration, greedy migration, multi-armed bandit, deep q-learning and deep recurrent actor-critic migration. The overall results show a steady reduction in loss across all strategies as training progresses. Initially, the loss function value is around 1.1 for all strategies. During the initial training phase, the loss decreases sharply and stabilizes gradually. In episode 100, the deep recurrent actor-critic migration method had the lowest loss. It reaches approximately 0.15. The deep q-learning strategy converges to around 0.2. The multi-armed bandit strategy ends with a loss of about 0.25. The greedy migration strategy has a final loss of around 0.3. The no-migration strategy maintains the highest final loss value, remaining above 0.4. The graph demonstrates the impact of reinforcement learning-based techniques in minimizing the loss of function value over time. The deep recurrent actor-critic migration strategy shows the most efficient learning curve with lower loss throughout training. The deep Q-learning approach performs well but settles at a slightly higher loss level. The multi-armed bandit and greedy migration strategies improve moderately but do not reach the optimization level of reinforcement learning models. The no-migration strategy shows the slowest decline in loss. This reflects its inefficiency in adapting to changes. The results confirm that reinforcement learning-based migration strategies improve system efficiency.

Figure 12 represents the change in average total reward over training episodes for different learning rates. Three different learning rates are compared: 0.001, 0.005 and 0.01. The higher learning rates lead to faster convergence and higher final reward values. Initially, at episode zero, all models start with a negative reward close to -500. As training progresses, the reward values increase steadily. The learning rate of 0.01 achieves the highest reward. It reaches nearly 2800 by episode 100. The learning rate of 0.005 performs well. It stabilizes at

around 2200. The lowest learning rate of 0.001 has a final reward of about 1700. The figure highlights the effect of different learning rates on training efficiency. A higher learning rate allows faster improvements in total reward risk instability if set too high. The learning rate of 0.01 shows the fastest increase. This suggests rapid learning.

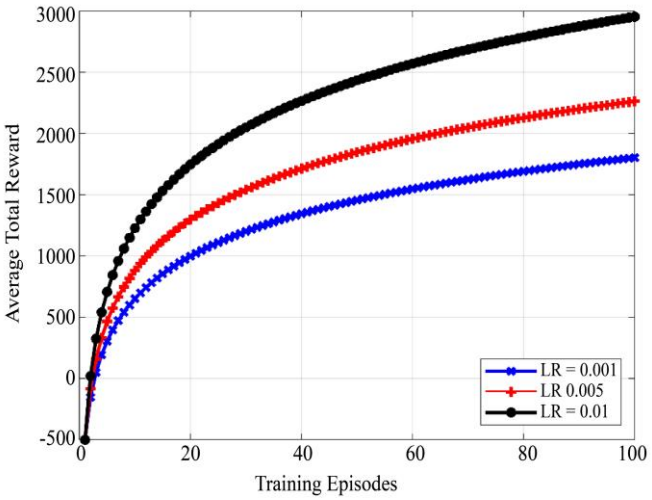

Fig. 12 Reward comparison of different learning rates

The learning rate of 0.005 performs well. It has a slightly slower convergence rate. The lowest learning rate of 0.001 results in the slowest learning process, with a more gradual increase in reward. The results confirm that selecting an optimal learning rate is crucial in reinforcement learning as it balances learning speed and stability. It enhances decision-making efficiency. This is important for reinforcement learning-based migration strategies.

Figure 13 illustrates the improvement in latency over time for different migration strategies. Five migration strategies are analyzed: no migration, greedy migration, multi-armed bandit, deep q-learning and deep recurrent actor-critic migration. The no migration strategy stays at 0 percent improvement. It shows no optimization. The greedy migration strategy improves slowly. It starts at about 5 percent. It reaches around 8 percent by the end.

The multi-armed bandit strategy performs better. It begins at 10 percent. It increases to about 14 percent. The deep q-learning method outperforms the previous strategies, showing an initial improvement of 15 percent and rising to nearly 20 percent over time.

The deep recurrent actor-critic migration method performs the best. It starts at 20 percent. It steadily increases to about 30 percent. The reinforcement learning-based strategies reduce latency better than conventional methods. The steady improvement over time suggests continuous adaptation. These models refine decision-making processes.

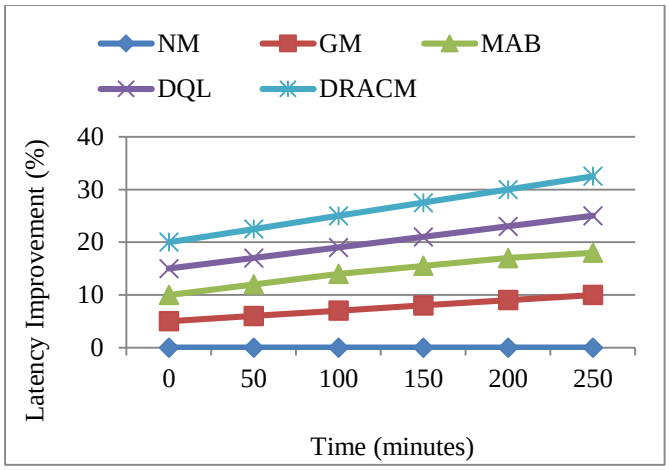

Fig. 13 Latency improvements versus Time

Figure 14 represents the variation of policy entropy over training episodes for different migration strategies. The graph compares five migration strategies: no migration, greedy migration, multi-armed bandit, deep q-learning and deep recurrent actor-critic migration. The result shows a consistent decline in policy entropy across all strategies as training progresses. At episode zero, the no migration strategy has the highest policy entropy. It starts at about 0.85. The greedy migration strategy follows. It begins at around 0.75. The multi-armed bandit and deep q-learning strategies begin at approximately 0.6 and 0.5, respectively. The deep recurrent actor-critic migration strategy starts with the lowest entropy at about 0.4. As training progresses, policy entropy decreases for all strategies. This shows that models become more confident in decision-making. By episode 100, the deep recurrent actor-critic migration strategy has the lowest policy entropy. It stabilizes at around 0.05. Deep Q-learning and multi-armed bandit strategies follow. The final entropy values are close to 0.12 and 0.15, respectively. The greedy migration strategy stabilizes at about 0.2. The no migration strategy has the highest entropy, staying above 0.3. The figure shows that reinforcement learning-based approaches improve policy stability.

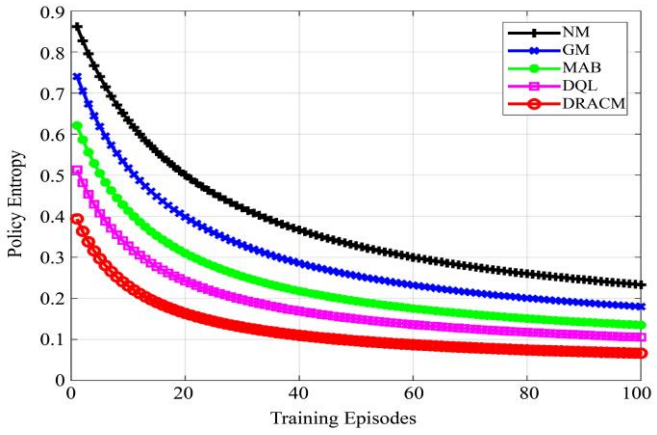

Fig. 14 Policy convergence versus Training episodes

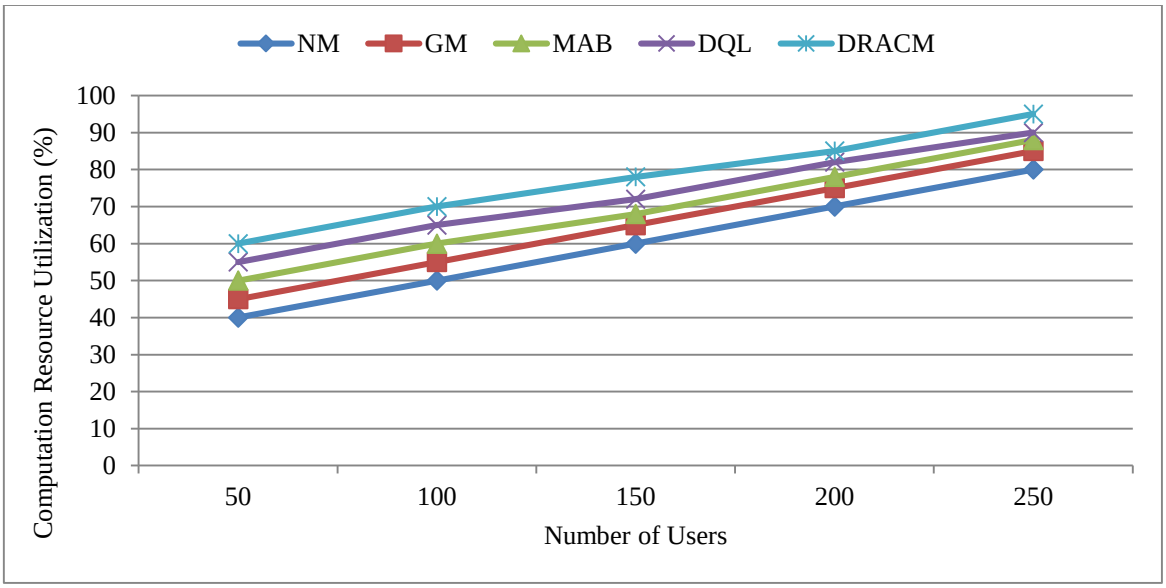

Fig. 15 Computational resource utilization versus Number of users

Figure 15 illustrates how computation resource utilization changes with the increasing number of users for different migration strategies. The graph compares five migration strategies: no migration, greedy migration, multi-armed bandit, deep q-learning and deep recurrent actor-critic migration. The results show that computation resource utilization increases across all strategies as the number of users grows. The no-migration strategy has the lowest utilization. It starts at about 40 percent for 50 users. It increases to around 80 percent at 250 users. The greedy migration strategy performs slightly better. Its utilization rises from about 45 percent. It reaches nearly 85 percent. The multi-armed bandit approach improves efficiency. It starts at 50 percent. It increases to around 88 percent. The deep q-learning strategy enhances resource utilization. It begins at 55 percent. It reaches about 90 percent. The deep recurrent actor-critic migration method has the highest efficiency. Utilization rises from 60 percent. It reaches nearly 95 percent as the number of users increases. These results show that reinforcement learning-based migration strategies improve resource allocation. This performs better than traditional methods. The overall suggests that deep reinforcement learning techniques optimize system performance.

6. Conclusion

This paper presented a novel approach for online service migration in MEC environments with incomplete system information. A reinforcement learning-based approach was employed to learn optimal migration policies without requiring complete knowledge of the system state. Extensive experiments were conducted to evaluate the performance of DRACM compared to traditional migration strategies. The results demonstrated that DRACM significantly reduced communication and computation delays while maintaining minimal migration overhead. The work analysed the scalability of DRACM under varying user densities. The findings indicated that DRACM maintains stable performance even as the number of users increases, unlike traditional approaches that suffer from high migration overhead and degraded system performance under heavy network loads. DRACM dynamically adjusts migration decisions to adapt to changing conditions. This flexibility makes it a promising solution for large-scale MEC deployments. It is useful in scenarios with unpredictable user mobility. The proposed approach provides a scalable and decentralized framework that allows mobile users to make autonomous migration decisions based on partial observations. By integrating LSTM networks, DRACM effectively captures temporal dependencies and improves policy stability.

Acknowledgments

The authors would like to express their sincere gratitude to all those who contributed to the success of this research work.

References

[1] Yuyi Mao et al, "A Survey on Mobile Edge Computing: The Communication Perspective," *Arxiv*, pp. 1-37, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[2] Tianchu Zhao et al., "A Cooperative Scheduling Scheme of Local Cloud and Internet Cloud for Delay-Aware Mobile Cloud Computing," *2015 IEEE Globecom Workshops (GC Wkshps)*, San Diego, CA, USA, pp. 1-6, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[3] Hongbo Jiang et al., "Joint Task Offloading and Resource Allocation for Energy-Constrained Mobile Edge Computing," *IEEE Transactions on Mobile Computing*, vol. 22, no. 7, pp. 4000-4015, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[4] Adberezak Touzene et al., *Immersive Virtual and Augmented Reality in Healthcare: An IoT and Blockchain Perspective*, CRC Press, pp. 1-244, 2023. [Google Scholar] [Publisher Link]

[5] Qiang Duan, Shangguang Wang, and Nirwan Ansari, "Convergence of Networking and Cloud/Edge Computing: Status, Challenges, and Opportunities," *IEEE Network*, vol. 34, no. 6, pp. 148-155, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[6] Leonard Kleinrock, *Communication Nets: Stochastic Message Flow and Delay*, Dover Publications, pp. 1-209, 2007. [Google Scholar] [Publisher Link]

[7] Shangguang Wang et al., "A Survey on Service Migration in Mobile Edge Computing," *IEEE Access*, vol. 6, pp. 23511-23528, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[8] Qing Zhao et al., "Decentralized Cognitive MAC for Opportunistic Spectrum Access in Ad Hoc Networks: A POMDP Framework," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 3, pp. 589-600, 2007. [CrossRef] [Google Scholar] [Publisher Link]

[9] Xiaoqian Li et al., "Intelligent Service Migration Based on Hidden State Inference for Mobile Edge Computing," *IEEE Transactions on Cognitive Communications and Networking*, vol. 8, no. 1, pp. 380-393, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[10] Yulu Gong et al., "Dynamic Resource Allocation for Virtual Machine Migration Optimization Using Machine Learning," *arXiv*, pp. 1-9, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[11] Robert Müller, Ulrike Greiner, and Erhard Rahm, "AgentWork: A Workflow System Supporting Rule-Based Workflow Adaptation," *Data & Knowledge Engineering*, vol. 51, no. 2, pp. 223-256, 2004. [CrossRef] [Google Scholar] [Publisher Link]

[12] Awder Ahmed, Sadoon Azizi, and Subhi R.M. Zeebaree, "ECQ: An Energy-Efficient, Cost-Effective and Qos-Aware Method for Dynamic Service Migration in Mobile Edge Computing Systems," *Wireless Personal Communications*, vol. 133, pp. 2467-2501, 2023. [CrossRef] [Google Scholar] [Publisher Link].

[13] Anita Choudhary et al, "A Critical Survey of Live Virtual Machine Migration Techniques," *Journal of Cloud Computing*, vol. 6, pp. 1-41, 2017. [CrossRef] [Google Scholar] [Publisher Link].

[14] Shenzhi Wang et al., "Train Once, Get a Family: State-Adaptive Balances for Offline-to-Online Reinforcement Learning," *37th Conference on Advances in Neural Information Processing Systems*, pp. 1-24, 2023. [Google Scholar] [Publisher Link]

[15] Sonja Haug, "Migration Networks and Migration Decision-Making," *Journal of Ethnic and Migration Studies*, vol. 34, no. 4, pp. 585-605, 2008. [CrossRef] [Google Scholar] [Publisher Link]

[16] Tao Ouyang, Zhi Zhou, and Xu Chen, "Follow Me at the Edge: Mobility-Aware Dynamic Service Placement for Mobile Edge Computing," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2333-2345, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[17] Shiqiang Wang et al., "Dynamic Service Migration in Mobile Edge Computing Based on Markov Decision Process," *IEEE/ACM Transactions on Networking*, vol. 27, no. 3, pp. 1272-1288, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[18] Tao Ouyang et al., "Adaptive User-Managed Service Placement for Mobile Edge Computing: An Online Learning Approach," *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, Paris, France, pp. 1468-1476, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[19] Yuxuan Sun et al., "Learning-Based Task Offloading for Vehicular Cloud Computing Systems," *2018 IEEE International Conference on Communications (ICC)*, Kansas City, MO, USA, pp. 1-7, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[20] Shangguang Wang et al., "Delay-Aware Microservice Coordination in Mobile Edge Computing: A Reinforcement Learning Approach," *IEEE Transactions on Mobile Computing*, vol. 20, no. 3, pp. 939-951, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[21] Chao-Lun Wu et al., "Mobility-Aware Deep Reinforcement Learning with Glimpse Mobility Prediction in Edge Computing," *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, Dublin, Ireland, pp. 1-7, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[22] Quan Yuan et al., "A Joint Service Migration and Mobility Optimization Approach for Vehicular Edge Computing," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 8, pp. 9041-9052, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[23] Xiaobo Zhou et al., "Energy-Efficient Service Migration for Multi-User Heterogeneous Dense Cellular Networks," *IEEE Transactions on Mobile Computing*, vol. 22, no. 2, pp. 890-905, 2021. [CrossRef] [Google Scholar] [Publisher Link]