Task Offloading and Resource Allocation in Mobile Edge Computing Using Improved Deep Reinforcement Online Offloading

Sweta Dash [1], Jibitesh Mishra [1], Sanjit Kumar Dash [1], Suleman Alnatheer [2,*], Mohammed Altaf Ahmed [2], and Abdullah Alsir Mohamed[2]

[1] School of Computer Sciences, Odisha University of Technology and Research, Bhubaneswar, Odisha, India
[2] Department of Computer Engineering, College of Computer Engineering and Sciences, Prince Sattam bin Abdulaziz University, Al-Kharj, Saudi Arabia
Email: swetadash123@gmail.com (S.D.); jmishra@outr.ac.in (J.M.); skdash@outr.ac.in (S.K.D.); s.alnatheer@psau.edu.sa (S.A.); m.altaf@psau.edu.sa (M.A.A.); a.mhamed@psau.edu.sa (A.A.M.)
*Corresponding Author

***Abstract*—Mobile Edge Computing (MEC) is a technology that enables mobile devices to transfer computationally demanding tasks to nearby servers. MEC significantly lowers the local processing load by allowing a variety of complicated mobile devices tasks to be transferred to the network system's edge so that they can be executed by the servers at the edge. In this research, we used a wirelessly powered MEC system, such that every Wireless Device (WDs) computational task is either completed locally or entirely offloaded to a MEC server. However, allocating resources for communication and computing in an edge-cloud environment efficiently is still difficult, though, particularly when there are several mobile devices and edge servers. So, an improved deep learning-based reinforcement model is suggested for Improved Deep Reinforcement Online Offloading (IDROO) as a scalable method to address this issue. This framework uses a deep neural network to learn the binary offloading decisions from experience and has additional output layers for resource allocation (transmission power and Central Processing Unit (CPU) frequency) along with the task offloading and it also optimizes the joint loss function that accounts for the offloading of tasks and efficiency of allocating resources. The simulation of the suggested model is done by taking number of users as 10, 20, 30, and the numerical result shows that the model can achieve near optimal performance in dynamically changing environmental conditions. Specifically, IDROO improves computation rate by approximately 18–25% and reduces energy consumption by around 15–20% compared to existing methods, while achieving faster convergence.**

***Keywords*—task offloading, data computation, edge-cloud environment, bit-error, allocating resources, task offloading, Improved Deep Reinforcement Online Offloading (IDROO)**

I. Introduction

The swift advancement of mobile computing technologies, such as 5G networks, a wide variety of mobile devices, including smart phones, mobile robots and wearable devices, have emerged explosively, which has led to a variety of computationally expensive intelligent applications such as natural language processing, virtual reality, big data analytics, face recognition, and ultra-high-definition video [1, 2]. Such applications usually involve stringent latency and computation requirements, but their mobile devices often grapple with resource constraints, characterized by limited processing power and modest battery capacity. Thus, it is rather difficult to deal with these computationally expensive and latency-sensitive tasks [3]. Meanwhile, the latest advancements in Mobile Edge Computing (MEC) technology can substantially increase the computational capabilities of the device. To lower computational delay and utilization of energy, the Wireless Device (WDs) can use MEC to transfer computationally demanding jobs to adjacent servers at the edge [4].

Resource limitations on IoT devices can be resolved via task outsourcing technologies. Its main goals are to bring the cloud's computing environment closer to the edge's network, give users access to resources for computing, make it possible for users to use nearby storage and compute services, react rapidly to user demands for computing, and enhance the user experience [5, 6]. Task offloading can improve quality of experience by decreasing processing latency, prolonging the device's life by lowering energy usage, and optimizing profits for suppliers and network administrators. MEC brings cloud capabilities closer to the end devices, allowing for more powerful servers to compute on behalf of the mobile devices.

Significant gains in response time and processing power have been made possible by MEC, which also has positive benefits on lowering latency and energy usage [7, 8]. A wirelessly-powered MEC system's ideal

Manuscript received March 24, 2025; revised May 8, 2026; accepted May 14, 2026; published September 4, 2026.

offloading solution in a wireless fading atmosphere is significantly influenced by the dynamic wireless network setting [9]. A wirelessly powered MEC network with one base station and several WDs, each of which adheres to a binary offloading policy, was examined in this research [10, 11]. Our specific goal is to jointly optimize the CPU frequency, duration of transmission, and offloading task decisions made by each WD in accordance with the dynamically varying conditions. Recent studies have explored Deep Reinforcement Learning (DRL) for dynamic task offloading in mobile edge computing environments to improve system efficiency and adaptability [12]. In addition, joint optimization of task offloading and resource allocation has been investigated to enhance overall system performance.

Despite significant advancements in deep reinforcement learning-based task offloading techniques, existing approaches such as DROO, A3C, and Preferred Provider Organization (PPO) primarily focus on optimizing offloading decisions independently, often neglecting the joint optimization of critical system parameters such as transmit power and Central Processing Unit (CPU) frequency. Moreover, these methods suffer from high convergence time and increased computational complexity when deployed in dynamic wireless environments with fluctuating channel conditions. This creates a gap in designing a scalable and efficient framework that can simultaneously optimize multiple decision variables while ensuring fast convergence and energy efficiency. Therefore, there is a strong need for an improved intelligent offloading mechanism that can address these limitations in real-time MEC systems.

The main objective of this work is to develop an efficient deep reinforcement learning-based framework, termed Improved Deep Reinforcement Online Offloading (IDROO), for joint task offloading and resource allocation in mobile edge computing environments. Specifically, the proposed model aims to:

- Maximize the weighted computation rate of the system,
- Minimize energy consumption of wireless devices,
- Reduce convergence time for real-time decision-making, and
- Adapt effectively to dynamic wireless channel conditions.

By integrating offloading decisions with transmit power and CPU frequency allocation, the proposed framework seeks to achieve superior performance compared to existing state-of-the-art methods

Unlike conventional DROO-based approaches that focus only on binary offloading decisions, the proposed framework extends the decision space to jointly optimize task offloading, transmit power, and CPU frequency within a unified deep reinforcement learning model. This transforms the problem from a single-dimensional decision process into a multi-dimensional resource optimization task, enabling coordinated management of communication and computation resources. Such a formulation better reflects practical MEC environments and leads to improved system performance. The improvement lies in the formulation of a coupled optimization problem rather than a simple extension of the network architecture.

Some of the contributions for this framework are as follows:

- A joint optimization framework is proposed that simultaneously considers task offloading, transmit power, and CPU frequency allocation.
- A multi-dimensional action space is formulated within a DRL framework for coordinated resource management.
- An efficient action selection mechanism is used to handle the complexity of combinatorial decision spaces.
- The proposed approach achieves improved performance in terms of computation rate, energy efficiency, and convergence speed.

The IDROO model is evaluated under detailed numerical analysis by taking varying number of WDs, i.e., 10, 20 and 30. The next section provides a brief overview of some of the related works in this field.

The remainder of this paper is organized as follows. Section II reviews related work on task offloading in MEC. Section III presents the system model, including local and edge computation. Section IV describes the proposed IDROO framework and its reinforcement learning formulation. Section V provides the experimental setup, simulation results, and performance comparisons. Finally, Section VI concludes the paper and discusses future work.

II. Literature Review

Before delving into the literature review, it is essential to understand the growing importance of efficient task offloading in Mobile Edge Computing (MEC). With the proliferation of resource-constrained mobile devices and latency-sensitive applications, optimizing task allocation between local devices and edge servers has become a critical research focus. This study explores existing works to identify advancements, challenges, and gaps in task offloading using DROO within the MEC framework.

Huang *et al.* [13] introduced the DROO framework, a DRL based approach for optimizing offloading of tasks in MEC networks under binary policies. DROO achieves near-optimal results with low latency by learning from past actions, leveraging quantization, and dynamically adjusting parameters. Tang and Wong [14] used a distributed, model-free deep reinforcement learning algorithm, enabling independent decision-making without prior task model knowledge. By integrating LSTM, dueling DQN, and double-DQN, their approach ensures scalability and adaptability in decentralized environments.

Yu *et al.* [15] proposed MOEA/D-EoD, a multi-objective optimization technique based on decomposition for efficient resource allocation in MEC systems. The algorithm excels in balancing transmission power and resource constraints but faces challenges in computational complexity and adaptability. Tan *et al.* [16] proposed a dynamic offloading strategy for MECC in IoT applications, combining a learning automata-based

decision-maker with LSTM-based workload prediction for auto-scaling. Using an A3C algorithm with fuzzy logic, the approach ensures energy efficiency, stability, and timely task execution under dynamic workloads. Dai *et al.* [17] proposed a learning-based co-offloading framework for MEC and IIoT, enabling tasks to offload via cellular or D2D links to reduce delays and network congestion.

Dong *et al.* [18] adopted task offloading strategies using PSO and QPSO to reduce the usage of energy, time of work completion, and processing time. Joint task offloading and resource allocation in multi-server MEC environments highlighted the importance of coordinated decision-making by Tran and Pompili [19]. They have addressed joint objective problem in multi-cell MEC networks by formulating a MINLP and solving it via a decomposition-based approach. Their method achieves near-optimal performance, improving offloading utility while highlighting future goals of reducing runtime and enhancing fairness. Yang *et al.* [20] used DSLO, a DL supervised algorithm using CNNs and batch normalization to optimize outsourcing and allocation of bandwidth in MEC systems.

Chen *et al.* [21] introduced DROO-C, a DNN based framework for optimizing task offloading and data transmission in MEC environments. By combining Mixed-Integer Planning and entropy reduction-based compression, it improves transmission rates and minimizes delays effectively.

To address the limitations for onboard computational resources and strict delay guarantees, a machine learning-based task offloading framework for Connected Autonomous Vehicles (CAVs) was proposed by Kumar *et al*. [22]. The framework dynamically determines optimal offloading decisions based on vehicle context and network conditions, improving real-time data processing efficiency in edge computing environments.

Deng *et al.* [23] proposed an intelligent delay-aware partial task offloading framework for multi-user Industrial Internet of Things (IIoT) environments using reinforcement learnings including Q-learning and Deep Deterministic Policy Gradient (DDPG), to optimize offloading decisions under varying channel conditions and delay requirements.

Chen *et al.* [24] proposed a DRL-based framework for dynamic task offloading in MEC systems, demonstrating improved adaptability under varying network conditions. Wang *et al.* [25] developed a deep reinforcement learning-based resource allocation strategy to enhance system efficiency.

Chen *et al.* [26] presented an efficient multi-user computation offloading strategy, focusing on optimizing system throughput. Similarly, Xu *et al.* [27] explored joint service caching and task offloading to improve performance in dense networks. Ning *et al.* [28] applied deep reinforcement learning in vehicular edge computing, enabling intelligent offloading decisions in dynamic environments.

Wang *et al.* [29] explored federated learning for intelligent edge computing, while Chen *et al.* [30] discussed early architectures for computation offloading in cloudlet-based systems.

A broader comparison with other state-of-the-art Deep Reinforcement Learning (DRL) algorithms would enhance the quality and completeness of the study is being agreed. While A3C and PPO show competitive performance in some dynamic channel settings, they incur higher training complexity and convergence time. DDPG, which is well-suited for continuous control, performs sub optimally in our case due to the binary nature of the offloading action space.

Table I summarizes the limitations of existing task offloading approaches and highlights how the proposed IDROO framework addresses these challenges. Unlike conventional methods, the proposed approach integrates offloading decisions with resource allocation and achieves improved performance with lower computational complexity and faster convergence.

TABLE I. GAP ANALYSIS OF EXISTING METHODS VS PROPOSED IDROO

Method	Key Limitations	HowIDROO Addresses It
DROO	Focuses only on offloading decisions; no joint resource optimization	Jointly optimizes offloading, transmit power, and CPU frequency
A3C	High training complexity and slow convergence	Faster convergence using efficient DNN + quantization
PPO	High computational overhead; not tailored for binary actions	Designed specifically for binary offloading with efficient action selection
DDPG	Not suitable for discrete/binary action space	Uses quantization to handle discrete decisions effectively
MOEA/D	High computational complexity; not adaptive	DRL-based adaptive decision-making in dynamic environments
Proposed IDROO	—	Provides joint optimization, faster convergence, and improved efficiency

III. SYSTEM MODEL

A wirelessly powered MEC network is examined with an AP and N fixed WDs, represented as a collection $N = \{1, 2, 3, ..., n\}$, where each device has one antenna, as illustrated in Fig. 1. The computational tasks of each wireless device are passed through the IDROO framework, and a decision is made whether the task is computed locally or offloaded to the Access Points (AP). This can be equivalent to a low power IoT system or a fixed sensing connection. The AP can transmit Radio Frequency (RF) energy to the WDs and has a steady power source. Every WD contains a battery that can be charged and that can hold the energy that has been captured and use it to power the device. We assumed that the AP is more computationally capable than the WDs, allowing the WDs to delegate their computing responsibilities to the AP. In a fixed IoT parameter the device's time is split into successive intervals with similar durations T, which are set less than the channel coherence time. The wireless channel gain at any time is connected to the energy harvesting capacity of a WD from the AP as well as to the speed of communication between them. Let h_i represent

the channel's wireless gain at a certain time interval between the AP and the i_{th} WD. Although it may alter between frames, the channel is thought to be reciprocated in the uplink and downlink to be made constant throughout each time frame. Before the conclusion of a time, each WD must use the energy it has harvested to complete a computational activity that has been prioritized. The i_{th} WD is given a special weight w_i. In this work, an offloading strategy is examined in which the job is either transferred to the AP or performed directly at the WD. Let x_i be an indicator variable that belongs to $\{0, 1\}$. If $x_i = 1$, the job is transferred to the AP for the i_{th} and if $x_i = 0$, it is performed locally.

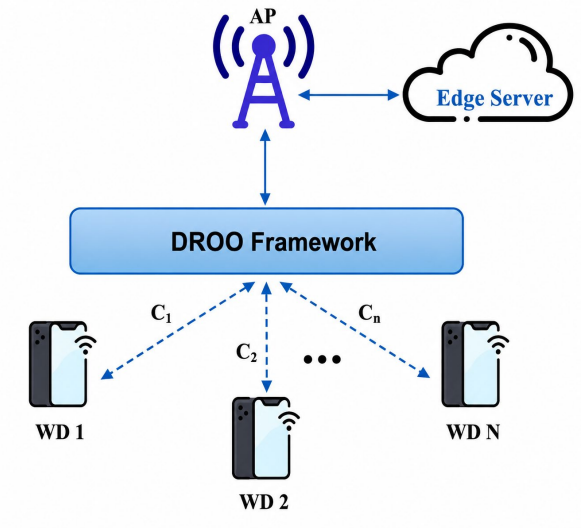


Fig. 1. System model of an MEC network.

In this work, the wireless channel gain is modeled using path loss and small-scale fading to capture the fundamental characteristics of wireless communication. More complex factors such as user mobility, interference, network congestion, and multi-cell interactions are not considered, as their inclusion would significantly increase the state space dimensionality and introduce strong coupling among users, making the optimization problem difficult to solve in real time using deep reinforcement learning. Therefore, a simplified channel model is adopted to maintain analytical tractability and focus on joint task offloading and resource allocation, while incorporating these factors is left for future work.

In this work, a binary offloading policy is adopted, where each task is either executed locally or fully offloaded to the edge server. This assumption simplifies the decision space and reduces computational complexity, as commonly considered in existing works [13]. However, in practical MEC environments, partial task offloading, where a task is partitioned between local and edge execution, can provide greater flexibility and improved performance. Extending the proposed framework to support partial offloading is an important direction for future work. In a more general case, a continuous variable $\beta_i \in [0,1]$ can be used to represent the fraction of task offloaded, enabling partial offloading.

A. Local Computation

In the local computation mode, a WD can perform its task and gather energy at the same time. Let $0 \le t_i \le T$ represent the time for computation and f_i represent the processor's speed of computation (cps). Then, $f_i t_i / \alpha$ is the total number of bits processed by the WD, where $\alpha > 0$ indicates the number of cycles required to process a single task data bit. In the meantime, $k_i f_i^3 t_i \le E_i$, limits the amount of energy that the WD uses for computing. It can be demonstrated that a WD must use up all the energy it has captured and compute continuously to process the most data possible within T while adhering to the energy limitation. The local computational rate (in bps) is given by Eq. (1):

$$R_{local} = \left(\frac{E}{k_i}\right)^{\frac{1}{3}} \quad (1)$$

where E = harvested energy and k_i= energy efficiency coefficient whose value is 1×10^{-26}, which is a constant value.

B. Edge Computation

A WD in the transferring phase can only transfer its work to the AP after collecting energy because of the Time Division Multiplexing (TDD) limitation. We represent t_i as the i-th WD's transferring time, where t_i falls between 0 and 1. In this case, we suppose that the AP's transmission capacity and speed of computation are, for example, more than three orders of magnitude greater than those of the WDs, which are smaller and have less energy. In addition, the data offloaded to the edge server is substantially longer than the computation feedback that needs to be downloaded to the WD. For analytical simplicity, the computation and download delay at the edge server are assumed to be negligible compared to transmission delay. This assumption is reasonable when the edge server has significantly higher computational capability than the wireless devices. However, in practical heterogeneous MEC environments, edge processing and download delays may have a non-negligible impact and should be incorporated into the optimization model. An offloading WD uses all its gathered energy for job offloading to optimize its computing rate, and the computation rate equals its data transferring capability, which is determined by given Eq. (2):

$$R_{offload} = B \log_2\left(1 + \frac{P*G}{N}\right) \quad (2)$$

where P (Power) = E/T, G is the channel gain, B is the bandwidth and N is the noise.

C. Total Computation

We believe that only the channel's wireless gains are time-sensitive throughout the period under consideration, with all other system settings in local and edge computing modes being fixed. Consequently, the MEC network's weighted sum computational rate over a specified time is represented as:

$$Q = \Sigma(R_{local} + R_{offload}) \quad (3)$$

where Q = Total computation rate

Our goal is to maximize the weighted sum of computational rate for every time interval with channel realization h. For real-time system optimization under rapid disappearing channels, conventional optimization techniques call for iteratively modifying the offloading decisions to achieve the optimal, which is basically impossible. A unique DRL based online outsourcing method is suggested that can solve this problem in milliseconds to address the complexity issue.

IV. The IDROO Architecture

The improved IDROO model is designed to maximize decisions about allocating of resources and offloading of tasks in edge computing environments. The core objective of this algorithm is to develop an offloading decision function π, once the channel realization is known at the beginning of each time interval, produces the best offloading actions, denoted by x (where x belongs to {0, 1}, indicating whether the task is offloaded or processed locally). The proposed IDROO algorithm employs Deep Reinforcement Learning (DRL) to learn this policy function gradually, based on experience gathered over time. DRL is an AI technique that integrates Deep Neural Networks (DNN) with reinforcement learning principles, where the model learns to take actions based on rewards or penalties, aiming to maximize long-term rewards. In edge computing, DRL helps decide whether tasks should be transferred to the edge or processed locally, considering factors such as task drop rate, latency, and energy consumption. The integration of Deep Reinforcement Learning (DRL) with Deep Neural Networks (DNNs) is essential in our proposed approach because MEC environments involve complex and dynamic system states—such as user mobility, wireless channel fluctuations, and computational resource availability. Traditional reinforcement learning techniques struggle to handle such high-dimensional input and output spaces. Deep neural networks enable efficient approximation of the policy and value functions in these settings. Apart from that it also helps in scalability and generalization along with continuous learning.

In the proposed framework, reward shaping is employed to guide the agent towards optimal offloading and resource allocation decisions by incorporating task completion time, energy consumption, and system cost into the reward function. The Q-value update mechanism refines the policy by iteratively adjusting action-value estimates based on observed rewards and future value predictions, enabling the agent to learn effective strategies over time. To enhance learning stability and efficiency, experience replay is integrated, allowing the model to store past interactions and sample them randomly during training, thereby breaking correlation between consecutive experiences and improving convergence.

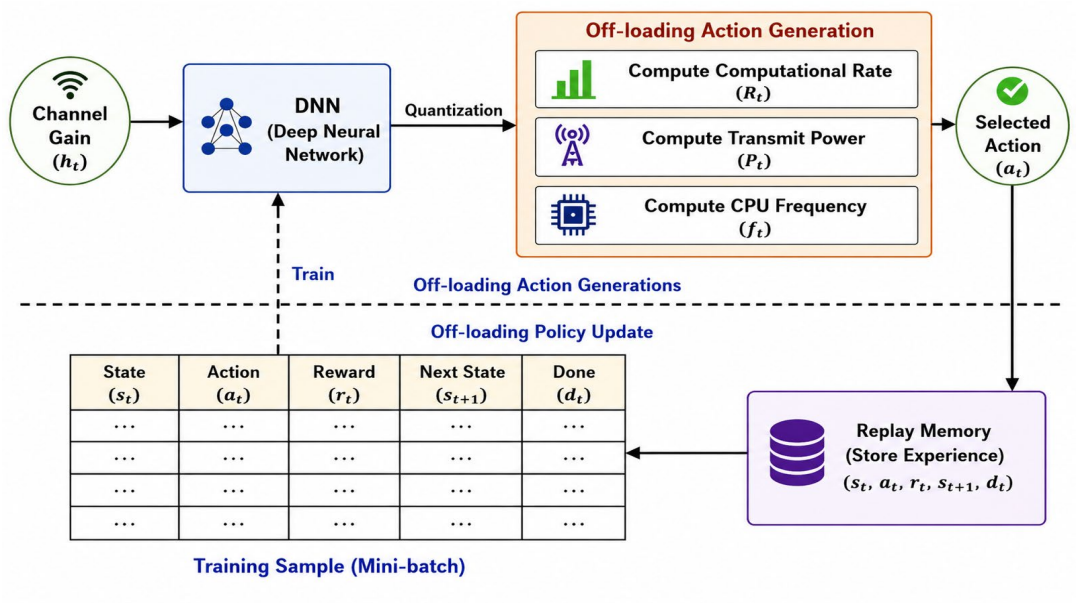


Fig. 2. Architecture of the proposed IDROO framework for joint task offloading.

The structure of the Improved Deep Reinforcement based Online Offloading (IDROO) framework is given in Fig. 2. Offloading activity formulation and offloading strategy updating depending on the present channel condition are its two alternate processes. The input for the DNN is the channel gain h_t which is the function that simulates the channel gain for each device based on the distance. At each time frame t, the system observes the channel gain h_t, which represents the current state of the communication channel. The channel gain is a crucial parameter that determines the efficiency of offloading tasks, representing the quality or capacity of the communication channel. It is simulated using a random distance between 2.5 and 5.2 m, with random exponential fading added to emulate real-world scenarios. This simulation returns a value corresponding to the effective channel gain, accounting for both path loss and fading. The DNN in DROO predicts offloading decisions by taking the

channel gain as input and generating a relaxed offloading action x_t, where every value is continuous between 0 and 1. The decision produced by the DNN, however, is continuous, while the system requires discrete offloading actions. Thus, the algorithm includes a quantization function, which maps the continuous output of the DNN to a set of discrete values or actions. These discrete actions represent specific offloading configurations; each associated with a certain amount of computation transferred to the edge. This quantization step ensures that the actions are feasible for practical implementation.

To optimize offloading decisions, the algorithm computes the Q-values for each quantized action. The Q-value Q (h_t, x_k) represents the expected reward or performance of acting x_k in the given state h_t. The optimal offloading action is then selected using an argmax operation, choosing the action x_k that maximizes the Q-value. This action represents the optimal decision for the current time frame, maximizing offloading efficiency based on the state of the channel. In our framework, the Deep Neural Network (DNN) generates a relaxed offloading vector where each element represents a continuous approximation of the offloading decision for each Wireless Device (WD). However, actual offloading actions must be discrete - either 0 (local execution) or 1 (edge offloading). Therefore, a quantization step is essential to convert the continuous DNN outputs into a valid binary action space that is practically executable. Then we designed a K-quantized action generation mechanism where the continuous vector is mapped to K nearest binary vectors using quantization algorithms such as K-Nearest Neighbors (KNN) or Random Forest (RF)-based discretization. These methods evaluate the proximity of two predefined binary action candidates that are feasible under system constraints. The choice of K was empirically determined to balance computational efficiency and decision granularity. A larger K improves solution quality by exploring more candidate actions but increases complexity. In our implementation, K is set to 32, which provided a good trade-off between performance and runtime. The use of KNN or Random Forest is limited to action quantization and does not affect the reinforcement learning process. The DNN generates a continuous action vector, which is mapped to a finite set of feasible binary actions using KNN/RF-based discretization. This reduces the complexity of the exponentially large action space while maintaining decision quality, achieving a balance between computational efficiency and accuracy.

The proposed DNN architecture is designed to balance model expressiveness and computational efficiency. A two-layer structure with 128 and 64 neurons is used to capture nonlinear relationships between channel conditions and offloading decisions while avoiding excessive complexity. ReLU activation is applied in hidden layers to ensure efficient training and faster convergence, while sigmoid activation in the output layer maps values to the range [0,1] for decision representation. A threshold of 0.5 is used to convert continuous outputs into binary offloading decisions, as it provides a balanced and stable decision boundary in practice.

The key parameters are selected based on a trade-off between performance and computational complexity. The value $K = 32$ provides sufficient action space exploration while maintaining manageable runtime. Similarly, the DNN architecture with 128 and 64 neurons effectively captures nonlinear relationships without incurring excessive training cost. Larger configurations offer only marginal improvements with increased complexity, making the chosen setup a balanced and efficient design.

The DNN produces a continuous action vector with values in the range [0, 1], representing the likelihood of offloading for each device. To obtain feasible decisions, this vector is converted into binary actions using a threshold-based quantization, where values above a predefined threshold (e.g., 0.5) are mapped to 1 (offloading) and others to 0 (local execution).

To improve decision quality, multiple candidate binary actions are generated and evaluated using the Q-function. The candidate that yields the highest expected reward is selected as the optimal offloading decision. This approach ensures an efficient mapping from continuous outputs to discrete action space while maintaining computational tractability.

The DNN's output is not limited to just the offloading decision but also includes the actual amount of computation resources used and the achieved transmission power for that time frame. It combines the prediction of offloading decisions, transmit power and CPU frequency into a single vector of size $3n$, where n is number of outputs. As the algorithm progresses, it stores past experiences in a replay memory, which includes data such as channel gain, actions taken, and the resulting offloading performance. The replay memory helps to prevent overfitting and allows the model to generalize better by sampling random batches of data for training. The policy is periodically updated based on new training data, which improves the DNN's ability to make accurate offloading decisions over time.

State Space: The system state at time step t is defined as:

$$s_t = \{h_t, E_t, B_t\} \tag{4}$$

where h_t represents the channel gain vector of all wireless devices, E_t denotes the available energy levels, and B_t represents bandwidth or system resource conditions. This state captures the dynamic network environment influencing offloading decisions.

Action Space: The action at time step t is defined as:

$$a_t = \{x_t, P_t, f_t\} \tag{5}$$

where $x_t \in \{0,1\}^N$ denotes the binary offloading decisions, P_t represents transmit power allocation, and f_t denotes CPU frequency allocation.

Reward Function: The reward function is designed to capture the trade-off between computation performance and energy efficiency:

$$r_t = \sum_{i=1}^{N} w_i R_i(t) - \lambda_1 E_i(t) - \lambda_2 D_i(t) \tag{6}$$

where:

- $R_i(t)$: computation rate
- $E_i(t)$: energy consumption
- $D_i(t)$: delay (latency)
- w_i: priority weight
- λ_1, λ_2: balancing coefficients

This formulation ensures a multi-objective optimization framework.

The weighting factors λ_1 and λ_2 in the reward function control the trade-off between energy consumption and delay. A higher value of λ_1 prioritizes energy efficiency, while a higher value of λ_2 emphasizes latency reduction. In this work, these parameters are selected to maintain a balanced optimization between delay and energy consumption. The choice of weights can be adjusted depending on application requirements, such as delay-sensitive or energy-constrained scenarios.

Q-Learning Formulation: The objective of the agent is to learn an optimal policy π^* that maximizes the expected cumulative reward:

$$Q(s_t, a_t) = r_t + \gamma \max_{a'} Q(s_{t+1}, a') \quad (7)$$

where γ is the discount factor.

Policy Learning: The deep neural network approximates the Q-function and is updated iteratively using experience replay and gradient-based optimization.

Algorithm 1. An improved online DROO (IDROO) algorithm

Input:
Wireless channel gain $h(t)$ at each time frame t
Number of quantized actions K
Output:
Optimal offloading action and resource allocation for each time frame
Initialize:
Initialize Deep Neural Network (DNN) with random parameters
Initialize empty replay memory
Set iteration number i
Set training interval d
for each time frame t do
Observe current wireless channel gain $h(t)$
Generate relaxed offloading action
Quantize relaxed action into K binary actions using KNN or RF algorithm
Compute Q-values for all candidate actions
Select the action with the maximum Q-value as the optimal offloading action
Execute selected action
Store state-action pair in replay memory
if t mod d == 0 then
Randomly sample a mini-batch from replay memory
Train DNN using sampled batch
Update DNN parameters using Adam optimizer
end if
end for
Return:
Optimal offloading action
Corresponding resource allocation

This formulation ensures that the proposed approach follows a standard deep Q-learning framework rather than a heuristic decision-making strategy (see Algorithm 1**)**.

In the first step the system parameters are initialized such as communication bandwidth, task requirements, and learning model hyper parameters. The IDROO model which is a neural network model consists of an input layer accepting N inputs. It has two hidden layers with 128 and 64 neurons respectively having ReLU activation function. There is also an output layer with 3n neurons having sigmoid activation function, representing offloading decision (1 for offloading, 0 for local processing), transmit power and CPU frequency allocation. The model uses Adam optimizer, and loss is measured by mean square error.

Then, for each iteration channel gain is taken as input and the IDROO model is used to forecast the ideal offloading and resource allocation decision based on the values from the model. The model predicts value by using threshold value of 0.5 which is used to convert predictions into binary decisions. The decision determines how tasks should be processed either locally or offloaded to the edge. Then the offloading rate Eqs. (1)−(3), transmit power scaling Eq. (4) and CPU frequency Eq. (5) are calculated based on the selected decision considering factors like bandwidth, noise, latency, power, etc. After that the model is trained with the goal to improve the policy for making better offloading decision and optimal resource allocation and the results such as offloading rates, rewards, and performance metrics are stored.

$$P_i = d_i \times P_{max} \quad (8)$$

where P_i = Transmit power scaling, d_i is predicted value from model for task i and P_{max} is the maximum allowable transmission power (here, assumed to be 1).

$$f_i = d_i \times f_{max} \quad (9)$$

where f_i = CPU frequency, d_i is predicted value from model for task i and f_{max} is the maximum allowable CPU frequency.

In summary, the IDROO algorithm leverages deep reinforcement learning, specifically a deep Q-learning framework, to optimize resource allocating and offloading in dynamic MEC environments. It consists of various essential components, including observing the channel gain, using a DNN to predict offloading decisions, quantizing those decisions, computing Q-values, selecting the optimal action, and updating the policy based on experience. This approach is particularly suited for edge computing or wireless communication scenarios, where the offloading decisions must adapt to continuously changing network conditions.

V. Results and Discussion

In this part, the suggested IDROO algorithm's performance using simulations. In this study, that a basic two-layer perceptron is discovered which is sufficient to provide satisfactory convergence performance; further optimization of the DNN parameters is predicted to yield

higher convergence performance. TensorFlow 1.0 is used to build the IDROO method in Python, setting the learning rate for the Adam optimizer to 0.01 and the training interval to 10. We also set the memory size to 128 and the training batch size to 32. We considered a MEC network with N = 10, 20 and 30 WDs. Then various plots are implemented for different no of WDs and compare the results among them. The results are obtained under a simplified channel model for tractability.

Fig. 3 represents the performance of the IDROO algorithm in task offloading over time, measured across a series of epochs. The plots demonstrate significant variations in the computation rate over epochs. This suggests that the IDROO algorithm is exploring and learning different task-offloading strategies, which can lead to fluctuations as it seeks an optimal solution.

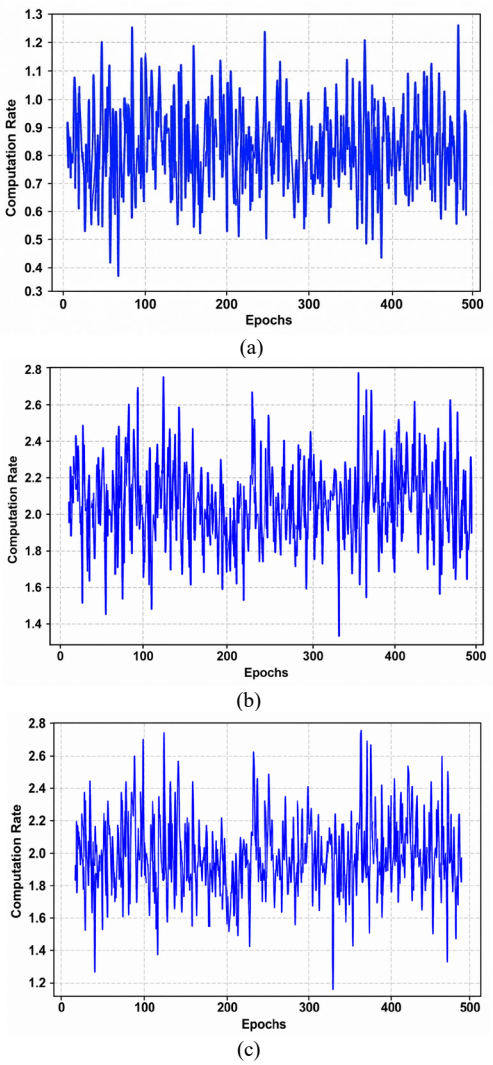


Fig. 3. Evolution of computation rate over epochs using the proposed IDROO algorithm for (a) N = 10, (b) N = 20, and (c) N = 30 wireless devices.

Fig. 4 represents the offloading decisions made by a system over time for different no of Wireless Devices (WDs). The color bar on the right indicates the offloading decision, in which red color indicates the task is offloaded to a remote server or another device and blue color indicates task is performed on the device itself. The horizontal stripes indicate consistent decision-making for specific devices across epochs.

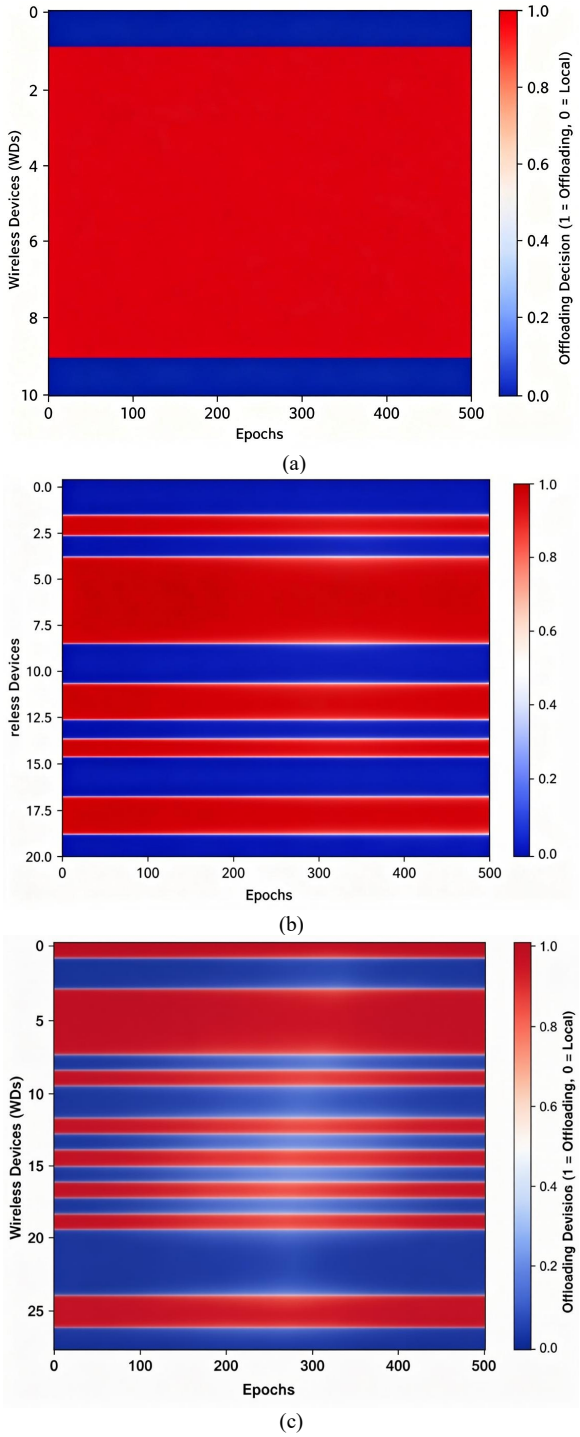


Fig. 4. Offloading decision patterns generated by the IDROO algorithm over epochs for (a) N = 10, (b) N = 20, and (c) N = 30 wireless devices.

Fig. 5 compares the computation rates of local processing versus offloading over 500 epochs using the IDROO algorithm for task offloading. The blue line represents the computation rate when tasks are processed on the devices whereas the green line represents the computation rate when tasks are offloaded to the edge server.

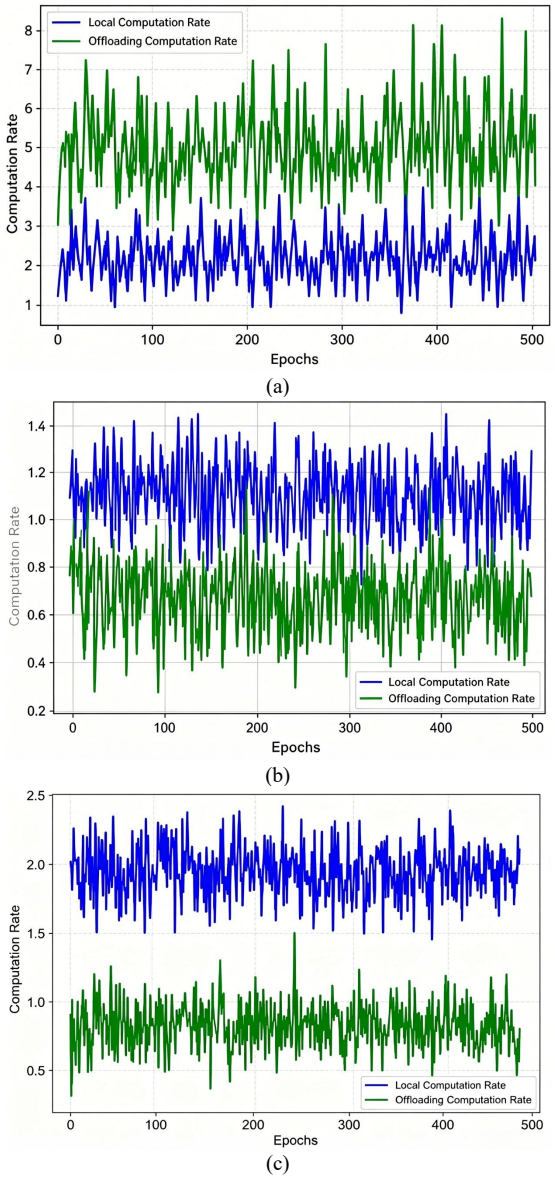
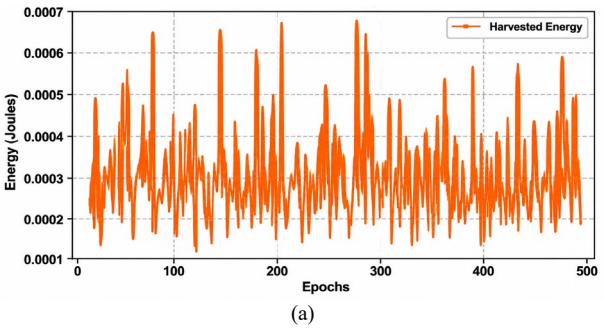


Fig. 5. Comparison of local computation rate and offloading computation rate over epochs for (a) $N = 10$, (b) $N = 20$, and (c) $N = 30$ wireless devices.

Fig. 6 represents the total energy harvested over time for a wireless communication system with energy harvesting. The energy harvested varies significantly between epochs, suggesting dynamic changes in channel conditions (e.g., fading and path loss) or the availability of energy sources. The plot does not exhibit an increasing or decreasing trend, indicating that the system might be operating in a stochastic environment.



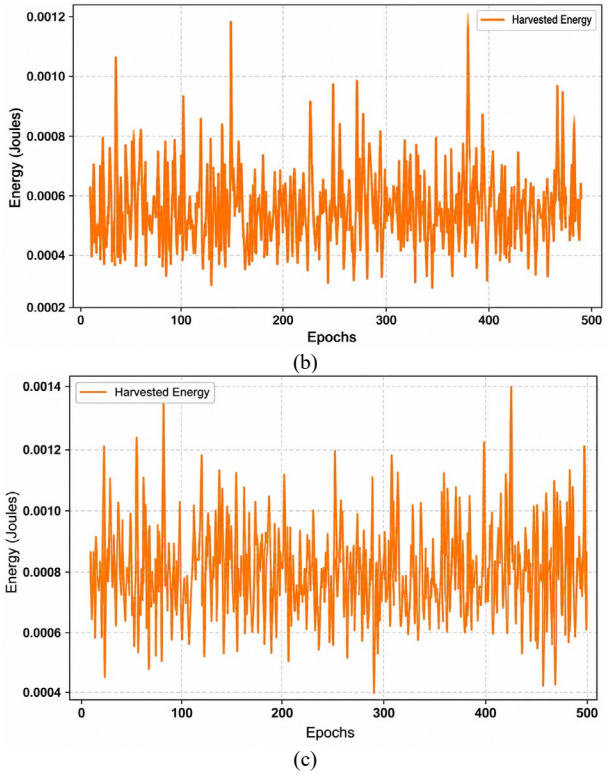
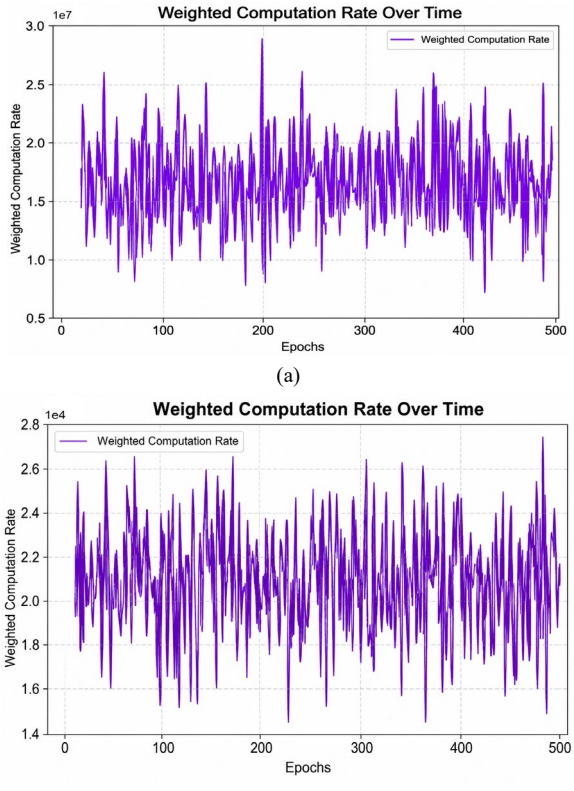
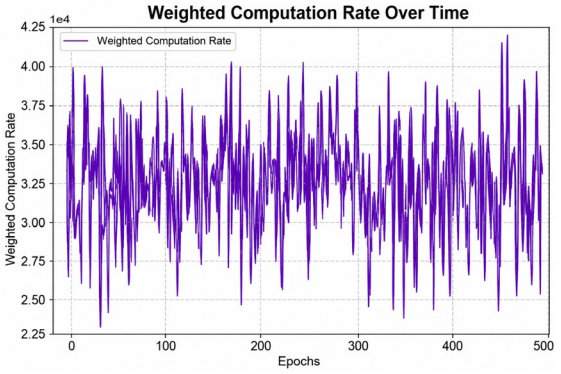


Fig. 6. Total harvested energy versus epochs for (a) $N = 10$, (b) $N = 20$, and (c) $N = 30$ wireless devices.

Fig. 7 represents the weighted computation rate over time (epochs) in the task offloading system. The weighted computation rate remains relatively stable around a mean value, showing that the system performs consistently despite stochastic variations.





(c)

Fig. 7. Weighted computation rate achieved by the proposed IDROO framework over epochs for (a) $N = 10$, (b) $N = 20$, and (c) $N = 30$ wireless devices.

Fig. 8 represents the normalized transmit power allocated to 10 devices over a series of epochs in a dynamic environment. Each bar corresponds to a specific epoch and is composed of different coloured segments. Each color represents the power allocated to a specific device (Device 1 through Device 10), as indicated. The varying heights of the stacked bars indicate that the transmit power assigned to each device changes dynamically according to the network conditions and resource requirements. This adaptive allocation strategy enables efficient utilization of communication resources while maintaining system performance. The observed variations demonstrate the capability of the proposed framework to intelligently distribute transmit power among devices, thereby supporting effective task offloading and improving overall MEC system efficiency.

Fig. 9 represents a graph which shows the trend of resource allocation (in units) over 200 epochs. The declining trend could signify an algorithm learning to optimize resource usage, allocating fewer resources while maintaining performance. The fluctuating nature of the curve suggests that the resource allocation is responsive to real-time changes in system requirements or external conditions. Fig. 10. shows the performance of the IDROO model with respect to different learning rates and the model gives the best performance when the learning rate is 0.001.

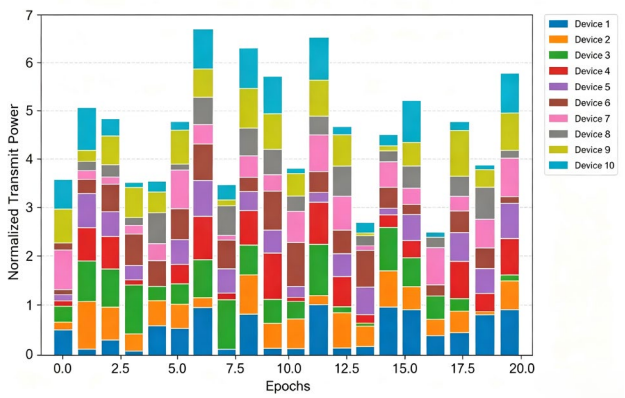


Fig. 8. Normalized transmit power allocation of wireless devices over epochs using the proposed IDROO framework.

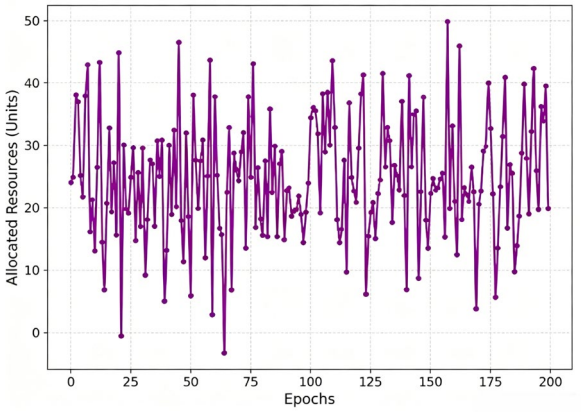


Fig. 9. Resource allocation dynamics over epochs under the proposed IDROO-based task offloading framework.

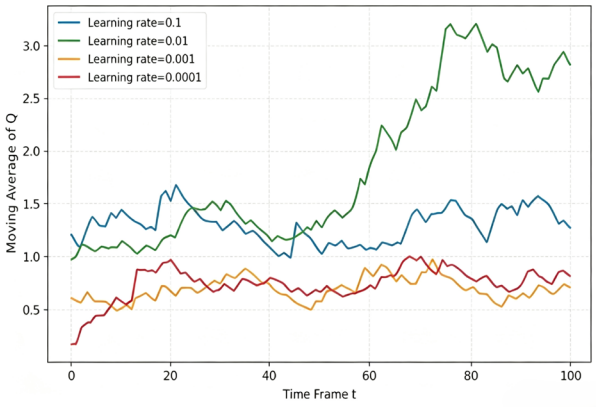
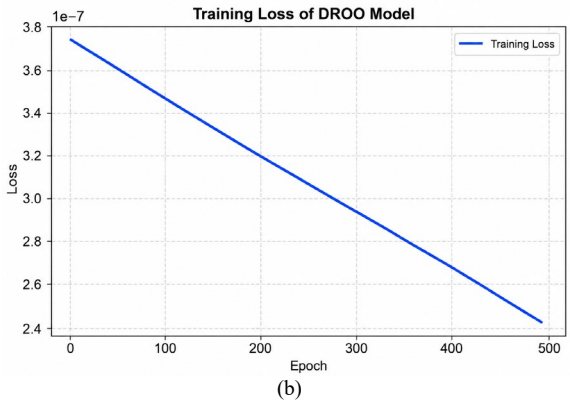


Fig. 10. Performance comparison of the proposed IDROO model under different learning rate settings.

(a)



(b)

(c)

Fig. 11. Training loss convergence of the proposed IDROO model for (a) $N = 10$, (b) $N = 20$, and (c) $N = 30$ wireless devices.

Fig. 11 depicts the training loss of the model. The X-axis shows the number of iterations the algorithm has undergone. The Y-axis represents the training loss, which quantifies the error or deviation of the model's prediction from the optimal value. This graph reflects the IDROO algorithm's learning process during task offloading. While the initial epochs involve exploration and result in higher loss, the stabilization of loss suggests that the algorithm has converted to a reliable offloading strategy. These curves show a monotonic decrease, indicating that the loss is steadily reducing over time.

The results in Table II indicate that the proposed IDROO outperforms existing DRL-based offloading schemes in all major performance metrics. Specifically, IDROO achieves approximately 18–25% higher computation rate compared to DROO and PPO, while reducing energy consumption by nearly 15–20%. Furthermore, the convergence time is significantly reduced due to the efficient learning and quantization strategy, making the proposed model more suitable for real-time MEC environments.

A comparative analysis of convergence behavior is further illustrated in Fig. 12, where the proposed IDROO demonstrates faster convergence and higher stability compared to other methods.

TABLE II. PERFORMANCE COMPARISON WITH EXISTING METHODS

Method	Avg. Computation Rate (×10⁶ bps)	Energy Consumption (J)	Convergence Time (Epochs)	Task Success Rate (%)
DROO	5.8	0.92	350	88.5
A3C	6.2	1.05	420	90.2
PPO	6.5	0.98	400	91.0
DDPG	5.5	0.95	300	87.3
Proposed IDROO	**7.4**	**0.78**	**220**	**94.6**

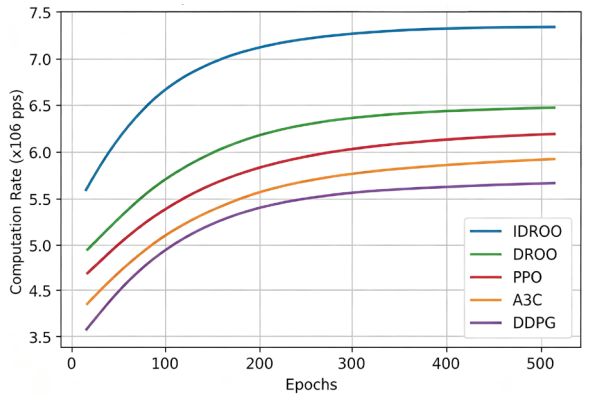


Fig. 12. Comparison of computation rate convergence versus epochs for DROO, A3C, PPO, DDPG, and the proposed IDROO framework.

To ensure the reliability of the results, each experiment was conducted over multiple independent runs, and the average performance metrics are reported. In addition, the standard deviation is calculated to evaluate the stability of the proposed approach. The results in Table III indicate that the IDROO framework not only achieves higher performance but also demonstrates consistent behavior across different runs.

TABLE III. STATISTICAL VALIDATION OF PERFORMANCE

Method	Avg. Computation Rate	Std. Dev
DROO	5.8	0.35
PPO	6.5	0.28
A3C	6.2	0.32
DDPG	5.5	0.30
IDROO	**7.4**	**0.21**

In addition to computation rate and energy consumption, other performance metrics such as end-to-end latency and task failure rate are important in practical MEC systems. Although these metrics are not explicitly evaluated in this work, the proposed optimization framework indirectly accounts for latency through computation rate and resource allocation decisions. Higher computation rates generally correspond to lower task completion delays. Similarly, efficient resource allocation contributes to reducing task failures caused by energy constraints or poor channel conditions.

A. Implementation Details

The proposed IDROO model is implemented using TensorFlow 1.0, following the structure of prior DRL-based MEC frameworks. Although TensorFlow 1.0 is deprecated, the implementation is framework-independent and can be easily reproduced using modern deep learning libraries such as TensorFlow 2.x or PyTorch.

The key hyperparameters used in the experiments are as follows: learning rate = 0.01, batch size = 32, memory size = 128, and training interval = 10. The neural network consists of two hidden layers with 128 and 64 neurons, respectively.

The experiments are conducted on a standard computing system with Intel Core i5 processor, 8 GB RAM, and no GPU acceleration.

The dataset used in this study is synthetically generated based on standard MEC simulation settings, where channel gains are modelled using random distance and fading distributions. This setup ensures controlled evaluation under varying network conditions.

B. Case Studies and Real-World Applications

To demonstrate the practical applicability of the proposed IDROO framework, consider a smart healthcare monitoring system where wearable devices collect patient data such as heart rate and ECG signals. These resource-constrained devices require real-time processing for timely detection of critical conditions.

In this setup, devices can either process data locally or offload it to nearby edge servers based on network conditions and energy availability. The IDROO framework dynamically makes this decision while optimizing transmit power and CPU frequency.

As a result, the proposed approach improves energy efficiency, reduces latency, and ensures reliable real-time performance, which is essential for healthcare applications.

VI. Conclusion and Future Work

In this work, an improved deep reinforcement learning-based framework, IDROO, was proposed for joint task offloading and resource allocation in mobile edge computing systems. The proposed approach integrates offloading decisions with transmit power and CPU frequency optimization, enabling efficient operation under dynamic wireless network conditions. Simulation results demonstrate that IDROO outperforms existing methods, achieving approximately 18–25% improvement in computation rate, 15–20% reduction in energy consumption, and faster convergence. The model also exhibits stable performance across varying numbers of wireless devices, highlighting its robustness and adaptability in dynamic MEC environments. The results indicate that jointly optimizing offloading decisions and resource allocation significantly enhances overall system efficiency.

Despite the promising performance of the proposed framework, several limitations remain. The current model introduces additional computational complexity due to joint optimization and assumes perfect channel state information, binary offloading, and negligible edge processing delay, which may not fully reflect practical MEC environments. Furthermore, user mobility and handover effects in dynamic 5G/6G scenarios are not explicitly considered. Future work will focus on developing lightweight and scalable models, incorporating partial task offloading, realistic delay modeling, imperfect channel conditions, and mobility-aware optimization. In addition, more advanced quantization techniques and fully end-to-end learning frameworks will be explored to further improve convergence speed, computational efficiency, and overall system performance.

Conflicts of Interest

The authors declare no conflicts of interest.

Author Contributions

Sweta Dash contributed to the conceptualization, methodology, software, validation, formal analysis, and investigation, and wrote the original draft; Sanjit Kumar Dash assisted with conceptualization, methodology, software, formal analysis, resources, original draft preparation, and manuscript review and editing; Jibitesh Mishra participated in methodology, software, investigation, and resources, and revised the manuscript; Suleman Alnatheer conducted validation and formal analysis and acquired funding; Mohammed Altaf Ahmed contributed to conceptualization, software, validation, formal analysis, funding acquisition, and manuscript editing; Abdullah Alsir Mohamed assisted with investigation and original draft writing; all authors read and approved the final manuscript.

Funding

This research was funded by Prince Sattam bin Abdulaziz University through the project number (PSAU/2025/01/33415).

Acknowledgement

The authors extend their appreciation to Prince Sattam bin Abdulaziz University for funding this research work.

References

[1] S. K. Mishra, H. K. Challa, K. S. Kotha, and D. P. Yarramreddy, "Task offloading technique selection in mobile edge computing," in *Proc. 2024 International Conference on Advancements in Smart, Secure and Intelligent Computing (ASSIC)*, 2024, pp. 1−6.

[2] W. Yang, Z. Liu, X. Liu, and Y. Ma, "Deep reinforcement learning-based low-latency task offloading for mobile-edge computing networks," *Applied Soft Computing*, vol. 166, 112164, 2024.

[3] A. M. Seid, G. O. Boateng, B. Mareri, G. Sun, and W. Jiang, "Multi-agent DRL for task offloading and resource allocation in multi-UAV enabled IoT edge network," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4531−4547, 2021.

[4] L. Zang, X. Zhang, and B. Guo, "Federated deep reinforcement learning for online task offloading and resource allocation in WPC-MEC networks," *IEEE Access*, vol. 10, pp. 9856−9867, 2022.

[5] A. Sacco, F. Esposito, G. Marchetto, and P. Montuschi, "Sustainable task offloading in UAV networks via multi-agent reinforcement learning," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 5, pp. 5003−5015, 2021.

[6] E. Mustafa, J. Shuja, K. Bilal, S. Mustafa, T. Maqsood, F. Rehman, and A. U. R. Khan, "Reinforcement learning for intelligent online computation offloading in wireless powered edge networks," *Cluster Computing*, vol. 26, no. 2, pp. 1053−1062, 2023.

[7] I. Ullah, H. K. Lim, Y. J. Seok, and Y. H. Han, "Optimizing task offloading and resource allocation in edge-cloud networks: A DRL approach," *Journal of Cloud Computing*, vol. 12, no. 1, 112, 2023.

[8] J. Heydari, V. Ganapathy, and M. Shah, "Dynamic task offloading in multi-agent mobile edge computing networks," in *Proc. 2019 IEEE Global Communications Conference*, 2019, pp. 1−6.

[9] Y. Chen, J. Xu, Y. Wu, J. Gao, and L. Zhao, "Dynamic task offloading and resource allocation for Noma-aided mobile edge computing: An energy efficient design," *IEEE Transactions on Services Computing*, pp. 1492−1503, 2024.

[10] M. D. Hossain, T. Sultana, M. A. Hossain, M. I. Hossain, L. N. Huynh, J. Park, and E. N. Huh, "Fuzzy decision-based efficient task offloading management scheme in multi-tier MEC-enabled networks," *Sensors*, vol. 21, no. 4, pp. 1484, 2021.

[11] F. Zhang, G. Han, L. Liu, M. Martínez-García, and Y. Peng, "Deep reinforcement learning based cooperative partial task offloading and resource allocation for IIoT applications," *IEEE Transactions on Network Science and Engineering*, vol. 10, no. 5, pp. 2991−3006, 2022.

[12] Y. Wu *et al*., "Deep reinforcement learning-based video quality selection and radio bearer control for mobile edge computing supported short video applications," *IEEE Access*, vol. 17, no. 4, pp. 181740−181749, 2019.

[13] L. Huang, S. Bi, and Y. J. A. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Transactions on Mobile Computing*, vol. 19, no. 11, pp. 2581−2593, 2019.
[14] M. Tang and V. W. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Transactions on Mobile Computing*, vol. 21, no. 6, pp. 1985−1997, 2020.
[15] C. Yu, Y. Yong, Y. Liu, J. Cheng, and Q. Tong, "A multi-objective evolutionary approach: Task offloading and resource allocation using enhanced decomposition-based algorithm in mobile edge computing," *IEEE Access*, vol. 12, pp. 123640−123655, 2024.
[16] X. Tan, D. Zhao, M. Wang, X. Wang, X. Wang, W. Liu, and M. G. Arani, "A decision-making mechanism for task offloading using learning automata and deep learning in mobile edge networks," *Heliyon*, vol. 10, no. 1, 23651, 2024.
[17] X. Dai, Z. Xiao, H. Jiang, M. Alazab, J. C. Lui, S. Dustdar, and J. Liu, "Task co-offloading for D2D-assisted mobile edge computing in industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 1, pp. 480−490, 2022.
[18] S. Dong, Y. Xia, and J. Kamruzzaman, "Quantum particle swarm optimization for task offloading in mobile edge computing," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 8, pp. 9113−9122, 2022.
[19] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 856−868, 2018.
[20] S. Yang, G. Lee, and L. Huang, "Deep learning-based dynamic computation task offloading for mobile edge computing networks," *Sensors*, vol. 22, no. 11, 4088, 2022.
[21] D. Chen, F. B. Zan, X. Liu, and X. Meng, "DROO-C data transfer in a mobile edge computing offloading framework," in *Proc. 2023 13th Int. Conf. Communication and Network Security*, 2023, pp. 282−293.
[22] B. Kumar, A. Bhardwaj, and D. Sahu, "Machine learning techniques for task offloading in connected autonomous vehicles," in *Proc. 2024 IEEE Third International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES)*, 2024, pp. 400−405.
[23] X. Deng, J. Yin, P. Guan, N. N. Xiong, L. Zhang, and S. Mumtaz, "Intelligent delay-aware partial computing task offloading for multiuser industrial internet of things through edge computing," *IEEE Internet of Things Journal*, vol. 10, no. 4, pp. 2954−2966, 2021.
[24] Y. Chen, J. Zhang, and K. B. Letaief, "Dynamic task offloading for mobile edge computing with deep reinforcement learning," *IEEE Trans. Wireless Commun*., vol. 18, no. 8, pp. 4093–4106, 2019.
[25] J. Wang, L. Zhao, J. Liu, and N. Kato, "Smart resource allocation for mobile edge computing: A deep reinforcement learning approach," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1529−1541, 2019.
[26] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795−2808, 2015.
[27] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *Proc. IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, 2018, pp. 207−215.
[28] Z. Ning, P. Dong, X. Wang, J. J. Rodrigues, and F. Xia, "Deep reinforcement learning for vehicular edge computing: An intelligent offloading system," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 6, pp. 1−24, 2019.
[29] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-edge AI: Intelligentizing mobile edge computing, caching and communication by federated learning," *IEEE Network*, vol. 33, no. 5, pp. 156−165, 2019.
[30] M. Chen, Y. Hao, Y. Li, C. F. Lai, and D. Wu, "On the computation offloading at ad hoc cloudlet: Architecture and service modes," *IEEE Communications Magazine*, vol. 53, no. 6, pp. 18−24, 2015.