IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 1 May 2025, accepted 16 May 2025, date of publication 26 May 2025, date of current version 4 June 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3573929

RESEARCH ARTICLE

Deep Reinforcement Learning With Dueling DQN for Partial Computation Offloading and Resource Allocation in Mobile Edge Computing

EHZAZ MUSTAFA[1], JUNAID SHUJA[2], (Member, IEEE), FAISAL REHMAN[1], ABDALLAH NAMOUN[3], (Member, IEEE), MAZHAR ALI[1], AND ABDULLAH ALOURANI[4], (Member, IEEE)

[1]Department of Computer Science, COMSATS University Islamabad, Abbottabad Campus, Abbottabad 22060, Pakistan
[2]Department of Computer Science, Southeast Missouri State University, Cape Girardeau, MO 63701, USA
[3]AI Center, Faculty of Computer and Information Systems, Islamic University of Madinah, Madinah 42351, Saudi Arabia
[4]Department of Management Information Systems, College of Business and Economics, Qassim University, Buraydah 51452, Saudi Arabia

Corresponding author: Abdullah Alourani (ab.alourani@qu.edu.sa)

The Researchers would like to thank the Deanship of Graduate Studies and Scientific Research at Qassim University for financial support (QU-APC-2025).

ABSTRACT Computation offloading transfers resource-intensive tasks from local Internet of Things (IoT) devices to powerful edge servers, which minimizes latency and reduces the computational load on IoT devices. Deep Reinforcement Learning (DRL) is widely utilized to optimize computation offloading decisions. However, previous studies fall short in two main ways: firstly, they do not collectively optimize the comprehensive state space, and secondly, their reliance on Q-learning and Deep Q Networks (DQN) makes it challenging for agents to discern the optimal action in large action spaces, as many actions may possess similar values. In this paper, we introduce a multi-branch Dueling Deep Q Network (MBDDQN) that tackles the challenges of high-dimensional state-action spaces and long-term cost optimizations in dynamic environments. The Dueling DQN alleviates the complexity of simultaneous offloading and resource allocation decisions, with each branch independently controlling a subset of the decision variables to scale efficiently with an increasing number of IoT devices, thereby avoiding the combinatorial explosion of potential actions. Furthermore, we implement a long short-term memory (LSTM) network with distinct advantage-value layers to enhance both short-term action selection and long-term system cost estimation, as well as improve the temporal learning capacity of the model. Finally, we propose an innovative adaptive cost-weighting mechanism within the reward function to dynamically balance competing objectives, including energy consumption, latency, and bandwidth utilization. Unlike prior works that use fixed reward structures, we leverage weighted state-action advantage values to dynamically adjust the optimization variables. This approach also enables the agent to self-tune, allowing it to prioritize delay minimization in delay-sensitive scenarios and energy conservation in resource-constrained environments. Simulation results demonstrate the superiority of the proposed scheme compared to benchmarks. For instance, MBDDQN reduces delay by 17.88% over DQN and 12.28% over DDPG. Additionally, regarding energy consumption, MBDDQN achieves a 10.1% improvement over DQN and a 7.64% enhancement over DDPG.

INDEX TERMS Computation offloading, deep Q network (DQN), edge computing, Internet of Things (IoT), resource allocation, reinforcement learning.

The associate editor coordinating the review of this manuscript and approving it for publication was Mehdi Sookhak.

I. INTRODUCTION

Mobile Edge Computing (MEC) succeeds cloud computing by positioning storage and computation resources closer to

the Internet of Things devices (IoT) devices at the network edge [1], [2]. This results in faster computation services compared to local processing. The integration of computation offloading with MEC enables IoT devices to effectively process their demanding workloads on nearby edge servers, minimizing latency and reducing the computational load on IoT devices, thereby conserving energy [4], [5]. Computation offloading encounters several challenges, specifically optimal offloading decisions [6]. These decisions encompass various strategies, including full offloading, partial offloading, and binary offloading [7]. In full offloading, all tasks are transferred to the edge server without taking environmental insights into account. Binary offloading involves the task being executed either locally at the IoT device or entirely at the edge server. The concept of partial offloading strikes a balance, leveraging both the resources of IoT devices and edge servers to execute the task. The second challenge faced by computation offloading encompasses the optimization of resource allocation, particularly in scenarios involving offloading in dynamic networks. A considerable example is the allocation of bandwidth to facilitate the offloading process [8].

There is extensive literature that enhances computation offloading decisions and resource allocation, including numerical optimization, deterministic approaches, and game theory-based optimization [9], [10]. However, these methods often fall short in practical MEC scenarios, failing to meet the requirements of IoT devices for effective offloading decisions, specifically in dynamic networks [11]. Machine learning-based techniques, such as supervised learning, offer superior results with their efficient learning processes. Deep learning also aids in optimizing the offloading process [12]. However, these approaches require substantial training data, which proves impractical in dynamic MEC networks characterized by fluctuating channels, random task arrivals, and varying resource needs.

Deep Reinforcement Learning (DRL) learns through trial and error in real-time, hence, practical in MEC settings where it optimizes offloading decisions and resource allocation while interacting with the environment [13], [14]. DRL effectively outperforms traditional numerical optimization and machine learning-based offloading methods [15], [16], [17], [18]. While effective, DRL encounters multiple challenges such as the collective optimization of the comprehensive state space and the difficulty for the agent to discern the optimal action in large action spaces due to the use of Q-learning and Deep Q Networks (DQN) as many actions may have similar values, especially with an increasing number of IoT devices [19].

Different from prior approaches that rely on traditional Q-learning and DQN architectures, we propose the following novel contributions.

- We propose a novel multi-branch Dueling Deep Q Network (MBDDQN) which is specifically designed to address the dual challenge of long-term cost optimization and high-dimensional state-action spaces in dynamic MEC environments. MBDDQN disentangles the complexity of simultaneous offloading and resource allocation decisions for more granular optimization of computation offloading and resource allocation. Each branch independently handles a subset of the decision variables, which allows the model to scale efficiently with an increasing number of IoT devices, while avoiding the combinatorial explosion of potential actions. This decomposition strategy generates independent, fine-tuned decisions for partial task offloading and bandwidth allocation in heterogeneous environments with varying task demands and network conditions.
- To enhance the temporal learning capacity of the model, we employ an LSTM network with distinct advantage-value layers to improve both short-term action selection and long-term system cost estimation.
- Finally, we employ a novel adaptive cost-weighting mechanism in the reward function to dynamically balance competing objectives, including energy consumption, latency, and bandwidth utilization. Unlike traditional approaches that use fixed reward structures, this reward-shaping mechanism leverages weighted state-action advantage values to adjust the optimization focus dynamically, depending on the system's real-time requirements. It also provides the agent with a self-tuning mechanism to prioritize latency or delay minimization in latency-critical situations, energy conservation in resource-constrained environments and allows the system to prioritize different objectives depending on the tasks and state of the network.

The rest of the article is organized as follows: Section II provides the related research. Section III presents our system model. Section IV provides the problem formulation. In Section V, we present the methodology and architecture of our proposed framework. The performance evaluation is discussed in Section VI, and finally, Section VII concludes the article.

II. RELATED WORK

Computation offloading is generally optimized with various approaches such as stochastic optimization, game theory-based optimization, and machine learning-based optimization [20], [21], [22]. The authors in [23] considered a multi-user network and provided a binary offloading scheme where the task is executed either locally or at the edge server. To deal with combinatorial optimization, the authors used a decoupled optimization using the Coordinate Descent method and alternating direction method of multipliers for mode selection and computation rate maximization. Authors in [24] considered a stochastic optimization and provided a stochastic game-based computation offloading model. They also proved that multi-user stochastic games can achieve the Nash Equilibrium in the proposed system. To deal

with high computational latency and energy consumption, Gao et al., [25] provided a solar energy harvesting and MEC-based computation offloading scheme. To solve the joint stochastic optimization problem, the authors used Lyapunov optimization for offloading decisions and resource allocation.

Game theory-based computation offloading also enhances the performance of MEC networks [26]. Researchers in [27] investigated the quality of services-oriented computation offloading problems for the Internet of Things. They considered a game-based computation offloading scheme and provided a non-cooperative game among the Internet of Things devices to enhance the computation offloading process, and also proved that there is a Nash Equilibrium for the proposed game. Similarly, in [28], the authors provided a game theory-based model to achieve an efficient computation offloading model for dynamic vehicular networks. They proved that heuristic schemes often falter in such dynamic networks. First, the authors provided a two-stage game theory-based model for making offloading decisions, and then utilized a neural network to predict the behaviors of the game theory model for offloading decisions in a more effective manner.

Machine learning is another viable solution to enhance the computation offloading and resource allocation process [29]. Erfan et al. investigated a machine learning-based offloading scheme while accounting for uncertainties in user mobility [30]. They formulated this problem as NP-hard and addressed it by designing two novel offloading approaches: S-OAMC and G-OAMC. These approaches fully assign applications to cloudlets by considering their expected future locations and specifications predicted by Matrix Completion. The results showed that the proposed scheme achieves near-optimal turnaround times for applications. Similarly, a task offloading decision and resource allocation scheme using deep learning and seagull optimization is proposed in [31]. The proposed scheme addresses resource allocation and optimal offloading decisions to reduce the overall system cost. Their results concluded that the proposed scheme effectively reduces the system cost while achieving a maximum reward of 0.8967.

Among all the solutions mentioned above, DRL is recognized as the superior method for optimizing offloading decisions and resource allocation due to its trial-and-error learning ability in dynamic networks [32]. In [33], the author investigated a multiple input multiple output (MIMO) based edge network to reduce the power consumption of and offloading latency under stochastic environments. To obtain the optimal offloading decisions with prior knowledge of the dynamic environment, the authors used DRL with DQN. Using reinforcement learning, authors in [34] investigated a binary offloading scheme under dynamic networks. They obtain an optimal offloading decision with a minimum execution latency using conventional neural networks. In [39], the authors proposed a multi-agent DRL-based computation offloading approach for uncertain load dynamics at edge nodes. The authors made optimal offloading decisions for delay-sensitive tasks without prior knowledge of the task decisions of other nodes. Similarly, the authors in [40] proposed a combined fuzzy and DRL-based task scheduling method. The aim was to optimize the makespan, energy consumption, fault tolerance, and cost. To this end, the authors implemented the dynamic prioritization of tasks and the real-time change of scheduling rules. In [41], the author provided a proximal policy optimization method computation offloading scheme that considers the intrinsic task dependencies and trivial deadlines for vehicular tasks to enhance performance. To address the problem of task dependency and resource competition, the authors used a DDPG-based DRL algorithm to optimize the offloading performance. They modeled the task dependency, priority, and resource consumption and utilized an LSTM-based recurrent neural network (RNN) algorithm to improve the performance [42]. In [43], the authors investigated the assignment of heterogeneous servers to real-time tasks. To this end, they proposed a novel suitability-based adaptive resource selection to analyze server characteristics and tasks in MEC. In [44], the authors proposed a machine learning based resource allocation mechanism for urban traffic flow prediction in the edge-cloud continuum to improve energy. Similarly, the novel traffic flow prediction mechanism was explored in [45] with attention-driven deep hybrid neural networks. To improve consumer trust, the authors proposed an entropy-based multi-criterion decision-making approach to assist personalization and to improve user interfaces [46].

Unlike all previous approaches, we propose MBDDQN specifically for long-term cost optimization in dynamic MEC environments. By decomposing the complexity of offloading and resource allocation, each branch optimizes specific decision variables independently. This design enables granular task offloading and resource allocation, efficiently scaling with IoT devices while adapting to varying demands and conditions. In Table 1, we provide a summary of related work and its comparison with the proposed approach. We propose a novel approach using a multi-branch architecture along with LSTM with distinct advantage-value layers to improve long-term system cost estimation and to enhance the temporal learning capacity of the model. Moreover, we incorporate an adaptive cost weighting mechanism to dynamically adjust the optimization variables which allows the agent in self-tuning to prioritize delay minimization and energy conservation.

III. SYSTEM MODEL

Consider Figure 1, where we consider a small cell network with multiple edge servers and multiple IoT devices. These devices are represented as $\mathcal{W} = \{1, 2, 3, \ldots, W\}$. Each $\mathcal{W}_i$ is positioned randomly within the network. These devices possess distinct computing capabilities, denoted as $\mathcal{F}_{\mathcal{W}} = \{f_1, f_2, f_3, \ldots, f_n\}$. We assume that there are multiple MEC servers, and their set can be expressed as $\mathcal{M} = \{1, 2, \ldots, M\}$. These MEC servers are embedded within base stations,

TABLE 1. Summary of related work.

Study	Focus of Study	Machine Learning	Dueling DQN	LSTM for temporal learning	Multi-Branch Network	Adaptive Cost Mechanism	Bandwidth Allocation	Granularity of Decision Making
[23]	Multi-user network and Binary offloading	✗	✗	✗	✗	✗	✗	✓
[24]	Multi-user IoT for Stochastic offloading	✗	✗	✗	✗	✗	✗	✓
[25]	Lyapunov optimization for energy harvesting offloading	✗	✗	✗	✗	✗	✗	✗
[26]	Game theory-based computation offloading	✗	✗	✗	✗	✗	✗	✗
[27]	Non-cooperative game based offloading	✗	✗	✗	✗	✗	✗	✗
[28]	Two-stage game model for offloading	✗	✗	✗	✗	✗	✗	✓
[29]	Machine learning and Mobility based offloading	✓	✗	✗	✗	✗	✗	✗
[30]	Machine learning based offloading with uncertainty	✓	✗	✗	✗	✗	✗	✗
[31]	Deep learning and seagull optimization for offloading	✓	✗	✗	✗	✗	✗	✗
[33]	Multiple input multiple output based offloading	✓	✓	✗	✗	✗	✗	✓
[34]	Convolution neural network based offloading	✓	✓	✗	✗	✗	✗	✓
[38]	UAV-based cooperative target search	✓	✗	✗	✓	✗	✗	✓
[35]	Multi-branch network based offloading	✓	✗	✗	✓	✗	✗	✓
[39]	Multi-agent DRL based offloading in edge-cloud	✓	✓	✓	✗	✗	✗	✗
[40]	Hybrid fuzzy logic and DRL based task scheduling	✓	✗	✗	✗	✗	✗	✗
This work	MBDDQN and LSTM based offloading	✓	✓	✓	✓	✓	✓	✓

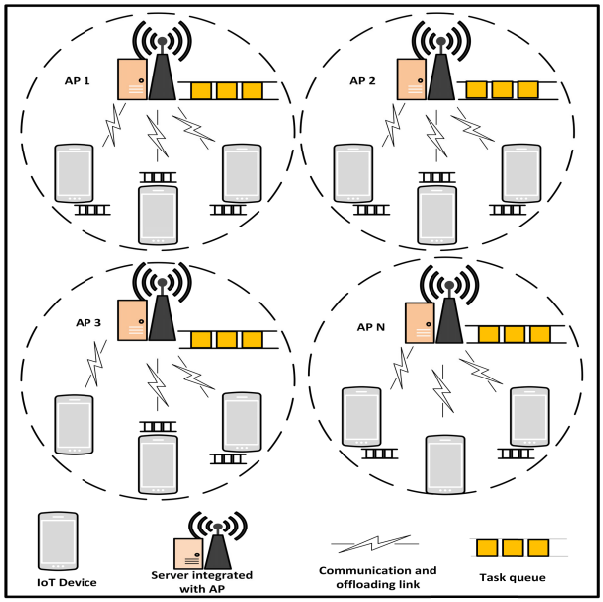

FIGURE 1. System model.

proximate to $\mathcal{W}$, providing wireless access and computational cycles. The computing power of the servers is denoted as $\mathcal{F}_{\mathrm{M}} = \{f_{\mathrm{M}1}, f_{\mathrm{M}2}, f_{\mathrm{M}3}, \ldots, f_{\mathrm{M}n}\}$.

In our system model, we use a time frame with length T_f, represented as $t = \{1, 2, 3, \ldots, \tau\}$, where τ signifies the number of total time frames. In each time frame, each $\mathcal{W}_i$ has the probability of generating tasks that must be determined, representing the ratio of the task to be executed either at the MEC server or at the local device. In time frame t, all the tasks are represented as $\mathcal{T}_t = \{\mathcal{T}_{t1}, \mathcal{T}_{t2}, \ldots, \mathcal{T}_{tn}\}$. We assume that both $\mathcal{M}$ and $\mathcal{W}$ have a task queue, operating on a priority basis. $\mathcal{W}_i$ maintain this queue and execute the tasks based on priority. Each $\mathcal{M}_i$ contains two queues including a waiting queue and a running queue. Tasks wait in the waiting queue and are transferred to the running queue when the required computing resources are met.

The process of computation offloading involves offloading decisions and resource allocation. Here, offloading decisions determine the ratio of tasks being executed, while resource allocation indicates the proportion of optimal bandwidth. In each time frame t, the offloading decision for each $\mathcal{W}_i$ is denoted as x_{ti}. Therefore, the system's offloading decision can be represented as $\mathbf{X}_t = \{x_{t1}, x_{t2}, \ldots, x_{tn}\}$. Similarly, the resource allocation for $\mathcal{W}_i$ can be denoted as y_{ti}. The overall allocated resources for the system at time frame t can be represented by $\mathbf{Y}_t = \{y_{t1}, y_{t2}, \ldots, y_{tn}\}$.

We implement task queue in a priority fashion and maintain at both $\mathcal{M}$ and $\mathcal{W}$. On the MEC side, there are two

TABLE 2. Notation and symbols.

Symbol	Meaning
R_t	Immediate reward received after taking action A_j in state S_j at time t
S_j	State at time step j
A_j	Action taken at time step j
S_{t+1}	Subsequent state after taking action A_j in state S_j at time t
$Q_d(S_j, A_j, d)$	Q-value function for action dimension d estimated by the online network in state S_j with action A_j
$Q'_d(S_{t+1})$	Q-value function for the next state S_{t+1} estimated by the target network
y'_d	Target Q-value for branch d
$\mathcal{L}_{oss}$	Loss function quantifying the difference between target Q-values (y'_d) and predicted Q-values ($Q_d(S_j, A_j, d)$)
N	Number of action dimensions
$\mathbb{E}$	Expectation operator
$\delta(i)$	Cumulative difference between expected reward and predicted Q-value in each branch of the network
$p(i)$	Probability of selecting the i-th tuple from the experience buffer
$w(i)$	Weight assigned to each tuple
α_0	Parameter controlling the priority assigned to each tuple
β_0	Parameter influencing the likelihood of a tuple being chosen
ζ	Small positive value introduced for feasibility in probability calculations
Γ_0	Parameter controlling the importance of weight adjustments
τ_{off}	Application offloading time
τ_{queue}	Queuing time
τ_{exec}	Execution time
τ_{trans}	Data transferring time
τ_{down}	Result downloading time
τ_{max}	Task deadline
t_{ready}	Ready time
τ_{upload}	Uploading time
τ_{exec_ready}	Execution ready time
$res(task)_i$	Task result
τ_{comp}	Task completion time
$src(edge)$	Source edge server
$dest(edge)$	Destination edge server
γ	Discount factor used to consider future rewards in the target Q-value computation

queues named as waiting queue and the running queue. The waiting queue at the $\mathcal{M}$ side is depicted as $\mathcal{Q}_{\text{MEC,waiting}} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_G\}$, where G represents the total task residing in waiting queue. The running queue at the MEC side is depicted as $\mathcal{Q}_{\text{MEC,execution}} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_H\}$, where H denotes the total tasks within the running queue. These tasks initially wait in a waiting queue till the availability of computing requirements at the MEC server. At the $\mathcal{W}$ side, we implement a single queue represented as $\mathcal{Q}_{\text{local}} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k\}$, where K denotes the maximum task in the queue.

A. LOCAL COMPUTATION MODEL

When a $\mathcal{W}_i$ has enough computing power and battery life, it can handle tasks on its IoT device without needing to offload at the edge server. The current study adopts a perspective where each computation workload is comprised of sub-workloads, which may either be completed by the $\mathcal{W}_i$ itself or dispatched to an edge server. To establish an optimal approach, an offloading ratio, designated as $g_w(t)$, is specified for each computation task associated with $\mathcal{W}_i$.

At a specific time slot t and for a given $\mathcal{W}_i$, the determination of CPU cycles for local processing ($f_w(t)$) involves a calculated formula. This formula is built upon the assigned power for local execution ($P_{l,w}(t)$) and takes into account the impact of an effective switched capacitance parameter j. The value of k is intrinsically linked to the chip architecture inherent in $\mathcal{W}_i$. The formula is given by:

$$f_w(t) = \sqrt{\frac{P_{l,w}(t)}{k \cdot c_w(t) \cdot (1 - g_w(t))}} \tag{1}$$

Moreover, we clarify the computation of the time required for a task to be performed locally, denoted as $d_{l,w}(t)$. This calculation is rooted in the interaction between the offloading ratio $g_w(t)$, the number of CPU cycles required for computation $c_w(t)$, and the allocation of CPU cycles $f_w(t)$.

$$d_{l,w}(t) = \frac{(1 - g_w(t)) \cdot c_w(t)}{f_w(t)} \tag{2}$$

As a logical progression, the quantification of energy consumption persisted by $\mathcal{W}_i$ during this local processing activity is encapsulated. This energy consumption is denoted by the product of the power dedicated to local execution ($P_{l,w}(t)$) and the execution time ($d_{l,w}(t)$). The energy consumption is given by:

$$E_{l,w}(t) = P_{l,w}(t) \cdot d_{l,w}(t) \tag{3}$$

B. OFFLOADING COMPUTATION MODEL

When computational workloads are designated for offloading, the activated edge servers are assigned to handle these offloaded tasks. As we assume fixed computational resources for MEC servers, this assumption simplifies the analysis and focuses on the task offloading and resource allocation

decisions, assuming that the computational capacity of the servers is sufficient to handle the offloaded tasks. However, in real-world MEC environments, resource availability can fluctuate due to varying loads, server contention, or dynamic allocation strategies. It's important to note that the sub-workloads of $\mathcal{W}$ are independent, and no interference occurs between them. The duration required to transmit computation workloads from $\mathcal{W}$ to the active edge server $\mathcal{M}$ is determined by the formula:

$$d^{\text{tran}}_{m;w}(t) = \frac{g_w(t) \cdot t_{\text{sw}}(t)}{r_{m;w}(t)} \tag{4}$$

Here, $g_w(t)$ represents the offloading ratio of the workload, $t_{\text{sw}}(t)$ signifies the transmission time of the computation workload for $\mathcal{W}$, and $r_{m;w}(t)$ represents a factor specific to the transmission. Here, $r_{m;w}(t)$ is determined by the following relationship:

$$r_{m;w}(t) = f(Q_m(t), C(t), T_{task}) \tag{5}$$

Specifically, it can be expressed as:

$$r_{m;w}(t) = \alpha_1 Q_m(t) + \alpha_2 \left(\frac{C(t)}{N_{ac}(t)}\right) + \alpha_3 \frac{T_{task}}{D_{max}} \tag{6}$$

where $Q_m(t)$ represents the channel quality. $C(t)$ is the network capacity. $N_{ac}(t)$ is the number of active IoT devices. T_{task} is the size of the task. As the task sizes can be different, D_{max} is the maximum task size and $\alpha_1, \alpha_2, \alpha_3$ are weighting factors reflecting the importance of each component. The values of $\alpha_1, \alpha_2, \alpha_3$ are not fixed, rather, we use empirical justification and determine based on the network environment. These parameters are adjusted adaptively to maintain a balance between different factors influencing the offloading decisions.

The energy consumption related to transmitting the computing data from $\mathcal{W}$ to MEC server $\mathcal{M}$ is defined as:

$$E^{\text{tran}}_{m;w}(t) = P_{m;w}(t) \cdot d^{\text{tran}}_{m;w}(t) \tag{7}$$

where $P_{m;w}(t)$ denotes the transmission power for offloading the workloads from $\mathcal{W}$ to MEC server $\mathcal{M}$.

For execution time required to complete the received workloads at MEC server $\mathcal{M}$, we introduce a time factor n_1 for task processing at the MEC server, represented as an M-dimensional vector $n_1 = [n_{1,1}, n_{1,2}, \ldots, n_{1,M}]^T$. This factor is proportional to $t_{\text{sw}}(t)$. The computation task execution time for $\mathcal{W}$ at MEC server $\mathcal{M}$ can be expressed as:

$$d^{\text{exec}}_{m;w}(t) = n_{1,m} \cdot g_w(t) \cdot t_{\text{sw}}(t) \tag{8}$$

Additionally, the energy consumption related to computation workloads of $\mathcal{W}$ at MEC server $\mathcal{M}$ is given by:

$$E^{\text{exec}}_{m;w}(t) = P_{m;w}(t) \cdot d^{\text{exec}}_{m;w}(t) \tag{9}$$

The overall time required for computation offloading from $\mathcal{W}$ can be calculated as:

$$d_{m;w}(t) = d^{\text{tran}}_{m;w}(t) + d^{\text{exec}}_{m;w}(t) \quad (12) \tag{10}$$

Subsequently, the total energy consumption associated with computation offloading for $\mathcal{W}$ is obtained as:

$$E_{m;w}(t) = E^{\text{tran}}_{m;w}(t) + E^{\text{exec}}_{m;w}(t) + E^{\text{wait}}_{m;w}(t) \quad (13) \tag{11}$$

In this equation, $E_{\text{wait},m;w}(t)$ represents the cost of waiting for execution by the MEC server, and n_2 signifies a constant coefficient. To obtain the waiting energy cost, we use Little's law, and can be expressed as:

$$E^{\text{wait}}_{m;w}(t) = n_{2,m} \cdot L_m(t) \quad (14) \tag{12}$$

where $L_m(t)$ denotes the number of tasks in the queue of MEC server $\mathcal{M}$ at time t.

IV. PROBLEM FORMULATION

We provide the problem formulation based on multiple constraints. We aim to deploy a multi-branch Dueling DQN to generate optimal offloading decisions and resource allocation to minimize energy consumption and latency for all IoT devices over time. To overcome stochastic task arrivals, channel states, and resource availability, the problem must account for these time-dependent variations. Based on previous equations and modeling, we formulate the problem as:

$$\begin{aligned}
\textbf{P1}: \min_{g_w(t), s_w(t)} \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} (\alpha E_w(t) + \beta L_t) \\
C1: \quad d_{l,w}(t) \le T_{w,\max}(t), \quad d_{m,w}(t) \le T_{w,\max}(t), \\
C2: \quad 0 \le s_w(t) \le 1, \quad \sum_{w=1}^{W} s_w(t) = 1, \\
C3: \quad 0 \le g_w(t) \le 1, \\
C4: \quad 0 \le f_w(t) \le f_{\max}(t).
\end{aligned} \tag{13}$$

Here, $E_w(t)$ denotes the energy consumption, and L_t denotes the latency or delay. A set of constraints guides the optimization process for the proposed system. Here α and β are not fixed. We use empirical justification and determine based on the network environment. These parameters are adjusted adaptively to maintain a balance between different factors influencing the offloading decisions. *C1* ensures that the delay of local execution and offloaded processing for a specific IoT device at time t does not exceed the predefined maximum delay, which may vary over time. *C2* maintains the ratio of bandwidth assigned to $\mathcal{W}$ within the range of 0 and 1 at each time slot t, optimizing resource allocation dynamically. *C3* restricts the offloading ratio of workloads from $\mathcal{W}$ to the 0-1 range, ensuring a balanced distribution of tasks across time. Lastly, *C4* enforces the allocation of CPU cycles for local processing to remain within the local processor's capacity $f_{\max}(t)$, which can vary over time based on the current processing load.

We aim to reduce the cost of both latency and energy consumption while adhering to the mentioned constraints. Based on the optimization problem described above, we deduce

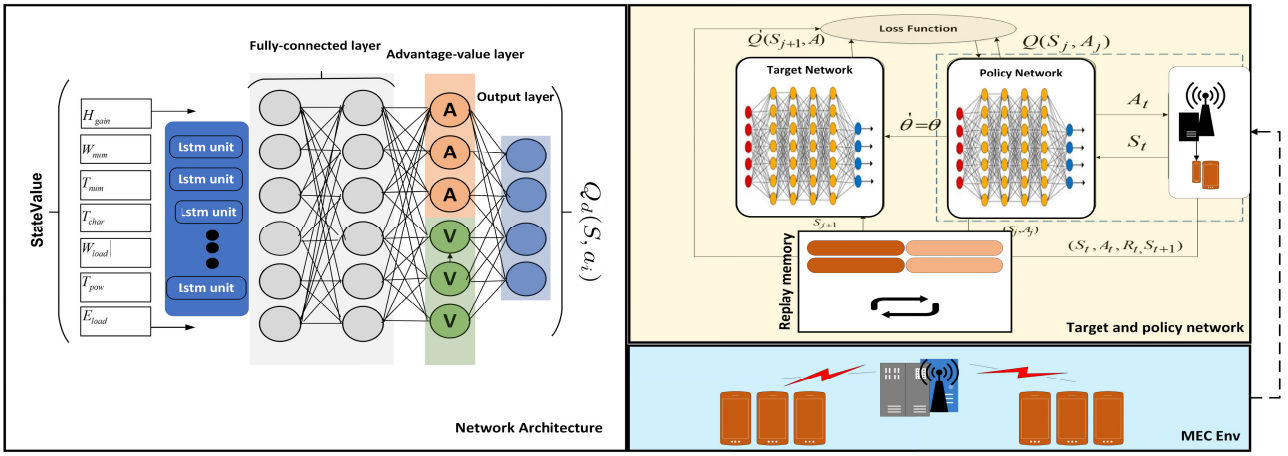

FIGURE 2. Proposed MBDDQN along with network architecture.

that addressing this issue involves determining the optimal offloading decisions and resource allocations for each time frame. Traditional optimization methods suffer from scalability issues when the number of IoT devices increases. Hence we use a multi-branch Dueling DQN that can be particularly useful in scenarios where a single Q-network struggles to represent the complexities of the environment adequately. It enables more focused learning for different aspects of the environment, potentially leading to better overall performance and more efficient training.

V. PROPOSED SOLUTION

Our primary goal is to achieve a reduction in both latency and energy consumption. To achieve this objective, we formulate our problem as MDP to provide a structured framework. In our approach, we employ a set of distinct branches, each operating as an independent network. This architectural arrangement is designed to effectively handle the issue of an exponentially expanding action space, which in turn makes the learning process of the algorithm challenging. Employing multiple branches allows the system to more effectively handle the exponential growth in the potential actions.

A. STATE SPACE DEFINITION

At the beginning of each time slot, the state space provides the system information observed by the agent. This state, denoted as $\mathbf{S}_t$, encompasses various parameters crucial to the decision-making process. Specifically, it is composed of the following elements: $\mathbf{S}_t = \{\mathcal{H}_{\text{gain}}, \mathcal{W}_{\text{num}}, \mathcal{T}_{\text{num}}, \mathcal{T}_{\text{char}}, \mathcal{W}_{\text{load}}, \mathcal{T}_{\text{pow}}, \mathcal{E}_{\text{load}}\}$

1) $\mathcal{H}_{\text{gain}}$ (*Channel gain*): This reflects the real-time fluctuations and attenuation in signal strength as data is transferred between different network devices, impacting optimal offloading decisions for efficient resource utilization.
2) $\mathcal{W}_{\text{num}}$ (*Number of Wireless devices*): This component quantifies the count of IoT devices present within the system. It offers a measure of IoT devices at a given time slot, impacting the overall load on the system.
3) $\mathcal{T}_{\text{num}}$ (*Task Generation*): This value indicates the volume of tasks generated during the time slot t. It characterizes the workload of the system, influencing the computational load and resource requirements.
4) $\mathcal{T}_{\text{char}}$ (*Task Characteristics*): This parameter provides the individual task characteristics within the time frame. Each task denoted as $\mathcal{T}_{ti}$, comprises two fundamental attributes including processing demands and data size. These attributes define the complexity and data requirements of the tasks.
5) $\mathcal{W}_{\text{load}}$ (*IoT device Load Information*): This parameter provides information about the load of IoT device. It includes details regarding the computational capacity and task queue length for each IoT device, offering insights into their processing capabilities and pending tasks.
6) $\mathcal{T}_{\text{pow}}$ (*Transmission power*): This parameter involves the utilization of transmission power. It influences communication quality and energy consumption within the system, providing a role in guiding informed decisions about task offloading and resource allocation.
7) $\mathcal{E}_{\text{load}}$ (*Edge Server Load Information*): This component involves load concerning the edge servers. It encompasses parameters such as computational capacity and task queue length for each edge server, providing a comprehensive view of their resource availability. Overall, the state $\mathbf{S}_t$ represents the dynamics of the proposed system, including IoT devices counts, task generation, task characteristics, IoT devices and edge server load statuses, and transmission power. This diverse decision-making process enables the agent to efficiently determine optimal task offloading and resource allocation strategies in a dynamic environment.

B. ACTION SPACE

The action space provides the set of all possible actions that an agent can take within a given environment or decision-making framework. In our scenario, the agent takes the offloading decision and the decision of optimal resource allocation. Hence the action space vector for offloading at time t is denoted as $\mathbf{O}_t = \{a_{t1}, a_{t2}, \ldots, a_{tn}\}$, and the corresponding resource allocation vector is $\mathbf{B}_t = \{b_{t1}, b_{t2}, \ldots, b_{tn}\}$. Thus, we represent the action vector as $\mathbf{A}_t = \{\mathbf{O}_t, \mathbf{B}_t\}$.

C. REWARD

Our core objective is to minimize the overall cost as described in the optimization problem by assigning weights to distinct cost components. This optimization introduces our objective function, termed as the reward function. In discrete time slot t, the reward function, represented as R_t, takes the form of the negative average of the cumulative cost, divided by a designated weight factor w_f.

To adapt the reward function for real-time MEC environments, the weight factor w_f is dynamically adjusted based on the real-time system states and task requirements. Specifically, the weights are fine-tuned using state-action advantage values that prioritize latency reduction in delay-sensitive scenarios and energy conservation under constrained resource conditions.

The optimization framework is represented as an MDP, encapsulating the transition probabilities that handle state changes, specifically from state s_t in one-time slot to the next. These transition probabilities, denoted as $P(fs_{t+1}|s_t, p_g)$, encapsulate the likelihood of state transitions, incorporating both the system's dynamics and the actions taken by the IoT devices. The decision-making policy of the agent, p, effectively maps states to actions, guiding the trajectory within the MDP. Aligned with this decision policy p, the cumulative reward function, denoted as $Q(s_t, a_t)$, encapsulates the collective reward associated with a specific state s_t and action a_t. It combines both the immediate reward R_t at time t and the sum of anticipated future rewards. These future rewards are influenced by the subsequent state s_{t+1} and action a_{t+1} in the following time slot $t+1$. Here, the discount factor c determines the importance of future rewards compared to immediate ones. Adjusting the weight of future rewards strikes a balance in the significance of temporal rewards.

Finally, the equation can be expressed as:

$$Q(s_t, a_t) = R_t + c \sum_{s_{t+1} \in S, a_{t+1} \in A} P(f|s_t, p_g) \cdot Q_p(s_{t+1}, a_{t+1}) \tag{14}$$

The bounds for the state-action value function under varying loads can be expressed as:

$$|V^{\pi}(s) - V^{*}(s)| \leq \epsilon(N) \tag{15}$$

where $\epsilon(N)$ quantifies the approximation error scaling with the number of devices. This is because increasing the number of devices may increase the approximation error due to the system's complexity. Under a bandwidth limit B, the following holds for the value function under the policy π:

$$V^{\pi}(s) \geq (1-\alpha)V^{*}(s) \tag{16}$$

where $\alpha = f(B, N)$ represents the penalty due to the bandwidth constraint. Here, $f(B, N)$ contains the relationship between the available bandwidth B and the number of devices N. This influences the achievable state-action value.

Under bandwidth limitations, we extend the penalty α as a function of both bandwidth and system load which is expressed as:

$$\alpha = \frac{B}{N \cdot B_{\max}} \tag{17}$$

Algorithm 1 Proposed Algorithm for Optimal Offloading and Resource Allocation

1: Initialize replay memory buffer for storing transitions related to offloading decisions and rewards (energy, latency).
2: Set parameters of online network with random weights θ.
3: Initialize parameters of target network: $\theta' = \theta$.
4: **for** *episode* $= 1, 2, \ldots, E$ **do**
5: Observe initial state S_0, which includes the vehicle's state (speed, location, task queue, etc.).
6: **for** $t = 1, 2, \ldots, \tau$ **do** ▷ Iterate through time slots for each episode
7: Choose random action A_t with probability ε, else choose action A_t maximizing Q-value, guided by constraints in **P1**.
8: Execute action A_t in the MEC environment, receive reward R_t (based on energy consumption and latency as defined in **P1**, observe next state S_{t+1}.
9: Calculate priority value of transition (S_t, A_t, R_t, S_{t+1}) and store it in replay memory buffer.
10: Sample minibatch of transitions (S_j, A_j, R_j, S_{j+1}) from replay memory buffer based on priority.
11: Compute target Q-value y'_j, i using Bellman equation:

$$y'_j = R_j + \gamma Q'_i(S_{j+1}, \text{argmax}_{a \in \mathcal{A}_i} Q_i(S_{j+1}, a))$$

12: Perform gradient descent step to update online network parameters θ by minimizing loss related to **P1**:

$$\mathcal{L} = \frac{1}{2N} \sum_{d} (y'_j, d - Q_d(S_j, A_j))^2$$

13: Update target network parameters every t_{copy} time steps: $\theta' = \theta$.
14: Update current state S_t to next state S_{t+1}.
15: **end for**
16: **end for**

where $B_{\max}$ is the maximum bandwidth capacity of the system. When the number of devices increases or available bandwidth decreases, the penalty α increases. This results in an increase in the gap between $V^{\pi}(s)$ and $V^{*}(s)$, which quantifies the impact of bandwidth limitations on the value function in constrained environments.

D. DUELING DQN APPROACH FOR Q-VALUE ESTIMATION

In the formulated MDP, we consider LSTM using DQN for mapping between each state-action pair to Q-values as depicted in Figure 2. We use state information as the input to the LSTM network. Despite the potential of DQN in handling Q-table for large state-action spaces, it encounters difficulties when dealing with an exponential increase in possible state-actions due to the increase in dimensions. To overcome this limitation, we consider MBDDQN. This approach addresses the problem of a vast state-action space by breaking the IoT device combinations of actions [35]. Here, each branch is tasked with generating actions in a specific dimension. This strategic adjustment leads to a linear growth in the action output space based on the quantification of IoT devices. The transition from nN to $n \times N$ output spaces addresses scalability issues when dealing with combinations of actions. Moreover, MBDDQN redefines architecture by employing multiple branches, each responsible for actions in distinct dimensions. This innovative approach effectively tackles the challenge of an exponential action space, significantly enhancing the efficiency of the computation offloading process. Here, it is worth noting that the proposed approach introduces additional computational overhead due to the increased complexity of managing multiple branches. The computational cost grows with the number of branches. To overcome this, one can use techniques such as efficient parameter management and reduced precision arithmetic to help mitigate this overhead.

We enhance the learning stability and generalization of the proposed algorithm. As DQN only directly predicts the Q-values of actions, the Dueling architecture enhances DQN by separating the Q-value estimation into two streams. The first stream estimates the value of the current state, while the second estimates the advantage of each action. This separation allows Dueling DQN to learn and prioritize actions more efficiently, particularly in scenarios where the values of different actions vary significantly. In addition, this method improves the model's capacity to evaluate the value of individual actions accurately. By splitting the estimation into a function of state value and an action advantage, these components are combined through an aggregation layer to form the function of action value. Mathematically,

$$Q(s, a) = V(s) + A(s, a) \tag{18}$$

To demonstrate that combining LSTM temporal dependencies with dueling advantage-value estimation preserves the Bellman optimality principle, the Bellman optimality equation for Q-values is needed, which is represented as:

$$Q^{*}(s, a) = \mathbb{E}[r + \gamma \max_{a'} Q^{*}(s', a')] \tag{19}$$

The LSTM network captures temporal dependencies in a sequence of observations, effectively storing past information. In the Bellman optimality equation, both $V(s)$ and $A(s, a)$ depend on the state-action pair and should satisfy the Bellman equation. The Bellman operator for the Q-function remains:

$$T_{\pi} Q(s, a) = \mathbb{E}[r + \gamma \max_{a'} Q(s', a')] \tag{20}$$

By combining LSTM and dueling advantage-value estimation, the LSTM enhances temporal representation, and the dueling architecture separates the value and advantage, both of which still satisfy the Bellman optimality conditions. As $k \to \infty$, the value function Q_k should converge to the optimal Q-function Q^{*}, thus:

$$\forall s \in S, \lim_{k \to \infty} ||Q_k - Q^{*}|| = 0 \tag{21}$$

Moreover, For LSTM hidden states h_t, the gradient norms are bounded as:

$$\lambda_{\min} \leq ||\nabla h_t|| \leq \lambda_{\max} \tag{22}$$

This ensures that the gradients are neither vanishing nor exploding, which is crucial for stable training with LSTM. The proposed MBDDQN network structure calculates the action advantages within the context of IoT device actions. This procedure treats each IoT device action as an individual action dimension. Multiple branches of the network calculate advantage values for every action dimension. This allows separate optimization for different action dimensions while enabling coordination among sub-actions through a shared decision function. To prove that separating the state-value and advantage streams maintains this relationship, we need to show the Bellman equation holds. The Q-function with multi-branch error $|Q_{MB}(s, a) - Q^{*}(s, a)|$ can be bounded as:

$$|Q_{MB}(s, a) - Q^{*}(s, a)| \leq \frac{\epsilon}{1 - \gamma} \tag{23}$$

This follows from the contraction property of the Bellman operator under the multi-branch structure. The Bellman operator $T_{\pi} Q(s, a) = \mathbb{E}[r + \gamma Q(s', \pi(s'))]$ is a contraction mapping under standard conditions. Since T_{π} is still a contraction under branch separation, this implies convergence to the optimal Q-function.

The combined outcome of action advantage values and state values generates the action-value function specific to each action dimension. This scheme effectively selects sub-actions within each dimension by computing their Q-values. Finally, these Q-values are determined by combining state values and advantage values using the equation:

$$Q_d(S, a_i) = V(S) + \left(A_d(S, a_i) - \frac{1}{n} \sum_{a' \in \mathcal{D}} A_d(S, a') \right) \tag{24}$$

where Q_d represents the Q-value of sub-action denoted as a_i in action dimension represented as d, $V(S)$ is the state value, $A_d(S, a_i)$ is the advantage value for sub-action a_i in dimension d, and $\mathcal{D}$ denotes the set of sub-actions in dimension d. The joint action encompasses the decision of partial offloading and resource allocation. Here, the quantification of action dimensions is $2N$. So, we represent the action dimension as d be within the range $1, 2, \ldots, 2N$, where each dimension encompasses a set of sub-actions with cardinality $|\mathcal{A}_d| = n$. The selected representative corresponds to the sub-action with the highest action value function in each action dimension. The resulting action is determined by integrating the actions across all dimensions. Moreover, our action includes both the offloading decision and resource allocation hence can be represented as:

$$A_i = \arg\max_{a'_d \in \mathcal{A}_i} Q_i(S, a'_d) \tag{25}$$

$$A = [A_{x1}, A_{x2}, \ldots, A_{xN}, A_{y1}, A_{y2}, \ldots, A_{yN}] \tag{26}$$

where $\mathbf{A}_{xi}$ signifies the offloading decision for IoT device i, and $\mathbf{A}_{yi}$ indicates the resource allocation of IoT device i. As depicted in Fig 2, the integration involves separating the processes of choosing actions and assessing them by using networks with separate parameters while still sharing a consistent structure. The online network estimates Q-values in the current state, a crucial component for action selection and policy updates. Contrarily, the target network estimates Q-values in the subsequent state to prevent neural network instability during training. The target Q-value for each branch, denoted as y'_i, is computed using a formula that factors in immediate rewards, future Q-values, and the optimal action selection in the next state, and presented as

$$y'_i = R_t + \gamma Q'_i(S_{t+1}, \arg\max_{a'_d \in \mathcal{D}_i} Q_i(s_{t+1}, a'_d)) \tag{27}$$

Throughout the training phase, the MBDDQN agent selects a subset of training samples from the replay memory, encompassing state-action pairs along with their corresponding rewards. These samples facilitate the iterative adjustment of the online network's parameters. The update process involves minimizing a loss function, represented as

$$\mathcal{L}_{oss} = \mathbb{E}\left[\frac{1}{2N}\sum_{d}(y'_d - Q_d(S_j, A_j, d))^2\right] \tag{28}$$

In equations 16 and 17, $Q_i(S_j, A_j, d)$ represents Q-value function for action dimension d estimated by the online network in state S_j with action A_j. $Q'_i(S_{t+1})$ represents Q-value function for the next state S_{t+1} estimated by the target network. y'_i is he target Q-value for branch i calculated using immediate reward (R_t), discount factor (γ), and optimal action selection in the next state. R_t is the immediate reward received after taking action A_j in state S_j at time t. γ represents the discount factor used to consider future rewards in the target Q-value computation. $\arg\max_{a'_d \in \mathcal{D}_i}$ is the operation to find the action a'_d that maximizes the Q-value for dimension d among all possible sub-actions. S_{t+1} represents the subsequent state after taking action A_j in state S_j at time t. $\mathcal{L}_{oss}$ is the loss function quantifying the difference between target Q-values (y'_d) and the Q-values predicted by the online network ($Q_d(S_j, A_j, d)$). y'_d denotes the target Q-value for branch d in the context of the loss function. $Q_d(S_j, A_j, d)$ represents the Q-value estimated by the online network for action dimension d in state S_j with action A_j. N denotes the number of action dimensions, and $\mathbb{E}$ represents the expectation operator, representing the average over the training dataset.

Different from the prior method of experience replay, where samples are drawn uniformly from a buffer, we employ a prioritized sampling approach. The objective is to enhance the efficiency of sample selection by favoring tuples that contribute more informative data. Tuples that exhibit notable disparities between the rewards predicted by the model and the actual target rewards are granted a higher probability of being selected for learning. In practical terms, the probability of selecting the i-th tuple from the experience buffer is determined using a formula that incorporates parameters such as α_0, which controls the priority assigned to each tuple. A small positive value ζ is introduced to ensure the feasibility of probability calculations. Similarly, β_0 influences the likelihood of a tuple being chosen more frequently when it is assigned a higher value.

We represent $\delta(i)$ as the cumulative difference between the expected reward and the predicted Q-value in each branch of the network. This measure calculates how much the expected reward differs from the model's prediction. To address the potential of overfitting due to the influence of sampling probabilities, we employ a strategy to balance the impact of various losses computed using Equation (17). This strategy involves assigning distinct weights to these losses. The weight $w(i)$ for each tuple is computed based on the sampling probability $p(i)$, and a parameter β_0. Hence, the formulated equation can be expressed as

$$p(i) = \frac{|\delta(i)| + \zeta}{\sum_j (|\delta(j)| + \zeta)^{\alpha_0}} \tag{29}$$

where ζ ensures the feasibility of probability calculations, α_0 influences tuple priority, and $\delta(i)$ represents the cumulative difference between the expected reward y and the predicted Q-function value for each network branch.

The cumulative difference between the expected reward y_i and the predicted Q-values from various branches of the network is computed using the equation:

$$\delta(i) = \sum_{k \in K} |y_i - Q_k(s, a_k)| \tag{30}$$

This calculation quantifies the overall divergence between the anticipated reward and the Q-values predicted by different branches. To address the potential risk of overfitting in Q-function computation due to the bias introduced by the sampling probability, a strategy is adopted to adjust the influence of different losses. The aim is to reduce the magnitude of gradient updates for tuples that are sampled more frequently.

This is achieved by varying the weights assigned to the losses. Each weight $w(i)$ is defined as:

$$w(i) = (B \cdot p(i))^{-\Gamma_0} \tag{31}$$

where Γ_0 controls the importance of the weight adjustments. The purpose of this approach is to achieve a balanced learning process by properly balancing the significance of loss contributions from different tuples. This helps prevent an excessive bias towards frequently sampled tuples, contributing to a more stable and effective learning process, particularly when prioritized sampling is utilized.

E. ADAPTIVE COST-WEIGHTING MECHANISM

We present a novel adaptive cost-weighting mechanism to dynamically adjust the trade-offs between energy consumption, latency, and bandwidth utilization based on real-time system conditions. Our adaptive approach allows the system to prioritize different objectives depending on the state of the network and the tasks being processed. The weights in this formulation are extended from the previous formulation by using an adaptive weighting mechanism to optimize system performance dynamically. This adaptive weighting extends the initial problem formulation by enabling real-time optimization.

Suppose N represents the total number of IoT devices in the system. Then the total system cost C can be expressed as a weighted sum of these components for all IoT devices, where w_E, w_L, and w_B are the weights for energy consumption, latency, and bandwidth, respectively:

$$C = \sum_{i=1}^{N} (w_E E_i + w_L L_i + w_B B_i) \tag{32}$$

F. DYNAMIC WEIGHTS FORMULATION

To ensure the adaptability of the proposed scheme, the weights w_E, w_L, and w_B are designed to be dynamic and according to the real-time state of the system. These weights are functions of the current system state, denoted as s_t at time t, which includes the average system latency denoted as $\bar{L}_t$, the average energy consumption of IoT devices denoted as $\bar{E}_t$: T and the average bandwidth utilization $\bar{B}_t$. Hence, the dynamic weights can be expressed as:

$$w_E(t) = \frac{\lambda_E}{\lambda_E + \bar{E}_t}, \quad w_L(t) = \frac{\lambda_L}{\lambda_L + \bar{L}_t}, \quad w_B(t) = \frac{\lambda_B}{\lambda_B + \bar{B}_t} \tag{33}$$

where λ_E, λ_L, and λ_B are tuning parameters that control the sensitivity of the weights to the corresponding metrics. Higher values of λ reduce the sensitivity and lower values make the system more responsive to changes in energy, latency, or bandwidth. With the dynamic weights, the total system cost at time t is:

$$C(t) = \sum_{i=1}^{N} (w_E(t) E_i + w_L(t) L_i + w_B(t) B_i) \tag{34}$$

The time complexity of the proposed algorithm can be expressed as:

$$T(n) = O(BS \cdot |S| \cdot \max(|A_i|))$$

where BS is the batch size, $|S|$ is the state space, and $|A_i|$ is the action space for branch i, and the space complexity is:

$$\text{Memory} = O(N \cdot |S| \cdot \sum |A_i|)$$

where N is the number of IoT devices, and the sum represents the total action space across all branches. By splitting the action space into separate branches, the total action space complexity is reduced from exponential $|A|$ to a linear sum $\sum |A_i|$, where A_i represents the action space for each branch.

VI. SIMULATION RESULTS

A. SIMULATION SETUP

To implement the MBDDQN, we use Ubuntu on an AMD Ryzen CPU with 32 GB RAM, equipped with a GPU, and utilize the TensorFlow framework, a widely used open-source software library for machine learning. We set the number of IoT devices for the simulation environment to $N = 10 - 20$, with each user assigned $M = 4$ computational tasks. We define the local computation time for each IoT device as 4.75×10^{-7} s/bit and the energy consumption for local processing as 3.25×10^{-7} J/bit. For the simulation of varying task demands, we set the input data sizes of all tasks to be uniformly distributed between 10 MB and 30 MB. The output data sizes are randomly distributed between 1 MB and 3 MB. The uplink bandwidth constraint is set to 150 Mbps. Additionally, the energy efficiency of IoT devices for data transmission and reception is set to 1.42×10^{-7} J/bit.The remote edge server's CPU processing rate is configured to 10×10^9 cycles/s, which ensures efficient task processing in offloading scenarios. We further set the task offloading energy coefficient $\lambda = 2.5 \times 10^{-7}$ J/bit, and the weight for task energy consumption $\omega_n = 1$ J/s. Table 3 shows a summary of the detailed parameter settings [36], [37], [38], [39], [40], [41], [42], [43], [44], [45], [46], [47]. To ensure optimal learning efficiency and performance in task offloading, the learning rate is initially set to $\alpha = 0.001$, which is optimized through grid search within the range [0.0001, 0.01]. The batch size is explored within the range [16, 32, 64] and 64 is chosen. The network architecture for the MBDDQN consists of six layers. There is one input layer, one LSTM layer with 64 units to capture temporal dependencies which were first tested between 32 and 128, followed by two fully connected dense layers with 128 units each and ReLU activation functions, and one advantage-value layer. The output layer with Sigmoid includes a binary output for the offloading decision and a continuous output for resource allocation, each with distinct dimensions. The buffer size is set to 2000 to balance training efficiency and memory usage. Finally, ϵ is initialized at 1.0 and decays to 0.01 with a decay rate of $\epsilon_{decay} = 0.995$. For validation, we conduct separate simulations where performance metrics,

TABLE 3. Simulation parameters.

Parameter	Value
Number of IoT devices	10-20
Local computation time	4.75×10^{-7} s/bit
Processing energy consumption	3.25×10^{-7} J/bit
Task input data size	10 MB to 30 MB
Task output data size	1 MB to 3 MB
Uplink bandwidth limit	150 Mbps
IoT device energy efficiency	1.42×10^{-7} J/bit
Edge server CPU rate	10×10^{9} cycles/s
Energy coefficient (λ)	2.5×10^{-7} J/bit
Energy weight (ω_n)	1 J/s

including energy and delay, are evaluated at each step. This validation ensures that the selected hyperparameter results in optimal performance. Moreover, explore different ranges of parameters such as learning rate from 0.0001 to 0.1, batch size 32 to 256, discount factor 0.9 to 0.99, and exploration rate 0.1 to 1.0.

The MBDDQN is compared with pure local computation, pure edge computation, greedy binary offloading, and random offloading. Each of these is described below:

- Pure Local: In this scenario, all the tasks are processed entirely at the IoT device without any partial offloading decisions.
- Pure Edge: In this case, all tasks are executed at the edge server, but the offloading decisions are made without utilizing any insights or optimization strategies.
- Greedy Binary: In this case, a greedy decision is made to make binary offloading decisions. Here, binary offloading means that the task will be executed either locally or completely at the edge server without partitioning. In each time frame t, if the total reward achieved during local execution surpasses that of offload mode for the workloads from each IoT device, then local execution is performed for the respective IoT device in the next time frame $t + 1$. Otherwise, the offload mode is chosen.
- Random Offloading: This approach involves randomly offloading some tasks to the edge server, without employing any informed decision-making process.
- DQN: This approach involves DQN for optimal computation offloading and resource allocation decisions [36].
- DDPG: This approach involves Deep Deterministic Policy Gradient (DDPG) for continuous and optimal computation offloading and resource allocation decisions [37].

B. COMPARISON OF THE AVERAGE DELAY

First, we compare the results based on local execution delay. After the evaluation, we conclude that pure local computation shows the weakest performance as shown in Figure 3. This is because the processing power of IoT devices is not as enough for certain tasks as the more powerful edge servers. Because of this difference, most of the functions cannot be efficiently carried out on the IoT devices themselves. The pure edge computation achieves

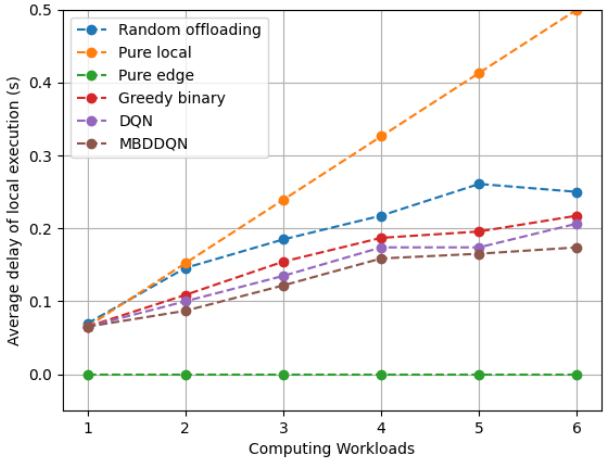

FIGURE 3. Comparison of the average delay in local execution.

a delay of zero for local execution. This is because of offloading all tasks to the edge servers rather than executing them locally. The MBDDQN strategy efficiently minimizes local execution by utilizing effective learning techniques for partial computation workloads. By offloading a portion of the workload from the IoT device, it reduces the delay in local execution. It surpasses DQN due to its ability to better distinguish between state values and action advantages. On the other hand, the greedy binary approach surpasses random offloading and achieves a similar outcome to MBDDQN by also distributing the workload away from the local IoT devices, based on predefined thresholds. Here, the x-axis represents the size of the individual tasks in MBs that need to be executed or offloaded. The tasks of IoT devices may vary in data size depending on the specific requirements of each IoT device but the number of IoT devices remains fixed. Figure 4 compares delay in offloading execution. This comparison is quite similar to the results obtained from local execution. Notably, local execution results in no delay since all tasks are offloaded to the edge server. In contrast, pure edge computation yields the highest delay due to the additional time needed for transferring tasks, waiting in the queue, and eventual execution. The greedy binary approach and DQN outperform pure edge and random offloading methods by utilizing predefined thresholds for decision-making, effectively distributing the workload between the edge server and IoT devices. Lastly, the MBDDQN strategy outshines all other approaches by incorporating intelligent partial decisions, and strategically lightening the load through necessary actions.

C. COMPARISON OF BANDWIDTH ALLOCATION

We examine our proposed framework's effectiveness using action parameters within our action space A, which includes the partial offloading decision and resource allocation. In Figure 5, we evaluate performance based on the bandwidth allocation ratio within the MBDDQN framework, comparing

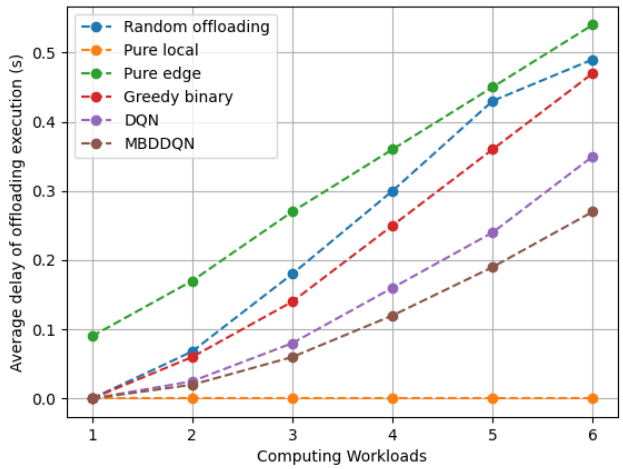

FIGURE 4. Comparison of the average delay in offloading execution.

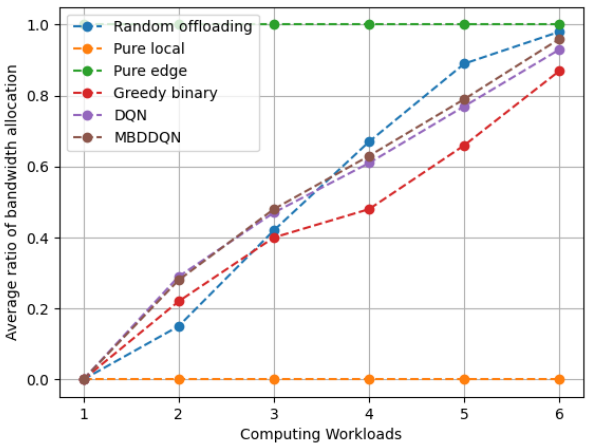

FIGURE 5. Comparison of the ratio of bandwidth allocation.

it with DQN, pure local computation, pure edge computation, greedy binary offloading, and random offloading. It is illustrated that the bandwidth allocation ratio for pure local execution is zero. This is due to all tasks being constrained to local IoT devices without any offloading possibility. Conversely, the pure edge computation consistently achieves a maximum bandwidth allocation ratio of one, given that all tasks are offloaded to edge servers. As workload increases, all algorithms tend to offload tasks to edge servers since the IoT devices lack sufficient resources to handle incoming tasks, hence consuming more ratio of bandwidth. MBDDQN surpasses DQN leading to more efficient decision-making, optimizing resource allocation, and enhancing bandwidth utilization.

D. COMPARISON OF THE ENERGY CONSUMPTION

Figure 6 compares the five algorithms based on energy consumption. Here, we can see that the pure edge computation achieves lower energy consumption compared with other schemes. Pure local computation consumes more energy as the tasks are executed locally at the IoT device. At a certain location, it can be constant because facing the penalties for the failure to complete the given tasks. Random and greedy binary offloading consumes more energy as random offloading migrates some tasks to the edge server in a random fashion, while greedy offloading executes certain tasks at the local IoT device based on cumulative rewards. Hence, considering these factors, the MBDDQN performs better than these algorithms. This is due to two reasons: First, the workload is partially divided between the end IoT device and edge servers. Secondly, intelligent decisions for optimal bandwidth allocations are based on the multi-branch factor derived from previous results. MBDDQN also exceeds DQN and DDPG to optimize energy consumption due to more precise offloading decisions, minimizing unnecessary computations, and reducing transmission power through better state-action pairs.

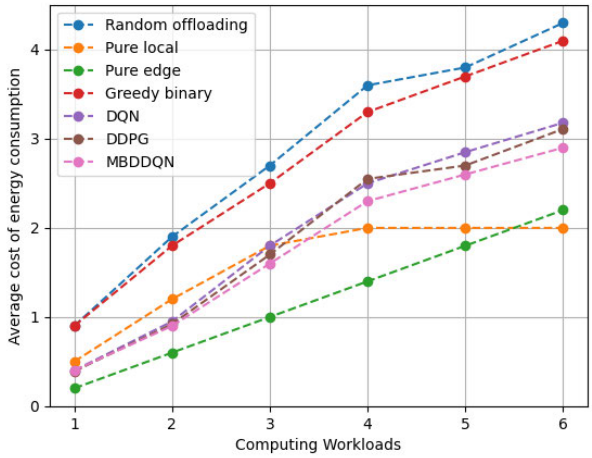

FIGURE 6. Comparison of the average energy consumption cost.

E. IMPACT OF INCREASE IN IoT DEVICES

In Figure 7, we present a comparison between the proposed framework, pure local computation, pure edge computation, and greedy and random offloading strategies. The results show that, apart from local computation, all other algorithms exhibit an increase in delay. This is attributed to local computation efficiently executing tasks within the IoT devices, and all IoT devices operate independently, mitigating delay. In contrast, other algorithms face significant delays. The random offloading approach outperforms pure local and pure edge computation, especially when the number of IoT devices exceeds 15. This advantage comes from the randomized utilization of IoT devices and edge server resources, which improves efficiency. In contrast, pure edge computation demonstrates the highest delay due to the necessity of task transmission, resulting in accrued transfer, queuing, and execution times. However, the proposed MBDDQN algorithm outperforms these other methods by utilizing a strategic learning process for partial offloading. This strategy efficiently combines the strengths of IoT devices and edge servers, resulting in improved results. Although there is still

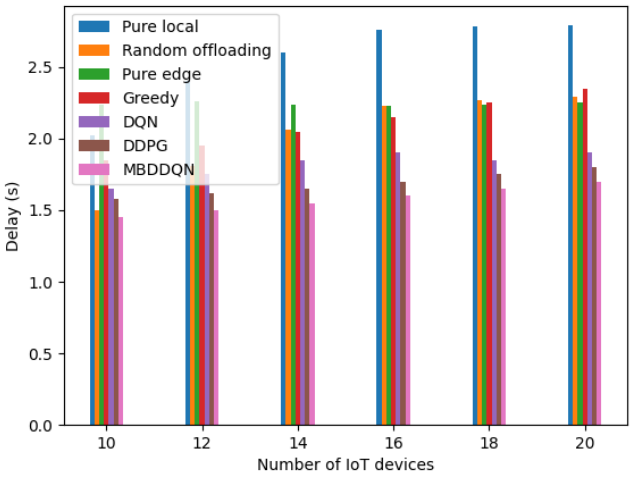

FIGURE 7. Comparison of the average delay as the number of IoT devices increases.

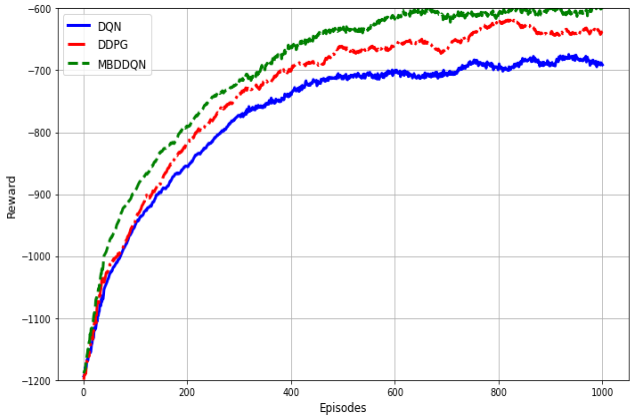

FIGURE 8. Comparison of reward with DRL based schemes.

a slight delay increase, it becomes much less noticeable and grows slowly as the number of IoT devices increases. Compared to DQN and DDPG, as the number of IoT devices increases in the network, MBDDQN better manages the average delay. This is due to its ability to evaluate state values and action advantages. Finally, we compare the reward of MBDDQN with DDPG and DQN in Figure 8, which shows the superiority of the convergence of the proposed algorithm. Although statistical significance tests and confidence intervals are valuable for some comparisons, they are not included in this work because we focus on the practical performance metrics related to task offloading and resource allocation in MEC.

F. ABLATION ANALYSIS

The ablation analysis presented in Figure 9 reveals that our model self-tunes to avoid excessive energy usage in resource-constrained environments. This is because the adaptive cost weighting dynamically adjusts the importance of energy conservation in the reward function. When the resources are critical such as in energy-critical situations,

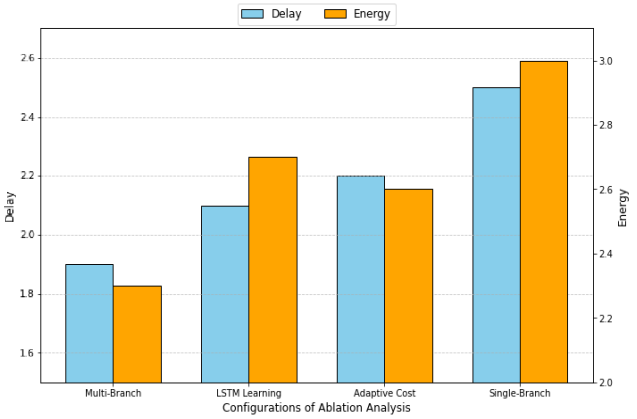

FIGURE 9. Ablation analysis on proposed approach.

it prioritizes minimal energy-offloading. The adaptive cost weighting mechanism also prioritizes the delay across other competing objectives. In critical situations, it prioritizes low latency actions such as local execution or offloading when the channel quality is optimal. Compared to this adaptive approach, fixed reward structures often fail to adapt to changing system requirements, however, the adaptive cost weighting mechanism enables the model to focus more on critical performance metrics at a given time slot. One can also see that multi-branch architecture reduces the delay compared with single-branch architecture. This is because multi-branch architecture independently optimizes the computation, offering decisions and resource allocation. This separation minimizes the combinatorial complexity of the algorithm and reduces the time for decision-making. Similarly, by distributing the computational load across multiple branches, it efficiently reduces energy consumption. We also use LSTM for temporal learning, which enables the model to predict future costs more effectively. This can result in better preemptive decisions for energy-efficient offloading. It can also provide optimal offloading decision time, such as when channel conditions are preferable. However, due to its computational overhead, the delay slightly increases. It is important to note that there are some internal and external validity threats. For example, one internal validity threat is model overfitting to the training data. Similarly, the external threat could be a limitation of the proposed model to capture the complexity and variability of a real-world MEC network. Another could be the scalability of the system in highly heterogeneous environments.

VII. CONCLUSION

Optimal computation offloading requires detailed consideration of parameters including dynamic channel gains, the number of IoT devices, and task characteristics including heterogeneous processing demands and data size, IoT devices load, transmission power, and edge server load while offloading decisions and resource allocation. Previous studies utilize Q-learning, and Deep Q Network (DQN) that limit offloading decisions while in the presence of diverse states, the number

of IoT devices growth, and large action space. To overcome this challenge, our key contributions are threefold: A multi-branch network with Dueling DQN (MBDDQN) to make partial offloading decisions and resource allocation with distinct network branches for each. The objective was to minimize the overall system cost, including energy consumption and latency. Second, to enhance both short-term action selection and long-term system cost estimation, a long short-term memory (LSTM) network is used with distinct advantage-value layers to enhance the temporal learning capacity. Finally, a novel adaptive cost-weighting mechanism is employed in the reward function to dynamically balance the energy consumption, latency, and bandwidth utilization. Simulation results showed the superiority of the proposed work over the benchmarks.

From a practical perspective, the proposed approach provides an adaptive and scalable offloading solution which makes it suitable for real-time mobile edge applications such as in smart cities, healthcare, and manufacturing. To support this applicability, the simulation results present the effectiveness of the proposed work compared to previous benchmarks in terms of system performance of resource utilization.

Our proposed system also has some limitations. For example, the simulations were performed in a controlled environment which may suffer to capture the complexities of real-world scenarios. Moreover, the proposed system does not cover the mobility-related challenges which h may also cause disruptions in the network environment. Finally, we consider delay and energy consumption while ignoring the security and privacy concerns of IoT devices which is a significant challenge in MEC.

In the future, we aim to deploy the proposed algorithm with more realistic and dynamic scenarios such as vehicular and mobile networks which may also cause frequent disruptions in the network degrading the offloading decisions. We also aim to enhance this work by utilizing Non-Orthogonal Multiple Access for radio access resources and the usage of the same channel by multiple users with Orthogonal Frequency-Division Multiple Access (OFDMA) to further increase the capacity and reduce latency. Finally, we also aim to explore DRL-based approaches for security-aware computation offloading to ensure data protection in MEC networks.

ACKNOWLEDGMENT

The Researchers would like to thank the Deanship of Graduate Studies and Scientific Research at Qassim University for financial support (QU-APC-2025).

REFERENCES

[1] X. Zhang and S. Debroy, "Resource management in mobile edge computing: A comprehensive survey," *ACM Comput. Surv.*, vol. 55, no. 13s, pp. 1–37, Dec. 2023.

[2] X. Wang, J. Li, Z. Ning, Q. Song, L. Guo, S. Guo, and M. S. Obaidat, "Wireless powered mobile edge computing networks: A survey," *ACM Comput. Surv.*, vol. 55, no. 13s, pp. 1–37, Dec. 2023.

[3] Á. Santos, J. Bernardino, and N. Correia, "Automated application deployment on multi-access edge computing: A survey," *IEEE Access*, vol. 11, pp. 89393–89408, 2023.

[4] E. Mustafa, J. Shuja, S. K. U. Zaman, A. I. Jehangiri, S. Din, F. Rehman, S. Mustafa, T. Maqsood, and A. N. Khan, "Joint wireless power transfer and task offloading in mobile edge computing: A survey," *Cluster Comput.*, vol. 25, no. 4, pp. 2429–2448, Aug. 2022.

[5] A. Ali, N. Azim, M. T. B. Othman, A. U. Rehman, M. Alajmi, M. H. Al-Adhaileh, F. U. Khan, M. Orken, and H. Hamam, "Joint optimization of computation offloading and task scheduling using multi-objective arithmetic optimization algorithm in cloud-fog computing," *IEEE Access*, vol. 12, pp. 184158–184178, 2024.

[6] H. A. Alharbi, M. Aldossary, J. Almutairi, and I. A. Elgendy, "Energy-aware and secure task offloading for multi-tier edge-cloud computing systems," *Sensors*, vol. 23, no. 6, p. 3254, Mar. 2023.

[7] M. Poposka and Z. Hadzi-Velkov, "Binary vs partial offloading in wireless powered mobile edge computing systems with fairness guarantees," *ITU J. Future Evolving Technol.*, vol. 3, no. 2, pp. 498–507, Sep. 2022.

[8] S. Song, S. Ma, X. Zhu, Y. Li, F. Yang, and L. Zhai, "Joint bandwidth allocation and task offloading in multi-access edge computing," *Expert Syst. Appl.*, vol. 217, May 2023, Art. no. 119563.

[9] Z. He, Y. Xu, M. Zhao, W. Zhou, and K. Li, "Priority-based offloading optimization in cloud-edge collaborative computing," *IEEE Trans. Services Comput.*, vol. 16, no. 6, pp. 3906–3919, Nov. 2023.

[10] L. Liu, B. Sun, Y. Wu, and D. H. K. Tsang, "Latency optimization for computation offloading with hybrid NOMA–OMA transmission," *IEEE Internet Things J.*, vol. 8, no. 8, pp. 6677–6691, Apr. 2021.

[11] S. Taheri-Abed, A. M. Eftekhari Moghadam, and M. H. Rezvani, "Machine learning-based computation offloading in edge and fog: A systematic review," *Cluster Comput.*, vol. 26, no. 5, pp. 3113–3144, Oct. 2023.

[12] B. Kar, W. Yahya, Y.-D. Lin, and A. Ali, "Offloading using traditional optimization and machine learning in federated cloud-edge-fog systems: A survey," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 2, pp. 1199–1226, 2nd Quart., 2023.

[13] Z. Zabihi, A. M. Eftekhari Moghadam, and M. H. Rezvani, "Reinforcement learning methods for computation offloading: A systematic review," *ACM Comput. Surv.*, vol. 56, no. 1, pp. 1–41, Jan. 2024.

[14] E. Mustafa, J. Shuja, F. Rehman, A. Riaz, M. Maray, M. Bilal, and M. K. Khan, "Deep neural networks meet computation offloading in mobile edge networks: Applications, taxonomy, and open issues," *J. Netw. Comput. Appl.*, vol. 226, Jun. 2024, Art. no. 103886.

[15] K. Zheng, G. Jiang, X. Liu, K. Chi, X. Yao, and J. Liu, "DRL-based offloading for computation delay minimization in wireless-powered multi-access edge computing," *IEEE Trans. Commun.*, vol. 71, no. 3, pp. 1755–1770, Mar. 2023.

[16] Z. Wei, R. He, Y. Li, and C. Song, "DRL-based computation offloading and resource allocation in green MEC-enabled maritime-IoT networks," *Electronics*, vol. 12, no. 24, p. 4967, Dec. 2023.

[17] C. Fang, Z. Hu, X. Meng, S. Tu, Z. Wang, D. Zeng, W. Ni, S. Guo, and Z. Han, "DRL-driven joint task offloading and resource allocation for energy-efficient content delivery in cloud-edge cooperation networks," *IEEE Trans. Veh. Technol.*, vol. 72, no. 12, pp. 16195–16207, Dec. 2023.

[18] Z. Aghapour, S. Sharifian, and H. Taheri, "Task offloading and resource allocation algorithm based on deep reinforcement learning for distributed AI execution tasks in IoT edge computing environments," *Comput. Netw.*, vol. 223, Mar. 2023, Art. no. 109577.

[19] M. Maray, E. Mustafa, and J. Shuja, "Wireless power assisted computation offloading in mobile edge computing: A deep reinforcement learning approach," *Hum.-Centric Comput. Inf. Sci.*, vol. 14, pp. 1–20, Apr. 2024.

[20] A. Shakarami, M. Ghobaei-Arani, M. Masdari, and M. Hosseinzadeh, "A survey on the computation offloading approaches in mobile edge/cloud computing environment: A stochastic-based perspective," *J. Grid Comput.*, vol. 18, no. 4, pp. 639–671, Dec. 2020.

[21] A. Shakarami, A. Shahidinejad, and M. Ghobaei-Arani, "A review on the computation offloading approaches in mobile edge computing: A game-theoreticperspective," *Softw., Pract. Exper.*, vol. 50, no. 9, pp. 1719–1759, Sep. 2020.

[22] A. Shakarami, M. Ghobaei-Arani, and A. Shahidinejad, "A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective," *Comput. Netw.*, vol. 182, Dec. 2020, Art. no. 107496.

[23] S. Bi and Y. J. Zhang, "Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading," *IEEE Trans. Wireless Commun.*, vol. 17, no. 6, pp. 4177–4190, Jun. 2018.
[24] G. Wu, H. Wang, H. Zhang, Y. Zhao, S. Yu, and S. Shen, "Computation offloading method using stochastic games for software defined network-based multi-agent mobile edge computing," *IEEE Internet Things J.*, vol. 10, no. 20, pp. 17620–17634, Oct. 2023.
[25] J. Gao, R. Wu, and J. Hao, "Lyapunov-guided energy scheduling and computation offloading for solar-powered WSN," *Appl. Sci.*, vol. 13, no. 8, p. 4966, Apr. 2023.
[26] Z. Wang, T. Wu, Z. Zhang, and H. Zhou, "A game theory-based computation offloading method in cloud-edge computing networks," in *Proc. Int. Conf. Comput. Commun. Netw. (ICCCN)*, Jul. 2021, pp. 1–6.
[27] Y. Chen, J. Hu, J. Zhao, and G. Min, "QoS-aware computation offloading in LEO satellite edge computing for IoT: A game-theoretical approach," *Chin. J. Electron.*, vol. 33, no. 4, pp. 875–885, Jul. 2024.
[28] K. Zhang, J. Yang, and Z. Lin, "Computation offloading and resource allocation based on game theory in symmetric MEC-enabled vehicular networks," *Symmetry*, vol. 15, no. 6, p. 1241, Jun. 2023.
[29] H. Lin, S. Zeadally, Z. Chen, H. Labiod, and L. Wang, "A survey on computation offloading modeling for edge computing," *J. Netw. Comput. Appl.*, vol. 169, Nov. 2020, Art. no. 102781.
[30] E. F. Maleki, L. Mashayekhy, and S. M. Nabavinejad, "Mobility-aware computation offloading in edge computing using machine learning," *IEEE Trans. Mobile Comput.*, vol. 22, no. 1, pp. 328–340, Jan. 2023.
[31] I. Abdullaev, N. Prodanova, K. Aruna Bhaskar, E. Laxmi Lydia, S. Kadry, and J. Kim, "Task offloading and resource allocation in IoT based mobile edge computing using deep learning," *Comput., Mater. Continua*, vol. 76, no. 2, pp. 1463–1477, 2023.
[32] D. Hortelano, I. de Miguel, R. J. D. Barroso, J. C. Aguado, N. Merayo, L. Ruiz, A. Asensio, X. Masip-Bruin, P. Fernández, R. M. Lorenzo, and E. J. Abril, "A comprehensive survey on reinforcement-learning-based computation offloading techniques in edge computing systems," *J. Netw. Comput. Appl.*, vol. 216, Jul. 2023, Art. no. 103669.
[33] A. Sadiki, J. Bentahar, R. Dssouli, A. En-Nouaary, and H. Otrok, "Deep reinforcement learning for the computation offloading in MIMO-based edge computing," *Ad Hoc Netw.*, vol. 141, Mar. 2023, Art. no. 103080.
[34] 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 Comput.*, vol. 26, no. 2, pp. 1053–1062, Apr. 2023.
[35] Y. Sun and Q. He, "Joint task offloading and resource allocation for multi-user and multi-server MEC networks: A deep reinforcement learning approach with multi-branch architecture," *Eng. Appl. Artif. Intell.*, vol. 126, Nov. 2023, Art. no. 106790.
[36] L. Huang, X. Feng, C. Zhang, L. Qian, and Y. Wu, "Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing," *Digit. Commun. Netw.*, vol. 5, no. 1, pp. 10–17, Feb. 2019.
[37] J. Ren and S. Xu, "DDPG based computation offloading and resource allocation for MEC systems with energy harvesting," in *Proc. IEEE 93rd Veh. Technol. Conf. (VTC-Spring)*, Apr. 2021, pp. 1–5.
[38] Y. Liu, X. Li, J. Yang, H. Liu, P. Hu, and Y. Chen, "Joint computation offloading and trajectory planning for multi-UAV cooperative target search," in *Proc. IEEE 35th Int. Symp. Pers., Indoor Mobile Radio Commun. (PIMRC)*, Sep. 2024, pp. 1–6.
[39] A. Giannopoulos, I. Paralikas, S. Spantideas, and P. Trakadas, "COOLER: Cooperative computation offloading in edge-cloud continuum under latency constraints via multi-agent deep reinforcement learning," in *Proc. Int. Conf. Intell. Comput., Commun., Netw. Services (ICCNS)*, Sep. 2024, pp. 9–16.
[40] P. Choppara and S. S. Mangalampalli, "A hybrid task scheduling technique in fog computing using fuzzy logic and deep reinforcement learning," *IEEE Access*, vol. 12, pp. 176363–176388, 2024.
[41] E. Mustafa, J. Shuja, F. Rehman, A. Namoun, M. Bilal, and A. Iqbal, "Computation offloading in vehicular communications using PPO-based deep reinforcement learning," *J. Supercomput.*, vol. 81, no. 4, pp. 1–24, Feb. 2025.
[42] H. Gao, X. Wang, W. Wei, A. Al-Dulaimi, and Y. Xu, "Com-DDPG: Task offloading based on multiagent reinforcement learning for information-communication-enhanced mobile edge computing in the Internet of Vehicles," *IEEE Trans. Veh. Technol.*, vol. 73, no. 1, pp. 348–361, Jan. 2024.
[43] R. Zakerian and H. Gholami, "SARS: A resource selection algorithm for autonomous driving tasks in heterogeneous mobile edge computing," 2024, *arXiv:2411.15989*.
[44] A. Ali, I. Ullah, S. K. Singh, A. Sharafian, W. Jiang, H. I. Sherazi, and X. Bai, "Energy-efficient resource allocation for urban traffic flow prediction in edge-cloud computing," *Int. J. Intell. Syst.*, vol. 2025, no. 1, Jan. 2025, Art. no. 1863025.
[45] A. Ali, I. Ullah, S. Ahmad, Z. Wu, J. Li, and X. Bai, "An attention-driven spatio-temporal deep hybrid neural networks for traffic flow prediction in transportation systems," *IEEE Trans. Intell. Transp. Syst.*, early access, Feb. 26, 2025, doi: 10.1109/TITS.2025.3540852.
[46] I. Ullah, D. Adhikari, F. Ali, A. Ali, H. Khan, A. Sharafian, S. M. Kesavan, and X. Bai, "Revolutionizing e-commerce with consumer-driven energy-efficient WSNs: A multi-characteristics approach," *IEEE Trans. Consum. Electron.*, vol. 70, no. 4, pp. 6871–6882, Nov. 2024.
[47] Y. Liu, H. Yu, S. Xie, and Y. Zhang, "Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 11, pp. 11158–11168, Nov. 2019.

EHZAZ MUSTAFA received the B.S. and M.S. degrees (Hons.) from COMSATS University Islamabad, Abbottabad Campus, Pakistan, where he is currently pursuing the Ph.D. degree. He is also a Lecturer with COMSATS University Islamabad. His research interests include deep learning in 6G wireless networks and vehicular networks. He received the Gold Medal for his B.S. degree.

JUNAID SHUJA (Member, IEEE) received the Ph.D. degree from the University of Malaya, Malaysia, in 2017, under the prestigious BrightSpark Scholarship. He is currently a Visiting Associate Professor with Southeast Missouri State University. He has previously held academic and research positions at several esteemed institutions, including Universiti Teknologi PETRONAS, Malaysia; the National University of Computer and Emerging Sciences (FAST-NUCES), and COMSATS University, Pakistan. His Ph.D. research focused on the execution and offloading of single instruction and multiple data (SIMD) instructions across heterogeneous mobile and cloud platforms, for which he was honored with the "Graduate on Time" award for his timely completion. He has published more than 75 research papers in leading international journals and conferences. His research interests include the application of machine learning techniques in edge computing, the advancement of large language models (LLMs) for low-resource languages, and the exploration of blockchain technologies. He serves as an Associate Editor for *Telecommunication Systems* and *Cluster Computing*.

FAISAL REHMAN received the M.S. and Ph.D. degrees in computer science from COMSATS University Islamabad, Abbottabad, Pakistan, in 2010 and 2018, respectively. He is currently an Associate Professor with COMSATS University Islamabad. His research interests include recommender systems, green computing, and wired and wireless networks.

ABDALLAH NAMOUN (Member, IEEE) received the bachelor's degree in computer science, in 2004, and the Ph.D. degree in informatics from The University of Manchester, U.K., in 2009. He is currently a Full Professor of intelligent interactive systems with the Faculty of Computer and Information Systems and the Director of the AI Center, Islamic University of Madinah. He has extensive experience in leading complex research projects (worth more than 23 million Euros) with several distinguished SMEs, such as SAP, BT, and ATOS. He has authored more than 115 publications in research areas spanning intelligent systems, machine learning, human–computer interaction, smart cities, and multi-agent systems. His current research interests include integrating state-of-the-art artificial intelligence models in the development of interactive systems and smart spaces.

MAZHAR ALI received the M.S. degree from the Royal Institute of Technology (KTH), Stockholm, Sweden, in 2009, and the Ph.D. degree from the Department of Electrical and Computer Engineering, North Dakota State University (NDSU), Fargo, ND, USA, in 2015. He is currently an Associate Professor with COMSATS University Islamabad, Abbottabad, Pakistan. His current research interests include cloud computing, information security, smart health, and data and social network analysis.

ABDULLAH ALOURANI (Member, IEEE) received the bachelor's degree in computer science from Qassim University, Buraydah, Saudi Arabia, the master's degree in computer science from DePaul University, and the Ph.D. degree in computer science from the University of Illinois Chicago, Chicago, IL, USA. He is currently an Associate Professor with the Department of Management Information Systems, Qassim University. His research interests include cloud computing, software engineering, security, and artificial intelligence. He is a member of ACM.