IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 27 August 2022, accepted 9 September 2022, date of publication 22 September 2022,
date of current version 30 September 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3208584

RESEARCH ARTICLE

A Deep Learning Approach for Task Offloading in Multi-UAV Aided Mobile Edge Computing

MOSHIRA A. EBRAHIM[1], GAMAL A. EBRAHIM[1], HODA K. MOHAMED[1], AND SAMEH O. ABDELLATIF[2], (Senior Member, IEEE)

[1]Computer and Systems Engineering Department, Faculty of Engineering, Ain Shams University, Cairo 11517, Egypt
[2]Electrical Engineering Department, Faculty of Engineering, The British University in Egypt (BUE), El-Sherouk, Cairo 11837, Egypt

Corresponding author: Gamal A. Ebrahim (gamal.ebrahim@eng.asu.edu.eg)

ABSTRACT Computation offloading has proven to be an effective method for facilitating resource-intensive tasks on IoT mobile edge nodes with limited processing capabilities. Additionally, in the context of Mobile Edge Computing (MEC) systems, edge nodes can offload its computation-intensive tasks to a suitable edge server. Hence, they can reduce energy cost and speed up processing. Despite the numerous accomplished efforts in task offloading problems on the Internet of Things (IoT), this problem remains a research gap mainly because of its NP-hardness in addition to the unrealistic assumptions in many proposed solutions. In order to accurately extract information from raw sensor data from IoT devices deployed in complicated contexts, Deep Learning (DL) is a potential method. Therefore, in this paper, an approach based on Deep Reinforcement Learning (DRL) will be presented to optimize the offloading process for IoT in MEC environments. This approach can achieve the optimal offloading decision. A Markov Decision Problem (MDP) is used to formulate the offloading problem. Delay time and consumed energy are the main optimization targets in this work. The proposed approach has been verified using extensive simulations. Simulation results demonstrate that the proposed model can effectively improve the MEC system latency, energy consumption, and significantly outperforms the Deep Q Networks (DQNs) and Actor Critic (AC) approaches.

INDEX TERMS Deep learning, deep reinforcement learning, Internet of Things, mobile edge computing, task offloading.

I. INTRODUCTION

The 5G era networks has been realized based on networking technologies, innovations, and the new computing and communication paradigms [1]. Mobile Edge Computing (MEC) is one of the key technologies for computation distribution that boosts the performance of 5G cellular networks [2]. The main role of MEC is the minimization of communication latency between the user and the server. This behavior has a great importance for Internet of Things (IoT) environments. IoT has become an important area of research due to its rapid use in our daily lives and in industry. Therefore, It faces numerous challenges, including latency reduction, storage management, energy consumption, task offloading, etc [3]. Increasing the number of end devices in IoT environments leads to a corresponding increase in the number of possible actions. Consequently, it is crucial to enhance the availability and the terminal-to-terminal delay [4].

By offloading IoT tasks to resource-rich terminals in cooperative edge servers or clouds, mobile end devices can release intensive computation and storage. The end-to-end performance of IoT applications is nevertheless significantly impacted by the various service architectures and offloading techniques. Indeed, computing needs have a greater impact on the performance of IoT applications as compared with connectivity requirements. However, communication bandwidth represents the most important resource as the system expands to support more IoT devices. As a result, it becomes the primary component that directly affects performance as a whole [5]. Building an orchestral IoT architecture is thus a must

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

to include optimum solutions under various limitations for the best offloading location. Even though MEC has several benefits, it is still constrained by the positions of fixed towers. Consequently, it is difficult to deploy MEC servers at any time or location. Furthermore, there is a good chance that natural disasters could occasionally destroy the infrastructure. Additionally, mounting infrastructure in remote locations such as hotspots and mountains is nearly challenging. The IoT nodes are unable to completely service their users in the aforementioned conditions. Unmanned Aerial Vehicles (UAVs) with MEC servers installed on board can be used to support MEC systems by taking advantage of their flexibility and ease of deployment. This support is necessary for tasks that mobile users in hotspot locations or in emergent scenarios have been temporarily offloaded. In order to provide computing servers for mobile user's terminals in adaptable positions, UAV-aided MEC [6] is introduced. By adding additional compute resources to MEC servers, UAV-aided MEC speeds up calculations and increases the operating lifetime of mobile devices [7].

Deep Learning (DL) [8] has been widely used to learn and optimize a variety of issues for UAV-aided MEC [9]. However, most DL approaches require labelled historical data. Meanwhile, labelling the training data requires a significant amount of human effort. By engaging with MEC surroundings, Reinforcement Learning (RL) [10] can learn and improve UAV-assisted MEC without training data. Therefore, to reduce overall energy consumption, Deep Reinforcement Learning (DRL) [11] approaches can be used to provide effective task offloading, resource allocation, and UAV control. DRL uses RL and Deep Neural Networks (DNNs) for collecting the complicated states of MEC with UAV assistance.

When developing distributed decision-making solutions for wireless task offloading problems, conventional approaches such as convex optimization [12] and mixed-integer programming [13] are not always appropriate. Mostly because such models become more complicated for systems with numerous agents. This work essentially devises Markov Decision Problem (MDP) [14] to model task offloading according to real knowledge and tries to solve this problem using a DRL algorithm. Hence, this paper aims to develop a DRL model to solve task offloading problem for multi-UAV-aided MEC systems. The proposed approach avoids unrealistic assumptions such as ignoring user's device mobility. Hence, the transmitting channel noise is taken into consideration in addition to the coordination of mobile users and UAVs. This model seeks to maximize the stability of the entire system while minimizing the time and energy it uses. Maximizing stability means balancing the computation power of the system workload, extending operation time, and maximizing the total number of completed tasks.

The remainder of the paper is organized as follows: Section II presents an overview of MEC and 5G networks followed by Section III that surveys the different methods for using DRL to solve task offloading optimization problem. Section IV presents the related work followed by Section V which provides the details of the system model and problem formulation. Then, simulation results are presented in section VI. Finally, Section VII, concludes the paper.

II. MOBILE EDGE COMPUTING AND 5G NETWORKS

Cloud computing and IoT infrastructure are combined by the aid of MEC. By enabling the edge network to assure Quality of Experience (QoE), it puts storage, computing, and management closer to the end user. Because of this, it maximizes resource usage and brings in money for network operators [15]. To decrease latency and traffic in the backbone network, end users can get data from nearby base stations rather than from the huge regional data centers.

MEC is a modern communication paradigm that has been utilized for bringing the energy resources and computing capability to the edge of the radio access network. Hence, computation capabilities can be distributed over the network instead of being centralized [16]. Thus, MEC has proven to be crucial to the rollout of 5G. It is necessary to reconsider network and information delivery strategies in order to meet the Internet continually shifting demand. To maximize the potential of MEC systems, integration with upcoming 5G technologies is required. This integration can be shown in different aspects such as Internet Service Providers (ISPs), Software-Defined Networking (SDN), Network Function Virtualization (NFV), and UAVs communication. SDN and NFV support agility and flexibility in multi-tenant MEC environments[17]. Moreover, a combination of 5G networks and MEC enables ISPs to meet consumer demands [18]. By utilizing cloud computing infrastructure, 5G networks can be able to meet the higher bandwidth, availability, and lower latency demands of the new Internet services and applications. The development of services that support multi-service and multi-tenant infrastructure is accelerated by cloud service providers using SDN and NFV [19]. NFV along with SDN enables network operators to achieve a high degree of flexibility and cost-saving, resulting in simplified and extensible placement and management. Additionally, efficient utilization of network functions and more reduction in power usage can be achieved [20], [21]. MEC builds a virtualized infrastructure that can be installed near the edge of the network. MEC architecture has a lot in common with NFV, which enables seamless and effective application operation over multi-access networks [22]. MEC has attracted interest as a viable alternative to provide computation networks close to consumers. It extends the cloud capabilities to a decentralized cloud utilizing the same SDN and NFV concepts as the bigger 5G networks.

UAV-assisted wireless communication is a promising essential component of MEC ecosystem and 5G cellular system because of the advantages of the large coverage and capacity enhancement. UAVs can be used as airborne edge servers to carry out complex calculations that have been transferred from users on the ground. On the other hand,

UAVs have the ability to function as airborne users and can be connected to ground base stations [23].

Besides the adaptation of UAVs and SDN/NFV technologies within 5G cellular networks, Artificial Intelligence especially DL is used with MEC for intelligent decision making. It offers the chance to precisely identify and categorize the applications on mobile nodes and automate the process of creating adaptive network slicing [24]. Also, it can be adapted to predict user's actions and automate the dynamic network resource management in 5G networks. In the future, it can save operational cost for telecommunication firms while enhancing user's experience.

III. DRL FOR TASK OFFLOADING INTO MEC

One of the major unresolved issues in IoT environments is how to extract real-world IoT data from complex noisy environment that may confuse traditional approaches of machine learning. DL is regarded as the most effective technique to resolve this problem. It has been used for a variety of applications in IoT nodes with promising early outcomes [25]. Because of its high efficiency in handling complex data, deep learning plays a very important role in future IoT services.

Traditional approaches such as genetic algorithms require prior knowledge of mobile users' patterns and network parameters. These approaches may be preferred in static or slowly varying environments. In rapidly and/or randomly changing IoT wireless networks, there is a limited to no prior knowledge for guiding decision making. As a result, approaches based on machine learning should be preferred. Furthermore, training over the entire dataset is computationally impossible in such situations [26]. The small cell stations in 5G networks could be used for offloading tasks that have too heavy computations or greatly consume battery for the majority of users' mobile devices. This is mainly due to their proximity to the users and the projected high number of stations.

DRL is a deep learning technique that replaces Q-table in the basic RL algorithm with a deep neural network [27]. Involving deep learning in RL allows it to handle high-dimensional actions and state spaces. Additionally, learning efficiency is highly improved and the limitations of reinforcement learning are relaxed to a certain extent.

There are different DRL techniques that are both accurate and effectively tackle problems that are too complex for the traditional learning approaches such as Q-learning. DRL uses different types of algorithms to perform specific tasks such as DNN models that use gradient descent to minimize the cost function. Cost functions for 5G networks could include operating cost, latency, and downtime. Another example is Deep Q-learning Networks (DQN), which use DNN to avoid feature engineering while training the policy and setting of replay memories to continue using the gained knowledge in earlier stages to simulate scenarios with enormous state spaces [28]. Although DQN models can solve problems that have high-dimensional state spaces, they still cannot deal with the continuous action spaces. Hence, the most efficient type of DRL models is the Deep Deterministic Policy Gradient (DDPG) approach [29]. DDPG is an off-policy actor-critic model-free algorithm which can learn policies in continuous action spaces. A policy function and a Q-value function comprise the actor-critic algorithm. The policy function generates actions by acting as an actor. The Q-value function acts as a critic that evaluates the performance of the actor and directs the actor's subsequent actions.

Mobile edge computing can take the advantage of its proximity to the user to address a variety of challenges in 5G networks. These challenges often require automated DRL management of a series of increasingly complex tasks. Solutions that combine DRL for 5G with efficiency are best when they are made closer to the end-user in MEC rather than the core network. DRL has numerous applications in MEC networking domain including predicting traffic, mobility modelling, responsive resource allocation, effective energy usage, and data privacy [30]. Meanwhile, this paper focuses on adaptive resource allocation and computation offloading.

IV. RELATED WORK

DRL approaches can autonomously extract features while minimizing human effort and domain expertise required to collect distinguishing characteristics. Hence, they play a key role against the heterogeneity of edge computing environments. Therefore, DRL models can efficiently optimize the task offloading strategy and determines offloading policies. Additionally, online heavy computation iterations can be avoided by offline training.

Many research efforts have been conducted in this direction. For cooperative UAV-enabled MEC networks, the study in [31] presents a cooperative offloading strategy based on UAV-to-device interference mitigation. DRL-based optimization is investigated to obtain the optimal offloading decisions and resource management policies in order to maximize the long-term system utility. Here, the system utility of the DRL-based model is better than related solutions that use non-cooperative UAV edge computing methods. Meanwhile, the study in [32] introduces the multi-objective ant colony optimization approach based on RL. It has been proposed for accurate resource allocation among end-users depending on the cost of creating Q-tables and optimal allocation in MEC. Additionally, fast responsive task offloading based on Meta-Reinforcement Learning (MRL) is introduced in [33] to overcome the low sample efficiency of the original RL-based algorithm. MRL enables learning and updating policies according to new environments. Additionally, it enables the user's equipment to run the training process by using its own data with little computing resources. Mobile applications are modeled as directed acyclic graphs and the dynamic offloading process are modeled as multiple MDPs. Moreover, the study in [34] presents the task offloading problem in satellite-terrestrial edge computing networks, where tasks can be offloaded to the visible urban terrestrial cloud via satellite link. DRL-based task offloading is used to accelerate the learning process by dynamically adjusting the number

of candidate locations and the size of action space. The offloading problem is modeled as a mixed-integer programming problem where the offloading location and bandwidth allocation depend only on the current channel state. Furthermore, a reinforcement learning approach is presented in [35] for computational offloading of energy harvesting for IoT devices. This approach uses DRL algorithm with a transfer learning strategy to compress the state space dimensions, accelerate the learning rate, and enhance the offloading system performance and system utility.

A distributed offloading approach called best response-based offloading algorithm has been introduced using game theory [36]. In this approach, users' devices work together to reduce energy cost and latency cost. Moreover, the authors in [37] investigate UAV-assisted MEC system. In this system, the UAV provides a complementary computation resource to the terrestrial MEC system. UAV tries to maximize the expected long-term computation performance. The study investigates a proactive model based on DRL techniques. MEC system is established for offline training of the proactive DRL model. Furthermore, DDPG based computation offloading algorithm has been introduced in [38] to find the best offloading policy in a dynamic environment for UAV-assisted MEC. Also, it can enable a continuous action space offloading decision and UAV mobility but with only one UAV server and one offloading layer. Hence, in this work, DDPG algorithm is investigated for more complex and heterogeneous environment with more than one offloading layer.

Different from all these studies, this paper concentrates on task offloading in heterogeneous 5G networks. Therefore, an optimal multi-level offloading system is proposed. The first layer contains multiple UAV servers instead of a single UAV as in the previous studies. Meanwhile, the next layer contains multiple MEC servers. The existence of multiple UAV and MEC servers helps in increasing the number of available servers for offloading in each layer. Consequently, the offloading system can accept a large number of offloading requests, which reduces the local computing requirements. On the other hand, the last layer is the cloud server layer that is used for computation-intensive tasks. UAV servers are closer to the end devices than MEC servers to support mobility and to provide fast response to users' offloaded requests. The proposed model jointly schedules resource allocation and computation offloading in multi-UAV aided MEC through a DDPG approach.

V. SYSTEM MODEL AND PROBLEM FORMULATION

In this section, the task offloading problem is going to be illustrated. Then, it is modeled using MDP and the optimal offloading decision is computed using a DRL algorithm. Additionally, the architecture of the adopted edge computing is detailed.

A. MEC SYSTEM ARCHITECTURE

The proposed offloading system architecture is illustrated in Fig. 1, which is composed of N end-user devices, M edge servers, K UAVs, one cloud server, and a Central Offloading Controller (COC). The COC is deployed in MEC layer, hence, it can be a master MEC server with special and higher efficiency storage and computing resources. This COC is a DDPG-based task offloading agent that is mainly responsible for responding to task computing requests of the end user's devices. It receives offloaded requests from different end devices, then it uses the DDPG algorithm to decide where to offload the user's task. Moreover, it has a control on resource cooperation and coordination between the access network edge server and the UAV cluster, as well as resource allocation on the cloud server. Agent application can get environment information through monitoring devices. These devices are deployed on the user's device, MEC, and UAV. The environmental information can contain the user's device status, resource request conditions, and disposable resources on MEC, UAV, and cloud server. MEC and UAV-aided MEC servers bring computing resources to the network edge, which is close to the user's device. Therefore, it can overcome high network congestion and long transmission delay when compared with the case of depending only on cloud computing. Furthermore, COC works as an orchestrator that manipulates numerous user offloading requests and collects information to select the optimal computing terminal. As a result, it avoids high congestion load on MEC terminals compared with the case when COC does not exist. In fact, cloud computing and MEC are mutually beneficial. Cloud computing provides a wealth of applications and computing resources. Meanwhile, MEC has short delays, high stability, and adaptability to diverse network environments, making it an excellent choice for delay-sensitive services [39].

To meet the delay constraints, it is necessary to utilize the benefits of cloud computing besides MEC to offload tasks to locations with different computing and communication capabilities. As a result, this study attempts to combine heterogeneous computing resources and construct a collaborative Device-UAV-Edge-Cloud computing environment. A task offloading environment based on DDPG is distributed and linked to the four-tier hierarchy, which includes IoT device layer, UAV server layer, MEC server layer, and cloud server layer. The following subsections describe the main characteristics of each layer.

1) IoT LAYER

A network of interconnected IoT devices is present at this layer. Through wireless access points, each device can link to UAV, MEC, and cloud servers. The IoT user must make dynamic task offloading decisions for each cycle of offloading based on QoS requirements and the state of the network (transmission bandwidth, task size, available resources, etc.).

2) UAV LAYER

This layer contains lightweight MEC servers on UAVs, which can provide high mobility and flexible deployment. Hence, the processing delay can be reduced since this layer can

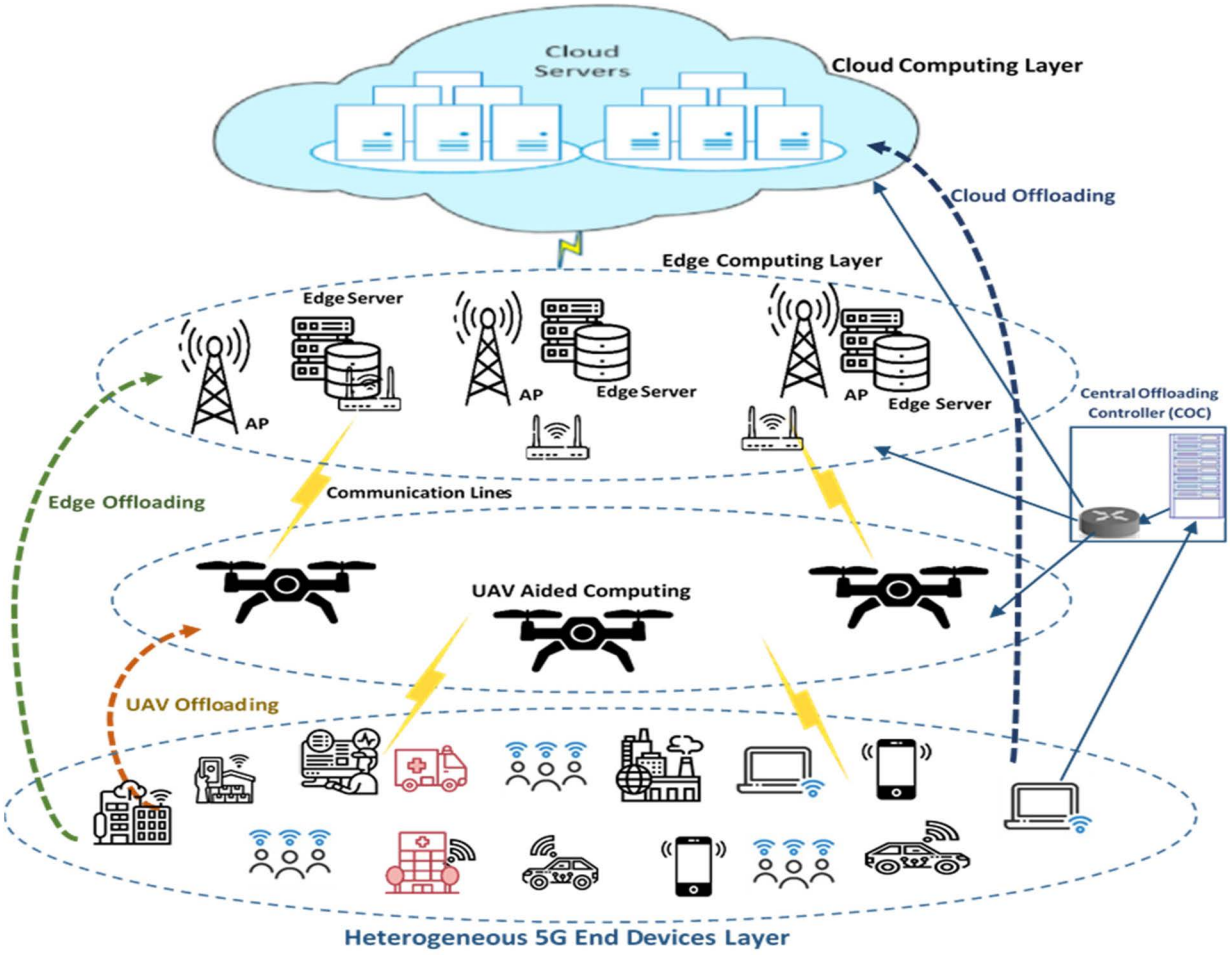

FIGURE 1. Multi-layer computational offloading system architecture.

offer computing support for jobs that mobile users offload in locations with temporary hotspots. For instance, sports stadiums or communities that have been devastated by natural disasters.

3) MEC LAYER

MEC servers for real-time task processing are present in this layer. They can offer lower latency computation services at the edge of the network. MEC servers may send complicated computational jobs to resource-rich cloud servers. In order to guarantee the security of offloading operations, MEC servers provide dependable communication with the IoT device layer and the cloud server layer.

4) CLOUD SERVER LAYER

This layer consists of numerous powerful virtual machines with higher storage and computational capacity. It is mostly utilized by IoT devices to do complicated computing tasks. Each cloud node in this layer is securely connected to MEC terminals and IoT nodes and runs in a decentralized safe manner.

B. OFFLOADING DELAY MODEL

Offloading delay model assumes that the user devices in the service scope are represented by $UD_N = UD_1, UD_2, \ldots, UD_n$. Each user has V independent tasks that need to be offloaded. Each task donated by the size of its data quantity to be offloaded Q_v and the task computation workload L_v (i.e., required CPU execution cycles). When the agent detects user's offloading request, it begins to develop an offloading strategy with the goal of minimizing task delay duration. The task delay is split into two portions: transmission delay and computation delay. The transmission delay represents the time required to offload the task to MEC, UAV, or cloud. While the computation delay represents the time needed for conducting computation on the server.

1) CLOUD COMPUTING OFFLOADING

The proposed model ignores the downlink delay. This is mainly because the size of the downloaded result of task execution is very small as compared with the uploaded task data size. The uplink data rate between the user's terminal and the cloud server through a wireless channel at moment t

is computed by (1).

$$r^t_{n,cloud} = \omega_c \log_2 (1+\frac{p_n g^t_{nc}}{\sigma^2}) \quad (1)$$

where ω_c represents the user-cloud channel bandwidth, g^t_{nc} identifies the channel gain loss between the user's device n and the cloud server c, p_n represents the transmitting power of user's device UD_n, and σ^2 represents the Gaussian noise. Therefore, the delay for task data transmission can be computed by (2).

$$T^{trans}_{nv,cloud} = \frac{q^t_{nv}}{r^t_{n,cloud}} \quad (2)$$

When computing task v is offloaded to a cloud server for execution then the time spent can be expressed by (3).

$$T^{exec}_{nv,cloud} = \frac{l_{nv}}{f_{cloud}} \quad (3)$$

where f_{cloud} denotes the computation frequency of the cloud server (CPU cycles per second).

2) MEC COMPUTING OFFLOADING

In MEC layer, MEC servers communicate with IoT devices through various wireless channels among LAN networks. The uplink data rate between the user's terminal and the MEC server via a wireless channel at moment t can be computed by (4).

$$r^t_{n,mec} = \omega_m \log_2(1 + \frac{p_n g^t_{nm}}{\sigma^2}) \quad (4)$$

where ω_m represents the user-MEC channel bandwidth, g^t_{nm} represents the channel gain loss between user's device n and MEC terminal m. Consequently, the delay for the task data transmission to MEC terminal can be computed by (5).

$$T^{trans}_{nv,mec} = \frac{q^t_{nv}}{r^t_{n,mec}} \quad (5)$$

When task v is offloaded to the MEC server for execution, the execution time is represented by (6).

$$T^{exec}_{nv,mec} = \frac{l_{nv}}{f_{MEC}} \quad (6)$$

where f_{MEC} represents the computation frequency of the MEC server (CPU cycles per second).

3) UAV COMPUTING OFFLOADING

UAV servers are lightweight MEC servers flying closely to IoT nodes. They communicate with IoT devices through wireless channels. The uplink data rate between the user's terminal and the UAV server via a wireless channel at moment t can be computed by (7).

$$r^t_{n,uav} = \omega_u \log_2(1 + \frac{p_n g^t_{nk}}{\sigma^2}) \quad (7)$$

where ω_u represents the user-UAV channel bandwidth and g^t_{nk} is the channel gain loss of the uplink between the UAV and user n, which can be expressed by (8) as in [40]:

$$g^t_{nk} = \frac{g_o}{d^2_{n,uav}} \quad (8)$$

where g_o represents the reference channel gain loss at a baseline distance $d = 1$m and $d^2_{n,uav}$ indicates the Euclidean distance between user n and the UAV.

The DDPG based controller agent in COC executes the action that decides if a task should be offloaded to the UAV, MEC, or cloud. Then, it drives UAVs to the new location if needed. In the 3D Cartesian coordinate system [41], the trajectory of the UAV can be indicated by the UAV discrete coordinates in each time slot, which is defined as $UV_k(t) = [x_k(t), y_k(t)] \in R^{2\times 1}$ at time slot t. Assume that UAV keeps flying at a fixed altitude H, where the UAV has a start coordinate $UV_k(t)$, and after it has been selected to execute the task of user's device UD_n, UV_k flies to a new coordinate $UV_k(t+1) = [x_k(t+1), y_k(t+1)] \in R^{2\times 1}$ with maximum flight speed ϑ_{max} to be closer to the corresponding user. The coordinate of user's deviceUD_n is $UD_n(t) = [x_n(t), y_n(t)] \in R^{2\times 1}$. Thus, $d^2_{n,uav}$ will be given by (9).

$$d_{n,uav} = \sqrt{\|UV_k(t+1) - UD_n(t)\|^2 + H^2} \quad (9)$$

At time slot t with a speed $\vartheta(t)$ and an angle$\beta \in [0, 2\pi]$. UAV flies from position $UV_k(t)$ to the new hover position $UV_k(t+1)$ given by (10).

$$UV_k(t+1) = \big[x(t) + \vartheta(t) t_{fly} \cos\beta(t), y(t) + \vartheta(t) t_{fly} \sin\beta(t)\big] \quad (10)$$

where t_{fly} donates the fixed UAV flight time. Consequently, the delay for task data transmission to UAV terminal can be computed by (11).

$$T^{trans}_{nv,uav} = \frac{q^t_{nv}}{r^t_{n,uav}} \quad (11)$$

When task v is offloaded to the MEC server for execution, the execution time can be represented by (12).

$$T^{exec}_{nv,uav} = \frac{l_{nv}}{f_{UAV}} \quad (12)$$

where f_{UAV} denotes the computing frequency of the MEC server (CPU cycles per second).

The offloading-decision matrix at moment t can be expressed by (13).

$$OL^t = \{ol^t_{i1}, ol^t_{i2}, \ldots, ol^t_{iv}, \ldots, ol^t_{nv}\} \quad (13)$$

where each ol^t_{nv} is the offloading action of task v of the user's device. It is a two-bit variable that can be selected from the set {0, 1, 2}. Where 0 represents offloading to the UAV layer, 1 represents offloading to the MEC layer, and 2 represents

offloading to the cloud layer. Therefore, the execution time can be computed by (14).

$$T_{nv}^{exc} = \begin{cases} T_{nv}^{UAV}, & ol_{nv}^{t} = 0, \ UAV\ offload \\ T_{nv}^{MEC}, & ol_{nv}^{t} = 1, \ MEC\ offload \\ T_{nv}^{cloud}, & ol_{nv}^{t} = 2, \ Cloud\ offload \end{cases} \tag{14}$$

Consequently, the total task delay T of user n at moment t (including the transmission and computation delays) can be computed by (15).

$$T_{nv}^{t} = \begin{cases} T_{nv,uav}^{trans} + T_{nv,uav}^{exec}, & ol_{nv}^{t} = 0, \ UAV\ offload \\ T_{nv,mec}^{trans} + T_{nv,mec}^{exec}, & ol_{nv}^{t} = 1, \ MEC\ offload \\ T_{nv,cloud}^{trans} + T_{nv,cloud}^{exec}, & ol_{nv}^{t} = 2, \ Cloud\ offload \end{cases} \tag{15}$$

C. OFFLOADING ENERGY MODEL

Various forms of consumed energy due to offloading process are considered in this model. One of the main forms of them is the consumed energy in data transmission of offloaded task to the computing terminal that has been selected by COC. Hence, the overhead for energy offloading by the user's device n at moment t can be computed by (16).

$$E_{vi}^{trans} = \begin{cases} p_i T_{nv,uav}^{trans}, & ol_{nv}^{t} = 0, \ UAV\ offload \\ p_i T_{nv,mec}^{trans}, & ol_{nv}^{t} = 1, \ MEC\ offload \\ p_i T_{nv,cloud}^{trans}, & ol_{nv}^{t} = 2, \ Cloud\ offload \end{cases} \tag{16}$$

When a task v_i is transferred to a cloud server to be computed then the energy consumed by the IoT device can be computed by (17).

$$E_{nv,cloud}^{exec} = p_{idle} T_{nv,cloud}^{exec} \tag{17}$$

where p_{idle} indicates the power consumed when the IoT device is idle, which means that the task is being offloaded elsewhere.

Consequently, when task v is transferred to the MEC server to be computed, the energy consumed by the IoT device can be described by (18).

$$E_{nv,mec}^{exec} = p_{idle} T_{nv,mec}^{exec} \tag{18}$$

For UAV offloading case, there are two additional energy components, which are the UAV energy needed for both flight and processing. The flight energy is consumed when the UAV flies to the airspace closest to a device position in order to aid in computing process; it can be expressed by (19) as in [42].

$$e_{uav}^{fly} = p_f \parallel \vartheta(t) \parallel^2 \tag{19}$$

where $p_f = 0.5 M_{uav} t_{fly}$ represents the power of flight of the UAV, M_{uav} represents the UAV mass, and $\vartheta(t)$ represents UAV flight speed at time t.

The stay energy is consumed when the UAV remains stationary at its current location for a while to perform the task$v$ for the user's device, The UAV stay consumed energy can be calculated by (20).

$$e_{uav}^{stay} = p_s . T_{nv,mec}^{exec} \tag{20}$$

where p_s represents the power of stay of the UAV.

Therefore, the energy consumption spent by the UAV server to execute task v is given by (21).

$$E_{uav}^{exec} = e_{uav}^{fly} + e_{uav}^{stay} \tag{21}$$

Moreover, the energy consumption spent by the user n device to execute task v on UAV server is given by (22).

$$E_{nv,uav}^{exec} = p_{idle} T_{nv,uav}^{exec} \tag{22}$$

Furthermore, the total consumed energy for transmitting and executing task v of user n at moment t can be computed by (23).

$$E_{nv}^{t} = \begin{cases} E_{nv,uav}^{trans} + E_{nv,uav}^{exec} & ol_{nv}^{t} = 0, \ UAV\ offload \\ E_{nv,mec}^{trans} + E_{nv,mec}^{exec} & ol_{nv}^{t} = 1, \ ME\ Coffload \\ E_{nv,cloud}^{trans} + E_{nv,cloud}^{exec}, & ol_{nv}^{t} = 2, \ Cloud\ offload \end{cases} \tag{23}$$

Additionally, the total latency and energy cost C_{nv}^{t} for transferring and executing task v of user n at moment t can be computed by (24).

$$C_{nv}^{t} = \begin{cases} \rho_t T_{nv}^{uav} + \rho_e E_{nv}^{uav}, & ol_{nv}^{t} = 0, \ UAV\ offload \\ \rho_t T_{nv}^{mec} + \rho_e E_{nv}^{mec}, & ol_{nv}^{t} = 1, \ MEC\ offload \\ \rho_t T_{nv}^{cloud} + \rho_e E_{nv}^{cloud}, & ol_{nv}^{t} = 2, \ Cloud\ offload \end{cases} \tag{24}$$

where ρ_t and ρ_e are constant weighting parameters corresponding to the time delay and the consumed energy for task v of user device n respectively.

Additionally, taking into account the user's mobility and the simplicity of the UAV processing, the system agent can gain a potential moving track of the user. Based on historical requests data and user's behavior patterns, it is possible to predict the number of tasks that the user may want to be offloaded. Meanwhile, the UAV movement pattern aids in obtaining the offloading plans, which save energy. As a result, the user's QoE and QoS can be considerably improved.

D. PROBLEM FORMULATION

The task offloading problem is represented as a server layer selection problem. In other words, each task group is mapped to a server-side, either UAV, MEC, or cloud server-side. MDP is adopted to model this problem instead of the classical modeling approaches such as convex optimization. Mainly because MDP is more efficient in modeling heterogeneous and stochastic real systems.

Based on equations (15), (23), and (24), the objective function O_{min} is optimizing MEC system by minimizing the total system cost of latency and consumed energy which can be expressed by (25).

$$\begin{aligned} O_{min} = & \sum_{n=1}^{N} \sum_{v=1}^{V} (1 - OL_{nv})(2 - OL_{nv}) C_{nv}^{uav} \\ & + OL_{nv}(2 - OL_{nv}) C_{nv}^{mec} + OL_{nv}(1 - OL_{nv}) C_{nv}^{cloud} \\ & subject\ to\ OL_{nv} \in 0, 1, 2\} \end{aligned}$$

$$\sum_{n=1}^{N}\sum_{v=1}^{V} E_{nv}^{exec} < E_{uav}^{battery},$$
$$OL_{nv} = 0,\ E_{nv}^{total} \geq 0$$
$$T_{nv} \leq \mathrm{d_v},\quad T_{nv} \geq 0 \tag{25}$$

where, $C_{nv}^{uav}, C_{nv}^{mec}$, *and* C_{nv}^{cloud} represent the total UAV offloading cost, total MEC offloading cost, and total cloud offloading cost respectively, $E_{uav}^{battery}$ is the maximum battery energy of UAV, and d_v is the maximum delay of task v.

According to the optimization problem in (25), it is considered an NP-hard problem based on the proof in [43]. Additionally, each offloaded task has three choices with respect to offloading decision variable OL_{nv}. Assume there are V tasks to be processed, hence, there are 3^{v+1} choices to offload task on, which demonstrates the complexity of the offloading optimization problem.

When taking into account MDP models, system states have a very high level of complexity. Additionally, continuous action space is required to support decision regarding offloading. Thus, a DDPG-based model is suggested in this study in order to determine the best policy for computation scheduling and offloading, UAVs mobility, and resource allocation in UAV-aided MEC based system.

DRL approaches, especially DDPG, are very well suited for learning in MDP based environments. This is primarily due to the fact that such approaches can be used to adapt to the collective action of other agents in the existence of information scarcity. The offloading problem is tackled by using MDP in conjunction with DDPG. In an MDP environment, the agent applies DDPG algorithm to learn the best action to be chosen in the next round.

E. DRL BASED OFFLOADING OPTIMIZATION ALGORITHM

DRL introduces a deep neural network to replace the Q-table in the RL algorithm. DRL differs from supervised learning in that the labelled data of RL emerges from recursive RL updates. As a result, the feedback varies with each iteration. Because of the environment noisy feedback, the DRL model may oscillate during exploitation and exploration. To address this issue, DRL employs target networks to supplement the DQN with fixed weights during specific episodes.

DDPG is an improved version of DQN to make DRL agent efficiently deal with continuous action space. As explained in Fig. 2, DDPG uses two separate DQNs for approximating the actor-network (policy-network) and the critic-network (Q-value network). In DDPG, both critic and actor networks have a target network with a similar structure as them.

1) STATE SPACE

In UAV-aided MEC environments, the state space is jointly described by N UDs, K UAVs, M MECs, and cloud and their surrounding environment. The system state at time slot t can be defined as $S_t = \{Q_{vi}, L_{vi}, F_s, UV_k, UD_i\}$:

- $Q_{vi} = [q_{i1}, q_{i2}, \ldots, q_{iv}, .., q_{nv}]$: A vector consists of the data size of offloaded tasks at time slot t.

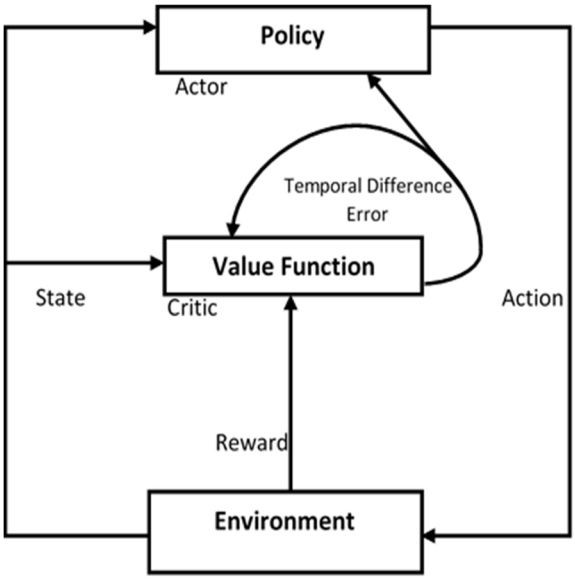

FIGURE 2. Actor-Critic (AC) algorithm.

- $L_{vi} = [l_{i1}, l_{i2}, \ldots, l_{iv}, .., l_{nv}]$: A vector consists of the required CPU cycles of offloaded tasks at time slot t.
- $F_s = [f_1, f_2, \ldots.., f_{k+m+1}]$: A vector consists of the remaining computing capacity of each available server at time slot t.
- $UV_k = [UV_1, UV_2, \ldots.., UV_k]$: A vector consists of the coordinates of each UAV at time slot t.
- $UD_n = [UD_1, UD_2, \ldots.., UD_n]$: A vector consists of the coordinates of each UD at time slot t.

For a given state S_t, the agent adopts action a_t according to the selected policy.

2) ACTION SPACE

Based on the current state S_t of the observed system environment parameters, the agent chooses a certain action a_t to offload the requested tasks of all mobile devices nodes to the available computing terminal servers. The action vector A_t can be represented as $A_t = \left\{OL^t, OS^t, \beta_k^t, FD_k^t\right\}$.

The action dimension OL^t represents the selected offloading layer where $OL^t \in \{0, 1, 2\}$. The next action dimension OS^t represents the selected server that the agent selects from the offloading layer where $OS^t \in [0, k+m+1]$. The last action dimensions β_k^t, FD_k^t represent the required flying angle and distance of selected UAV to serve the upcoming user's task.

3) REWARD FUNCTION

The behavior of DDPG agent is based on rewards. Hence, the effectiveness of the DDPG framework is greatly influenced by the selection of a suitable reward function. In order to optimize the reward, it is important to reduce the total processing time and the energy consumption as stated in (26).

$$R_t(s_t, a_t) = (1-\varepsilon) F_t - \rho_e \log_5\left(E_t^{total}\right) - \rho_t \log_3\left(T_t^{total}\right) + C \tag{26}$$

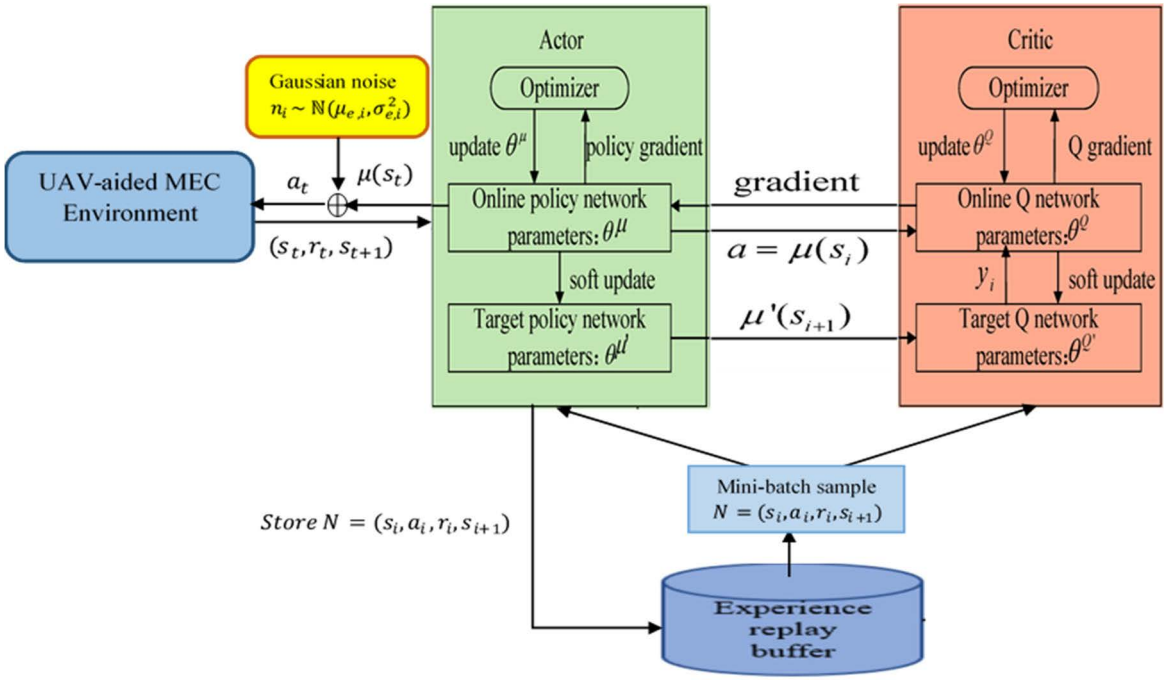

FIGURE 3. Deep deterministic policy gradient algorithm.

Equation (26) shows that whenever user task is completed before it expires, the agent is rewarded by a reward R_t. The task expiration state is described by a flag F_t. The agent punishment is represented in terms of energy and time cost through ρ_e and ρ_t factors. The total consumed energy E_t^{total} and the total delay T_t^{total} values are smoothed by a logarithmic function to avoid feedback of fluctuation in the learning model. Additionally, it is a time-consuming process if the original values are used. Moreover, C and ε are small constants that are randomly generated to encourage the model to keep running and accumulate rewards over time steps. Where, both ε and $C \in [0.1, 0.5]$.

4) PROPOSED DDPG-OFFLOADING ALGORITHM

DDPG learning approach is an advanced DRL algorithm. The DDPG is based on Actor-Critic (AC) algorithm that explained in Fig. 2 [44]. In AC algorithm, the actor explores the policy μ that maps the observation S_t of the agent to the action A_t. On the other hand, the critic evaluates actor's actions and estimates the Q-value function $Q(S_t)$. At time moment t, when the action a_t is taken by the actor, the agent will apply it on the environment and sends the current environment observation S_t along with the feedback to the critic. The feedback includes the reward r_t and the new observation S_{t+1}. Then, the critic computes the temporal difference error by (27).

$$\delta_t = R_t + \gamma Q(S_{t+1}|\mu) - Q(S_t|\mu) \tag{27}$$

where $\gamma \in (0, 1)$ is the discount factor.

The AC critic network is updated according to the optimal value function Q^* which aims to minimize the least square of temporal difference as in (28).

$$Q^* = arg\, min_{Q^{\mu}}(\delta_t)^2 \tag{28}$$

The DDPG algorithm illustrated in Fig. 3 is the efficient enhanced version of AC algorithm. This is mainly because DDPG uses four neural networks: a Q network (critic) θ^{Q}, a deterministic policy network (actor) θ^{μ}, a target Q network $\theta^{Q'}$, and a target policy network $\theta^{\mu'}$. The Q network and policy network are much similar to actor and critic networks [45]. However, in DDPG, the Actor is used to create a unique action by directly mapping states to actions instead of outputting the probability distribution across a discrete action space. On the other hand, critic is used to approximate the Q-value action function. The target networks are always time-delayed copies of their original networks that slowly track the learned networks. Therefore, using target value networks $\theta^{Q'}$ can greatly improve stability in the learning process. As explained in Algorithm 1, the actor network employs an approximated deterministic strategy μ to acquire deterministic action a_t that complies with state S_t and receives a reward R_t in addition to the new state S_{t+1}. The critic-network critiques and directs the actor performance using the approximated action-value function $Q[(S_t, a_t|\theta^{\mu})$.

The improvement of equation (28) in the proposed DDPG algorithm is that critic is updated by minimizing the sum of gradient update loss for each experience sample N. This improvement makes the DDPG model more effective and practical than AC based model. The improved loss equation is described in (29):

$$Loss\left(\theta^{Q}\right) = \frac{1}{N}\sum_t (y_t - Q(S_t, a_t/\theta^{Q}))^2 \tag{29}$$

DDPG algorithm adds experience replay buffer and target network properties to both actor and critic networks. These two properties reduce the correlation among the collected data and boost algorithm stability. The replay memory buffer contains the previous system state transition experience. In order to increase learning performance, it randomly selects samples from the buffer during training process. Besides the observed state transitions, the replay memory stores the rewards obtained as a result of actions throughout each time. Moreover, a group of N saved experiences are randomly chosen as samples from the replay memory for training the parameters of critic network.

Algorithm 1 DDPG-Based Computation Offloading for UAV-Aided MEC System

Initialize: UAVs-aided MEC environment including User's Devices (UDs) coordinates, UAVs coordinates, empty replay buffer B, actor weights, critic weights, and targets weights.
Input: Number of tasks V, task data size, task CPU cycles, task maximum delay, number of episodes T, batch sample length N, actor learning rate α, critic learning rate β, discounting factor γ, and replay buffer size L.

1. **for** each episode $t = 1$ to T **do**
2. Reset simulation network parameters and obtain initial observed state S_1.
3. **while** number of executed tasks $< V$ **do**
4. Select action $a_t = \mu(S_t|\theta) + n^e$ according to current environment state and exploration noise n^e.
5. Perform action a_t to obtain reward r_t and new state S_{t+1}.
6. **if** replay buffer B is not full **then**
7. Store transition (S_t, a_t, r_t, S_{t+1}) in B
8. **else**
9. Replace randomly a transition in replay buffer B with (S_t, a_t, r_t, S_{t+1})
10. Select randomly a mini-batch of N transitions from B
11. Set $y_t = R_t + \gamma Q'(S_{t+1}, \mu'(S_{t+1}|\theta^{\mu'}), \theta^{Q'})$
12. Update the critic network by minimizing the loss:

$$Loss\left(\theta^{Q}\right) = \frac{1}{N}\sum_{i}\left(y_t - Q(S_t, a_t|\theta^{Q})\right)^2$$

13/ Update actor policy θ^{μ} by the sampled policy gradient:

$$\nabla_{\theta^{\mu}} J \approx \frac{1}{N}\sum_{i}\left[\nabla_{a_t} Q\left(S_t, a_t|\theta^{Q}\right)\nabla_{\theta^{\mu}}\mu(S_t|\theta^{Q})\right]$$

14. Soft update target actor and critic networks by:
$\theta^{Q'} \leftarrow \theta^{Q} + (1-\tau)\,\theta^{Q'}$
$\theta^{\mu'} \leftarrow \theta^{\mu} + (1-\tau)\,\theta^{\mu'}$
15. **end if**
16. **end while**
17. **end for**

VI. SIMULATION RESULTS

In this section, the details of the simulation study are presented. PyTorch is adopted for developing the proposed DDPG based offloading environment. The adopted simulator in the experiments has three main components. These components are system environment, UAV-aided MEC, and the DDPG controller agent. The entire UAV-aided MEC offloading environment is described as MDP environment. The MDP environment is the focus of the DDPG model actions. Consequently, the proposed DDPG approach is compared with DQN and AC approaches.

TABLE 1. Simulation parameters.

Parameter	Value
Number of IoT application Tasks	$V = 50$
Number User's devices	$N = 10$
UD-UAV Bandwidth	$\omega_u = 30MHz$
UD-MEC Bandwidth	$\omega_m = 75MHz$
UD-Cloud Bandwidth	$\omega_c = 150MHz$
UAVs computation capacity	$f_{uav} = [2.8, 3.2]GHz$
MECs computation capacity	$f_{mec} = [2.8, 3.2]GHz$
Cloud computation capacity	$f_{cloud} = 20\ GHz$
User's device computation capacity	$f_{local} = [0.8,0.9]GHz$
User's device idle power	$P_{idle} = 0.3W$
User's device transmission power	$P_{tr} = 1.3W$
Task data size	$q_v = [400,1200]MB$
Task CPU cycles	$l_v = [200,600]cycles/b$
Task max. delay	$d_v = [0.01,0.1]s$
Actor learning rate (α)	0.0001
Critic learning rate (β)	0.001
Number of hidden layer (actor)	512
Number of hidden layer (critic)	256
Total number of episodes T	1000

A. SIMULATION SETTINGS

Simulation adopts 2D square areas, each of them has $N = 10$UDs randomly distributed in $300 \times 300m^2$ area. Additionally, it is assumed that the UAVs fly at a fixed height $H = 100m$. Each UAV has a unique mass $M_{uav} = 9.65KG$ [46] and a maximum flight speed $\vartheta = 50$m/s [42]. During the training phase, the batch and the buffer sizes are set to be 64 and 10^5, respectively. The rest of the key simulation parameters are described in details in Table 1.

B. PERFORMANCE EVALUATION

The results of the evaluation of DDPG-based computational offloading model is presented in this section. Adam optimizer [47] is adopted to train the DDPG agent, which is an adaptive learning optimization method. To optimize the offloading decision, the state of UAVs, MECs, cloud, users, and the other environment parameters are used as inputs to the actor-network. Meanwhile, the output is the UAV new position and the offloading decision. The input environment state parameters should include UAV current position, UAVs and MECs frequencies, the coordinates and frequencies of the users being served at the moment, and the parameters of the

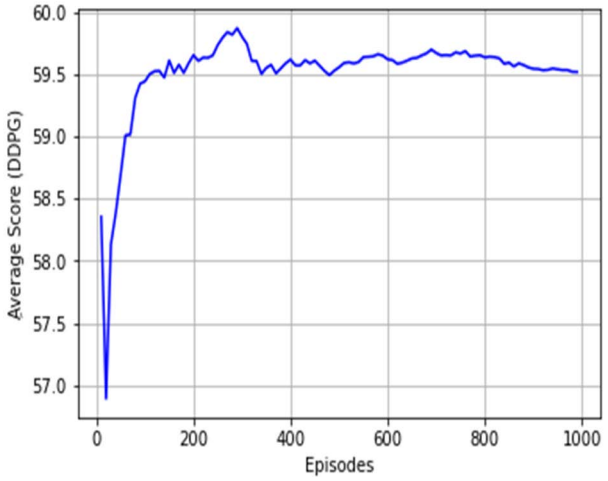

FIGURE 4. Average score of DDPG agent.

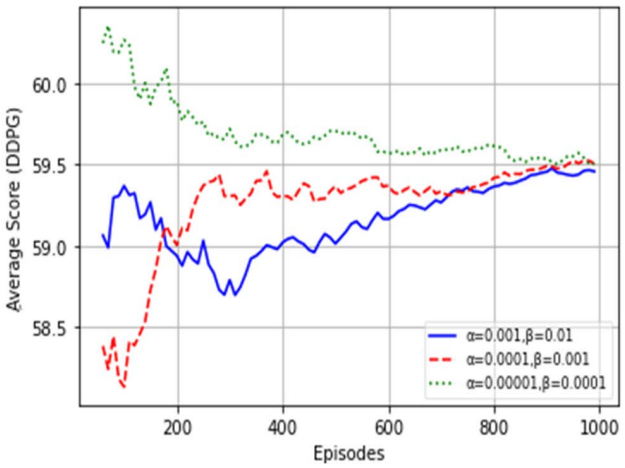

FIGURE 5. DDPG convergence under different learning rates.

offloaded tasks such as required computation cycles, data size, and expiration time.

It is assumed that there is a batch of task requests arriving at the DDPG agent controller system in each time slot and each UD generates its bulk task requests in each time slot. The performance of the offloading strategy is evaluated using the offloading time cost T_t^{total} and the energy cost E_t^{total} in each time slot. The offloading time cost is the total time taken by all served UDs to complete their tasks within the time slot. Meanwhile, the offloading energy cost is the total amount of energy used by all served UDs to complete their corresponding tasks within a time slot.

Fig. 4 shows the average score of the DDPG-based agent. As shown in this figure, the average score increases during the training interval T. This indicates that the performance of DDPG learning improves with the increase of training steps and achieves higher rewards in each episode. Moreover, the proposed DDPG-based agent can efficiently explore the MEC environment action space, which demonstrates that efficient task offloading policies can be successfully learned.

Based on the comparison among various learning rates for actor and critic networks, the convergence performance of the proposed algorithm is studied with different learning rates as shown in Fig. 5. It can be noticed that when $\alpha = 0.0001$ and $\beta = 0.001$, the proposed DDPG algorithm can have the best convergence. Very small learning rates such as $\alpha = 0.00001$ and $\beta = 0.0001$ make both actor and critic networks take more iterative episodes to converge. While larger learning rates such as ($\alpha = 0.001$ and $\beta = 0.01$) or ($\alpha = 0.0001$ and $\beta = 0.001$) make DDPG algorithm have faster convergence. Thus, it is clear that learning rates $\alpha = 0.0001$ and $\beta = 0.001$ are more stable and appropriate for the proposed DDPG algorithm convergence. Fig. 6 and Fig. 7 show the offloading time cost and energy cost of the proposed DDPG-based model. These figures show that both time cost and energy cost decrease over the interval T. these decays of both costs over time prove the efficiency of the proposed model.

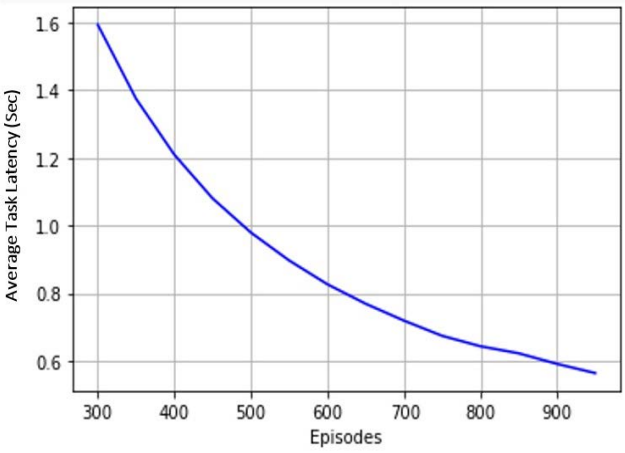

FIGURE 6. DDPG average time delay (seconds) for 50 tasks.

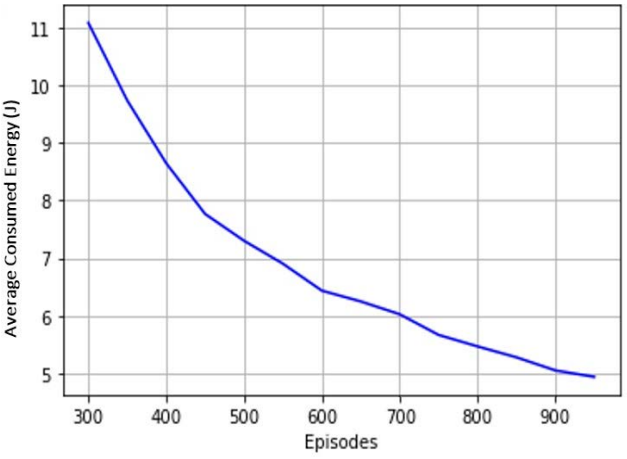

FIGURE 7. DDPG average consumed energy for 50 tasks.

Moreover, the correct offloaded task ratio is measured to evaluate the offloading decision strategy. Fig. 8 shows the correct offloaded task ratio of the DDPG-based model with

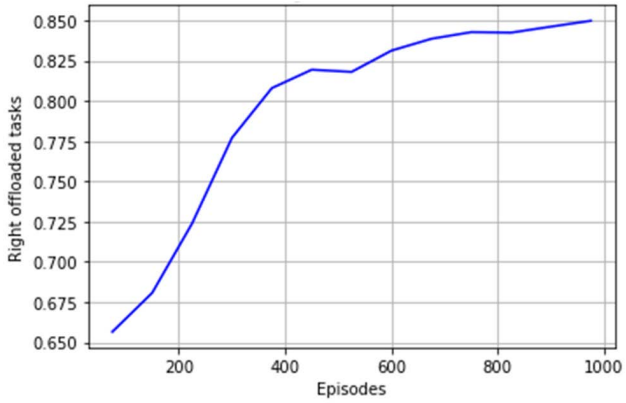

FIGURE 8. Correct offloaded tasks ratio.

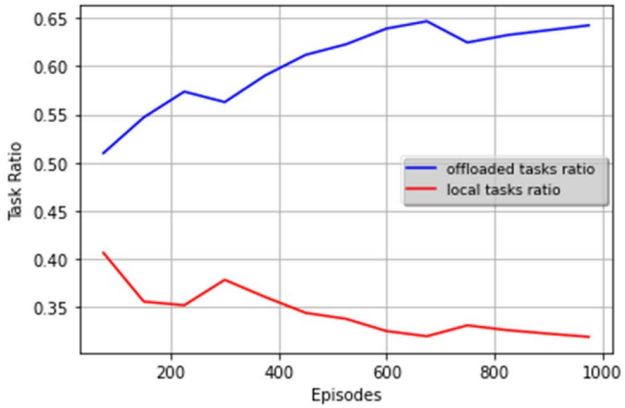

FIGURE 9. Offloading ratio versus local ratio.

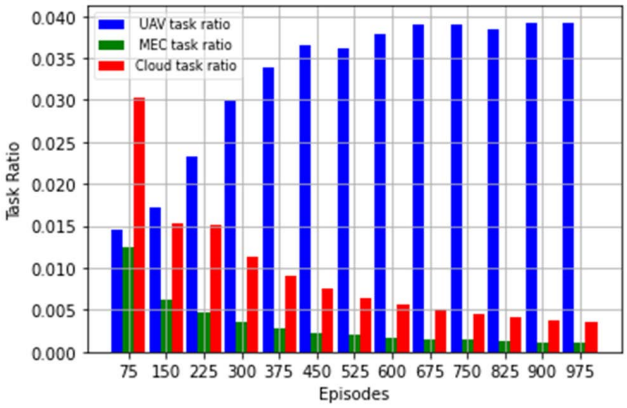

FIGURE 10. Offloaded task ratio to UAV, MEC and cloud layer.

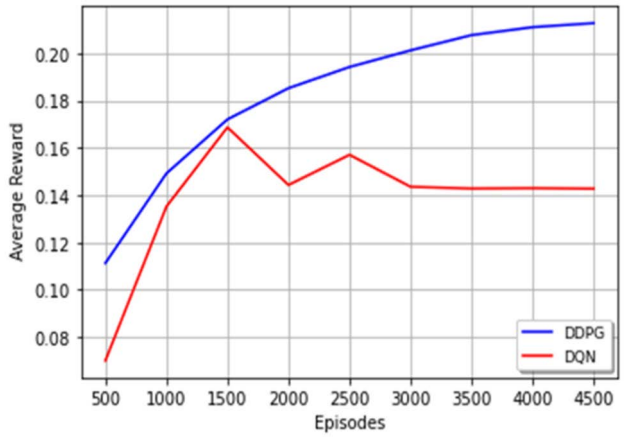

FIGURE 11. DDPG versus DQN average reward evaluation.

respect to all requested tasks. The right offloaded task is the task that has been offloaded and executed before being expired (i.e., $t^{exec} < task^{max\_delay}$). Meanwhile, Fig. 9 shows the offloading computational tasks ratio versus the local computed tasks ratio that has been achieved by the proposed model. As shown in this figure, the offloading ratio is much higher than the local ratio. Hence, the proposed model can achieve lower local computational load on user's devices. Additionally, Fig. 10 shows the percentage of tasks that have been offloaded to each system layer. This figure indicates that the number of tasks transferred to cloud and MEC layers is much lower than the number of tasks transferred to UAV layer. Also, it shows that the cloud task ratio decreases over time while the UAV task ratio increases. These observations proves that the DDPG can efficiently learn over time how to take the best offloading action that minimizes the offloading cost.

The performance of the proposed model is compared with three main models, which are edge offloading only, DQN-based offloading, and AC-based offloading. The comparison against edge offloading only is conducted to prove the positive effect gained when UAV layer exists. Meanwhile, DQN-based offloading and AC-based offloading are used because of the similarity between them and DDPG as all of them depend on the Q-value. Hence, the comparison is conducted to assess the efficiency of DDPG offloading decision over the DQN offloading decision.

1) DQN-BASED OFFLOADING

In this model, the agent applies DQN to select from offloading to UAV, MEC, or cloud server. Fig.11 shows that DDPG-based model achieves a larger average reward with a faster convergence than the DQN model over the time. Meanwhile, Fig. 12 shows that the proposed DDPG-offloading achieves higher average reward and lower task latency delay as compared with DQN-offloading over different numbers of offloaded tasks. These results show that the proposed model is more efficient in making the right offloading decision as compared with the DQN model.

2) EDGE OFFLOADING ONLY

In this model, the agent applies DDPG algorithm to select between two offloading choices, which are local offloading and edge offloading. Fig. 13 shows the comparison between the proposed model and the edge offloading only model. As shown in this figure, the consumed energy in edge-only

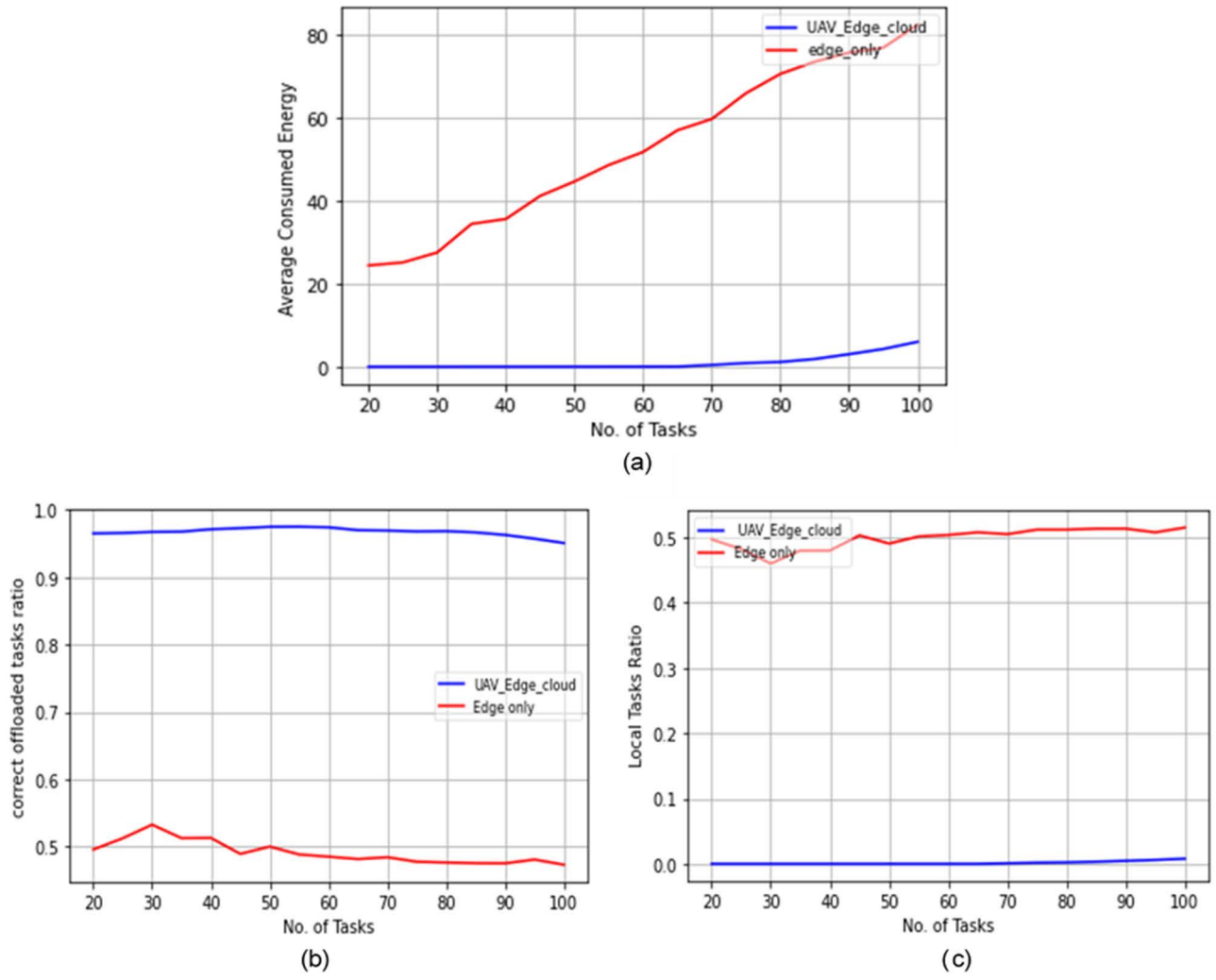

FIGURE 12. Comparison between the proposed model architecture and edge-only architecture over different numbers of offloaded tasks in terms of: a) Average consumed energy. b) Correct offloading Ratio. C) Local computing ratio.

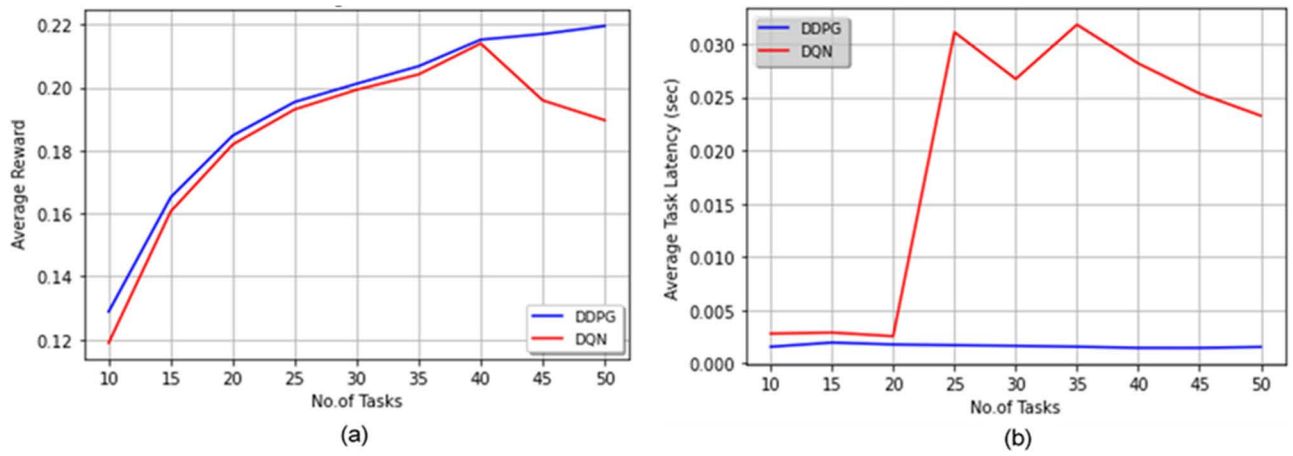

FIGURE 13. DDPG versus DQN comparison over different numbers of tasks.

scenario increases when the number of offloaded tasks increases. Hence, the UAV-Edge-Cloud proposed scenario can preserve lower energy consumption with the increase in the number of tasks. Additionally, it can be noticed that the lower percentage of correct offloaded tasks in edge-only scenario is compared with the corresponding higher value

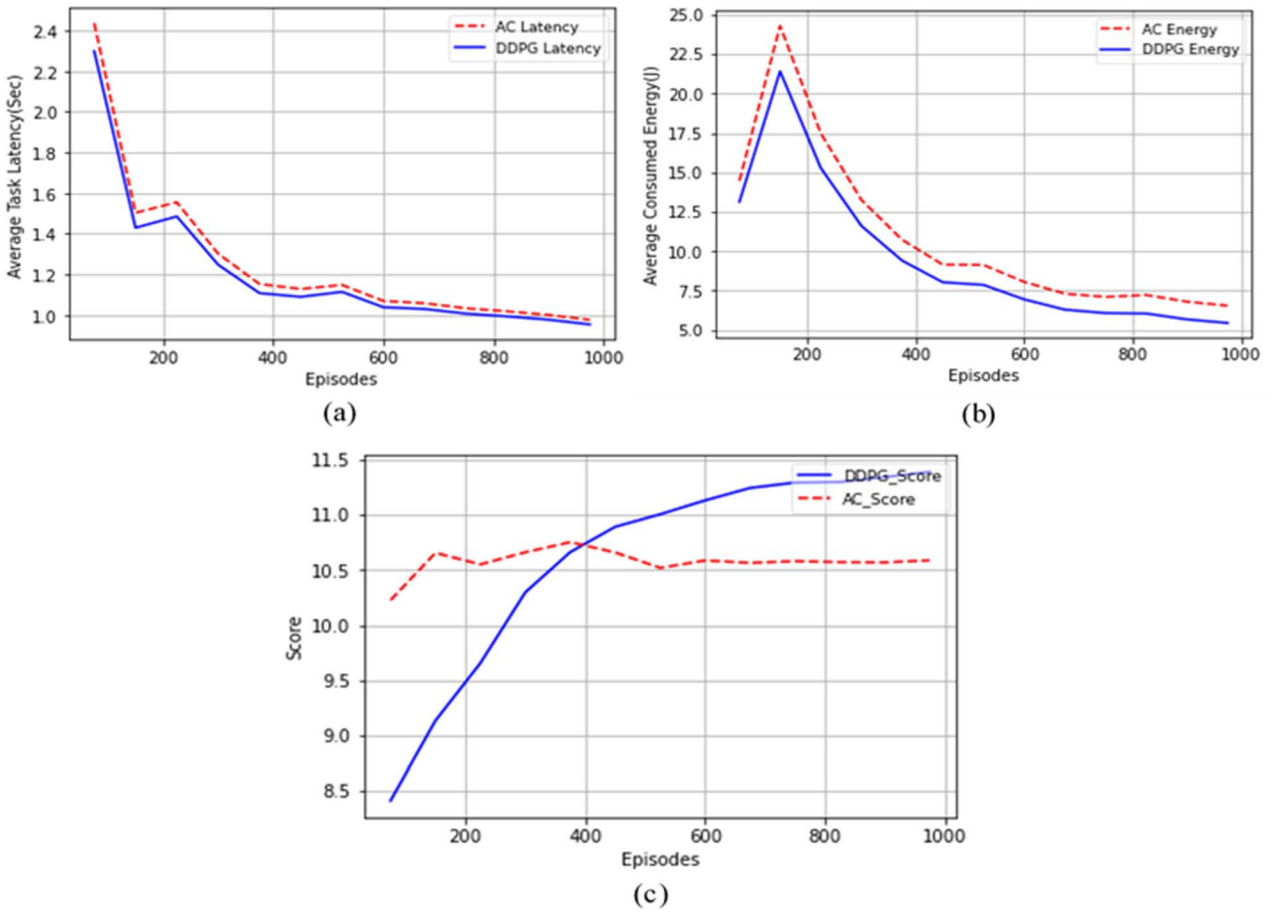

FIGURE 14. DDPG versus AC comparison in terms of: a) Average latency. b) Average consumed energy. C) Average agent score.

in the UAV-Edge-Cloud proposed scenario. Furthermore, as shown in Fig. 13, the number of local computed tasks ratio in the Edge-only scenario is larger than the corresponding value in UAV-Edge-Cloud proposed scenario. Therefore, this decrease in local computation rate can enhance system stability as compared with edge-only scenario. These comparative experiments prove that UAV existence with the MEC and the cloud can efficiently improve the performance of the offloading system.

3) AC-BASED OFFLOADING

In this model the agent applies AC algorithm to select where to offload users' tasks. Fig. 14 shows the comparison between the proposed model and AC model in terms of agent average score, consumed energy, and latency delay. This comparison indicates that the DDPG achieves higher score as compared with AC model. Also, DDPG based offloading system achieves lower energy cost and lower latency cost as compared with AC based offloading.

VII. CONCLUSION

Edge computing is evolving rapidly toward the fundamental infrastructure and facilitating the future of IoT. Efficient coordination mechanisms and task offloading models are leveraged to enable mobile devices and edge-cloud to cooperatively work. This paper investigated one of the efficient deep reinforcement learning algorithms, which is the DDPG algorithm. It proposed a DDPG-based offloading system to improve the efficiency of offloading decision strategy. It assessed the UAV benefits in 5G IoT environments to maximize the percentage of offloaded tasks from the total requested task. The paper proposed a DDPG model to tackle the offloading optimization problem for making correct decisions regarding offloading to one of the previously mentioned layers to reduce energy and time. It was demonstrated that DDPG performed better than DQN. Moreover, offloading to UAVs in cooperation with MECs and cloud servers resolves incomplete offloaded task requests. Hence, the three-layer offloading system and the deep learning-based algorithms can serve most of the task offloading requests and can achieve effective offloading decisions and resource management.

In future work, an improvement of the proposed model will be made to maximize the offloading system stability in dynamic and uncontrollable networking environments.

REFERENCES

[1] T. Dragičević, P. Siano, and S. R. S. Prabaharan, "Future generation 5G wireless networks for smart grid: A comprehensive review," *Energies*, vol. 12, no. 11, p. 2140, Jun. 2019, doi: 10.3390/en12112140.

[2] I. Al Ridhawi, M. Aloqaily, Y. Kotb, Y. Al Ridhawi, and Y. Jararweh, "A collaborative mobile edge computing and user solution for service composition in 5G systems," *Trans. Emerg. Telecommun. Technol.*, vol. 29, no. 11, p. e3446, Nov. 2018, doi: 10.1002/ett.3446.

[3] S. Nižetić, P. Šolić, D. López-de-Ipiña González-de-Artaza, and L. Patrono, "Internet of Things (IoT): Opportunities, issues and challenges towards a smart and sustainable future," *J. Cleaner Prod.*, vol. 274, Nov. 2020, Art. no. 122877, doi: 10.1016/j.jclepro.2020.122877.

[4] M. S. Hossain, C. I. Nwakanma, J. M. Lee, and D.-S. Kim, "Edge computational task offloading scheme using reinforcement learning for IIoT scenario," *ICT Exp.*, vol. 6, no. 4, pp. 291–299, Dec. 2020, doi: 10.1016/j.icte.2020.06.002.

[5] J. Almutairi and M. Aldossary, "Modeling and analyzing offloading strategies of IoT applications over edge computing and joint clouds," *Symmetry*, vol. 13, no. 3, p. 402, Mar. 2021, doi: 10.3390/sym13030402.

[6] W. Zhang, L. Li, N. Zhang, T. Han, and S. Wang, "Air-ground integrated mobile edge networks: A survey," *IEEE Access*, vol. 8, pp. 125998–126018, 2020, doi: 10.1109/ACCESS.2020.3008168.

[7] L. Zhang, Z.-Y. Zhang, L. Min, C. Tang, H.-Y. Zhang, Y.-H. Wang, and P. Cai, "Task offloading and trajectory control for UAV-assisted mobile edge computing using deep reinforcement learning," *IEEE Access*, vol. 9, pp. 53708–53719, 2021, doi: 10.1109/ACCESS.2021.3070908.

[8] Z. Zhou, H. Liao, B. Gu, K. M. S. Huq, S. Mumtaz, and J. Rodriguez, "Robust mobile crowd sensing: When deep learning meets edge computing," *IEEE Netw.*, vol. 32, no. 4, pp. 54–60, Jul. 2018, doi: 10.1109/MNET.2018.1700442.

[9] F. Jiang, K. Wang, L. Dong, C. Pan, W. Xu, and K. Yang, "Deep-learning-based joint resource scheduling algorithms for hybrid MEC networks," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 6252–6265, Jul. 2020, doi: 10.1109/JIOT.2019.2954503.

[10] A. M. Andrew, "Reinforcement learning: An introduction," *Kybernetes*, vol. 27, no. 9, pp. 1093–1096, 1998, doi: 10.1108/k.1998.27.9.1093.3.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.

[12] H. Mei, K. Yang, Q. Liu, and K. Wang, "Joint trajectory-resource optimization in UAV-enabled edge-cloud system with virtualized mobile clone," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 5906–5921, Jul. 2020, doi: 10.1109/JIOT.2019.2952677.

[13] Q. Wang, A. Gao, and Y. Hu, "Joint power and QoE optimization scheme for multi-UAV assisted offloading in mobile computing," *IEEE Access*, vol. 9, pp. 21206–21217, 2021, doi: 10.1109/ACCESS.2021.3055335.

[14] O. Alagoz, H. Hsu, A. J. Schaefer, and M. S. Roberts, "Markov decision processes: A tool for sequential decision making under uncertainty," *Med. Decis. Making*, vol. 30, no. 4, pp. 474–483, Jul. 2010, doi: 10.1177/0272989X09353194.

[15] M. McClellan, C. Cervelló-Pastor, and S. Sallent, "Deep learning at the mobile edge: Opportunities for 5G networks," *Appl. Sci.*, vol. 10, no. 14, p. 4735, Jul. 2020, doi: 10.3390/app10144735.

[16] Q.-V. Pham, F. Fang, V. N. Ha, M. J. Piran, M. Le, L. B. Le, W.-J. Hwang, and Z. Ding, "A survey of multi-access edge computing in 5G and beyond: Fundamentals, technology integration, and state-of-the-art," *IEEE Access*, vol. 8, pp. 116974–117017, 2020, doi: 10.1109/ACCESS.2020.3001277.

[17] N. Kiran, X. Liu, S. Wang, and C. Yin, "VNF placement and resource allocation in SDN/NFV-enabled MEC networks," in *Proc. IEEE Wireless Commun. Netw. Conf. Workshops*, Apr. 2020, pp. 1–6, doi: 10.1109/WCNCW48565.2020.9124910.

[18] M. McClellan, C. Cervelló-Pastor, and S. Sallent, "Deep learning at the mobile edge: Opportunities for 5G networks," *Appl. Sci.*, vol. 10, no. 14, p. 4735, Jul. 2020, doi: 10.3390/app10144735.

[19] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—A key technology towards 5G," *IEEE Internet Things J.*, vol. 11, no. 11, pp. 1–16, 2015.

[20] M. A. Abdelaal, G. A. Ebrahim, and W. R. Anis, "High availability deployment of virtual network function forwarding graph in cloud computing environments," *IEEE Access*, vol. 9, pp. 53861–53884, 2021, doi: 10.1109/ACCESS.2021.3068342.

[21] M. A. Abdelaal, G. A. Ebrahim, and W. R. Anis, "Efficient placement of service function chains in cloud computing environments," *Electronics*, vol. 10, no. 3, p. 323, Jan. 2021, doi: 10.3390/electronics10030323.

[22] K. Antevski, C. J. Bernardos, L. Cominardi, A. de la Oliva, and A. Mourad, "On the integration of NFV and MEC technologies: Architecture analysis and benefits for edge robotics," *Comput. Netw.*, vol. 175, Jul. 2020, Art. no. 107274, doi: 10.1016/j.comnet.2020.107274.

[23] Z. Ullah, F. Al-Turjman, and L. Mostarda, "Cognition in UAV-aided 5G and beyond communications: A survey," *IEEE Trans. Cognit. Commun. Netw.*, vol. 6, no. 3, pp. 872–891, Sep. 2020, doi: 10.1109/TCCN.2020.2968311.

[24] A. Nakao and P. Du, "Toward in-network deep machine learning for identifying mobile applications and enabling application specific network slicing," *IEICE Trans. Commun.*, vol. E101.B, no. 7, pp. 1536–1543, Jul. 2018, doi: 10.1587/transcom.2017CQI0002.

[25] M. A. Al-Garadi, A. Mohamed, A. K. Al-Ali, X. Du, I. Ali, and M. Guizani, "A survey of machine and deep learning methods for Internet of Things (IoT) security," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 1646–1685, 2020, doi: 10.1109/COMST.2020.2988293.

[26] H. Li, K. Ota, and M. Dong, "Learning IoT in edge: Deep learning for the Internet of Things with edge computing," *IEEE Netw.*, vol. 32, no. 1, pp. 96–101, Jan. 2018, doi: 10.1109/MNET.2018.1700202.

[27] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, and S. Petersen, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.

[28] M. J. J. Douglass, "Book review: Hands-on machine learning with scikit-learn, keras, and tensorflow, 2nd edition by Aurélien Géron," *Phys. Eng. Sci. Med.*, vol. 43, no. 3, pp. 1135–1136, Sep. 2020, doi: 10.1007/s13246-020-00913-z.

[29] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2016, *arXiv:1509.02971*.

[30] Z. Xiong, Y. Zhang, D. Niyato, R. Deng, P. Wang, and L.-C. Wang, "Deep reinforcement learning for mobile 5G and beyond: Fundamentals, applications, and challenges," *IEEE Veh. Technol. Mag.*, vol. 14, no. 2, pp. 44–52, Jun. 2019, doi: 10.1109/MVT.2019.2903655.

[31] Y. Liu, S. Xie, and Y. Zhang, "Cooperative offloading and resource management for UAV-enabled mobile edge computing in power IoT system," *IEEE Trans. Veh. Technol.*, vol. 69, no. 10, pp. 12229–12239, Oct. 2020, doi: 10.1109/TVT.2020.3016840.

[32] S. Vimal, M. Khari, N. Dey, R. G. Crespo, and Y. Harold Robinson, "Enhanced resource allocation in mobile edge computing using reinforcement learning based MOACO algorithm for IIOT," *Comput. Commun.*, vol. 151, pp. 355–364, Feb. 2020, doi: 10.1016/j.comcom.2020.01.018.

[33] I. Alghamdi, C. Anagnostopoulos, and D. P. Pezaros, "Data quality-aware task offloading in mobile edge computing: An optimal stopping theory approach," *Future Gener. Comput. Syst.*, vol. 117, pp. 462–479, Apr. 2021, doi: 10.1016/j.future.2020.12.017.

[34] D. Zhu, H. Liu, T. Li, J. Sun, J. Liang, H. Zhang, L. Geng, and Y. Liu, "Deep reinforcement learning-based task offloading in satellite-terrestrial edge computing networks," in *Proc. IEEE Wireless Commun. Netw. Conf.*, Mar. 2021, pp. 1–7, doi: 10.1109/wcnc49053.2021.9417127.

[35] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu, and W. Zhuang, "Learning-based computation offloading for IoT devices with energy harvesting," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1930–1941, Feb. 2019, doi: 10.1109/TVT.2018.2890685.

[36] B. Wu, J. Zeng, L. Ge, X. Su, and Y. Tang, "Energy-latency aware offloading for hierarchical mobile edge computing," *IEEE Access*, vol. 7, pp. 121982–121997, 2019, doi: 10.1109/ACCESS.2019.2938186.

[37] X. Chen, T. Chen, Z. Zhao, H. Zhang, M. Bennis, and Y. Ji, "Resource awareness in unmanned aerial vehicle-assisted mobile-edge computing systems," in *Proc. IEEE 91st Veh. Technol. Conf. (VTC-Spring)*, May 2020, pp. 1–6, doi: 10.1109/VTC2020-Spring48590.2020.9128981.

[38] Y. Wang, W. Fang, Y. Ding, and N. Xiong, "Computation offloading optimization for UAV-assisted mobile edge computing: A deep deterministic policy gradient approach," *Wireless Netw.*, vol. 27, no. 4, pp. 2991–3006, May 2021, doi: 10.1007/s11276-021-02632-z.

[39] T. Zhao, S. Zhou, X. Guo, and Z. Niu, "Tasks scheduling and resource allocation in heterogeneous cloud for delay-bounded mobile edge computing," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2017, pp. 1–7, doi: 10.1109/ICC.2017.7996858.

[40] J. Xiong, H. Guo, and J. Liu, "Task offloading in UAV-aided edge computing: Bit allocation and trajectory optimization," *IEEE Commun. Lett.*, vol. 23, no. 3, pp. 538–541, Mar. 2019, doi: 10.1109/LCOMM.2019.2891662.

[41] X. Cheng, F. Lyu, W. Quan, C. Zhou, H. He, W. Shi, and X. Shen, "Space/aerial-assisted computing offloading for IoT applications: A learning-based approach," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 1117–1129, May 2019, doi: 10.1109/JSAC.2019.2906789.
[42] Q. Hu, Y. Cai, G. Yu, Z. Qin, M. Zhao, and G. Y. Li, "Joint offloading and trajectory design for UAV-enabled mobile edge computing systems," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1879–1892, Apr. 2019, doi: 10.1109/JIOT.2018.2878876.
[43] K. Li, X. Wang, Q. Ni, and M. Huang, "Entropy-based reinforcement learning for computation offloading service in software-defined multi-access edge computing," *Futur. Gener. Comput. Syst.*, vol. 136, pp. 241–251, 2022, doi: 10.1016/j.future.2022.06.002.
[44] H. Wang, P. Zhang, and Q. Liu, "An actor-critic algorithm using cross evaluation of value functions," *IAES Int. J. Robot. Autom.*, vol. 7, no. 1, p. 39, Mar. 2018, doi: 10.11591/ijra.v7i1.pp39-47.
[45] Y. T. Liu, J. M. Yang, L. Chen, T. Guo, and Y. Jiang, "Overview of reinforcement learning based on value and policy," *Chin. Control Decis. Conf. (CCDC)*, Aug. 2020, pp. 598–603, doi: 10.1109/CCDC49329.2020.9164615.
[46] S. Jeong, O. Simeone, and J. Kang, "Mobile edge computing via a UAV-mounted cloudlet: Optimization of bit allocation and path planning," *IEEE Trans. Veh. Technol.*, vol. 67, no. 3, pp. 2049–2063, Mar. 2018, doi: 10.1109/TVT.2017.2706308.
[47] S. Bock and M. Weiß, "A proof of local convergence for the Adam optimizer," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2019, pp. 1–8, doi: 10.1109/IJCNN.2019.8852239.

MOSHIRA A. EBRAHIM received the B.Sc. and M.Sc. degrees from the Faculty of Engineering, Ain Shams University, Cairo, Egypt, in 2009 and 2018, respectively. She is currently a Teacher Assistant with the Computer and Systems Engineering Department, Modern Academy for Engineering and Technology. Her research interests include intelligent systems, software engineering, cloud computing, computer networks, and artificial intelligence.

GAMAL A. EBRAHIM received the B.Sc. and M.Sc. degrees from the Faculty of Engineering, Ain Shams University, Cairo, Egypt, in 1994 and 2000, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Miami, Coral Gables, FL, USA, in 2004. He is currently a Professor with the Computer and Systems Engineering Department, Faculty of Engineering, Ain Shams University. His current research interests include machine learning, intelligent systems, computer networks, cloud computing, and distributed multimedia communication.

HODA K. MOHAMED received the B.Sc., M.Sc., and Ph.D. degrees from the Faculty of Engineering, Ain Shams University, Cairo, Egypt, in 1978, 1983, and 1992, respectively. She has been a Professor at the Computer and Systems Engineering Department, Faculty of Engineering, Ain Shams University, since 2009. Her research interests include intelligent systems, e-learning systems, data mining, database systems, software engineering, natural language processing, cloud computing, and image processing.

SAMEH O. ABDELLATIF (Senior Member, IEEE) received the B.Sc. degree in electronics and communication and the M.Sc. degree in semiconductor nanostructures from Ain Shams University, Cairo, Egypt, in 2009 and 2012, respectively. His Ph.D. work was funded by DAAD and awarded from Ain Shams University, in 2017, it was focusing on the utilization of nano-photonic structures in enhancing solar cells efficiency. Currently, he is enrolled as a Lecturer with the Electrical Engineering Department, The British University in Egypt (BUE), where he is the Sub-Group Leader at the Fab Laboratory, Centre of Emerging Learning Technology (CELT). In addition, he has collaborative research with the Max-Planck-Institut für Kohlenforschung, Mülheim, Germany, the University of Duisburg-Essen, Germany, and the University of Glasgow, U.K.