IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 26 December 2023, accepted 20 February 2024, date of publication 1 March 2024, date of current version 7 March 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3371993

RESEARCH ARTICLE

DRL-Based Distributed Task Offloading Framework in Edge-Cloud Environment

HEBA NASHAAT, (Member, IEEE), WALAA HASHEM, RAWYA RIZK, (Senior Member, IEEE), AND RADWA ATTIA

Electrical Engineering Department, Port Said University, Port Said 42526, Egypt

Corresponding author: Heba Nashaat (hebanashaat@eng.psu.edu.eg)

ABSTRACT The Internet of Things (IoT) and real-time media streaming have increased due to the rapid development of wireless communication technologies and the enormous growth of computation and data transmission tasks. Edge-Cloud Computing (ECC) combines the benefits of Mobile Cloud Computing (MCC) and Mobile Edge Computing (MEC) to meet energy consumption and delay requirements, and achieve more stable and affordable task execution. The most significant challenge in ECC is making real-time task offloading decisions. In order to generate offloading decisions in ECC environments in an efficient and near optimal manner, a Deep Reinforcement Learning (DRL)-based Distributed task Offloading (DRL-DO) framework is proposed. The Keras ML library is used to implement and evaluate the proposed DRL-DO and other offloading algorithms in Python experiments. Experimental results demonstrate the accuracy of the DRL-DO framework; it achieves a high Gain Ratio (GR) of about 22.3% and greatly reduces the energy consumption, response time, and system utility by about 7.6%, 43%, and 26.2%, respectively, while attaining moderate time cost compared with other offloading algorithms.

INDEX TERMS IoT, MEC, MCC, ECC, DRL, task offloading.

I. INTRODUCTION

With the rapid development of new computing and wireless communication technologies, tremendous growth of computation, storage, and data transmission tasks have emerged, leading to the spread of the Internet of Things (IoT), Internet of Vehicles (IoV), virtual / argument reality, real-time media streaming, etc. Centralized Mobile Cloud Computing (MCC) [1] is used to provide the execution of computation-intensive IoT applications however, it cannot efficiently satisfy the service demand of sensitive IoT applications. As a novel decentralized computing paradigm, Fog Computing (FC) or Mobile Edge Computing (MEC) [2], [3] harnesses computing, storage and data resources in the proximity of physical devices in order to bridge users and Edge Servers (ES). MEC can extremely improve the Quality of Service (QoS) and reduce tasks' execution delay, energy consumption, network congestion, and latency problems but still face many challenges such as the proper tasks' server resources assignment and low computing power of ES and Cloud Server (CS). As shown in Fig. 1, Edge-Cloud Computing (ECC) can be used to better serve IoT users' diverse requirements; it combines the advantages of both MCC and MEC to overcome the resource lack of smart devices, meet delay and energy consumption requirements, and achieve cheaper and more stable task execution. Real-time task offloading decision is the most important challenge in ECC [4], [5]. Generally, the offloading calculation process may be executed; locally for low computing power requirements' tasks, fully in ES, or partially by splitting the calculation process locally and to the ES.

The decision of task offloading in ECC from user devices either to edge or cloud doesn't come without difficulties due to the heterogeneity between edge and cloud regarding resources, network complexity and edge/cloud task assignment. Energy consumption, latency, QoS, response time, Quality of Experience (QoE), and cost metrics should be considered as a multi-objective challenge in offloading. The imposed workload should be distributed evenly among MEC servers to avoid congestion and in turn,

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

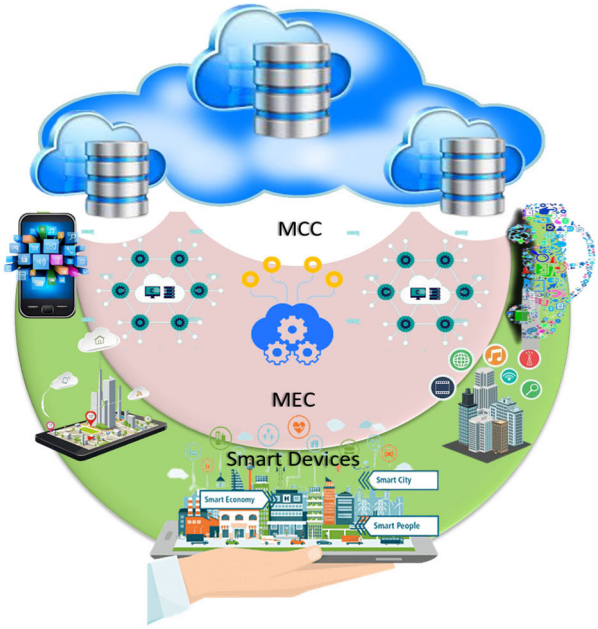


FIGURE 1. Edge cloud computing (ECC) architecture.

significant various offloaded tasks' response time and delay.

In practical scenarios, the rapid changes of the whole system parameters lead to a great demand for fast offloading decision-making and resource allocation. In addition, with the increasing number of users and tasks, conventual and heuristic task offloading techniques must be applied to execute the offloading decision and solve complex and large amount of computation problems. Traditional approaches [6], [7], [8], [9] can obtain stable management and scheduling decisions, however, large-scale ECC network always take too much time, which is not practical for real life applications. Deep Reinforcement Learning (DRL) is a subfield of Machine Learning (ML) [10] methodology that combines the Reinforcement Learning (RL) and Deep Neural Network (DNN) to facilitate and obtain the optimal offloading decision-making in volumes of communication and computation. DRL agents help solve complex problems in dynamic and stochastic environments and large state space, as they can accurately learn the optimal policy and long-term rewards without prior knowledge of the system thus, it is more practical, intelligent and inevitable to apply DRL instead of traditional methods.

Although the DRL has been demonstrated to be the most effective technique in several papers [11], [12], [13], there are still several challenges for practical implementation of DRL in an ECC environment such as, the increasing of agents' exploration cost for getting enough trajectories of experience to capture the environment properties, and the significant process delay because of the DRL agents' training. Obviously, Centralized DRL agents are not suited to highly stochastic ECC environments due to their poor scalability and large overhead. Therefore, one critical challenge is adjusting distributed DRL techniques to work effectively in ECC.

In this paper, a DRL-based Distributed task Offloading (DRL-DO) framework is proposed to efficiently and effectively produce offloading decisions in ECC environments. DRL is used as a decision-maker for offloading the incoming dynamic workloads into the ESs or central CS. It makes offloading decision in a distributed manner to maintain availability, preserve generalizability and quickly adapt to new environments. Multiple Offload Generators (OG) and Mobile Devices (MD) are used, each OG knows the workload information of the MD and specific server directly connected to it by the last locally and offloaded executed tasks on that MD and server, respectively. The main contributions of this paper are summarized as follows:

1) The system utility of the ECC network is modeled as the weighted sum of task completion time and energy consumption for all MDs. The ECC task placement challenge is formalized as a multi-purpose optimization challenge. An efficient and effective offloading framework with innovative decision-making capabilities is proposed for non-divisible and delay aware tasks to jointly reduce the energy consumption ratio and long-term delay of offloaded tasks for each MD.
2) A DRL-DO algorithm is proposed to transfer the dynamic workloads of mobile applications by:
   - Generating a multi-class offload action for executing independent tasks locally, offloading them to the central CS, or one of the ESs.
   - The offloading decision is generated in a distributed manner by having multiple OGs knowing the system state information by their last offloaded tasks. Also, each OG can serve more than one MD.
   - Making decision of the task offloading and load balancing at one step.
   - Utilizing a three-dimensional input Convolutional Neural Network (CNN) while considering multiple tasks with multiple features from different users, and the previous workload in the MDs, ESs, and central CS as input to CNN to generate offload decision.
   - Achieving efficient task computation in terms of energy and delay.
   - Evaluating the convergence of proposed algorithms and the impacts of system parameters on the weight sum cost. Under various parameter conditions, DRL-DO achieves near-optimal offloading decisions in a fraction of a second.
3) Simulation experiments are performed in distinct situations to evaluate the effectiveness of DRL-DO. The proposed framework can effectively utilize the processing ability of ECC by achieving superior performance in terms of Gain Ratio (GR), energy consumption, response time, and system utility while achieving moderate time cost compared with several offloading algorithms.

The remainder of this paper is organized as follows: Section II presents the related works of DRL-based offloading approaches, Section III explains the system model and problem formulation for the proposed DRL-DO framework. An explanation of the framework of each OG is presented in Section IV. Evaluation methodology and Simulation results are shown in Section V. Finally, the conclusion is drawn out in Section VI.

II. RELATED WORK

MCC, MEC and ECC have become the most important solutions used to better serve IoT users' diverse requirements, the best task offloading decision to improve QoS while efficiently enhancing the terminal devices' computing power have drawn interest from the research community, leading to the recent several studies in the literature.

In recent years, traditional offloading approaches [6], [7], [8], [9] have failed to adapt policies to changing environments and cannot achieve long-term performance due to the highly dynamic and time-critical behavior of smart systems. Therefore, it is more advantageous to apply ML methods [14], [15] rather than classical methods. DRL holds great promise for resolving complex real-world problems, intelligent task offloading decisions for MEC, MCC and/or ECC, is becoming more and more dependent on deep learning-driven offloading systems. The decision-making offloading process can be made locally, considering the mobile users' perceptions condition, or globally, considering the state of the total system [2].

A DRL-based offloading algorithms for MEC networks are proposed in [16], [17], [18], [19], and [20]. In [16], [17], and [18], offloading decisions are generated through multiple parallel DNNs, while [19] employs a single CNN along with a quantization procedure for making the offloading decisions. Newly generated offloading decisions are stored in a shared replay memory, which is adjusted to further train and enhance all DNNs/CNN. However, these algorithms ignore the overall system state. It is proposed in [20] to combine the benefits of DRL and Lyapunov optimization to design an online computation offloading algorithm that maximizes network data processing capacity while considering average power and long-term data queue stability. Nevertheless, this algorithm only considers the workload of the MD and ignores the state of ESs and CS.

Several centralized offloading algorithms are proposed in [21], [22], and [23] to solve the above problems while considering the total system state. An online predictive offloading algorithm based on DRL and Long Short-Term Memory (LSTM) networks is proposed in [21], it predicts the load of the ES in real time during the model's training phase and allocates the computational resources for the task in advance to substantially increase the convergence speed and accuracy of the DRL algorithm during the offloading process. In [22], a DRL-based Task Offloading with cloud edge jointly Load Balance Optimization (TOLBO) algorithm is proposed to select the best ES or CS for offloading in order to minimize long-term task latency and energy consumption by jointly considering the requirements of latency and energy-sensitive tasks and the overall load dynamics in the cloud, edge, and end layers. An advanced DRL-based offloading algorithm is proposed in [23], considering the previous processing time for MDs, ESs, and CS, it can generate and store multi-class offloading decisions with the system state together in a database and then training and updating multiple parallel CNNs with a batch of labeled data for executing independent tasks. Despite the rapid convergence and global optimum benefits of centralized techniques, they are not scalable and unable to effectively address application placement issues due to their excessive computation, communication overhead, limited generalizability, and slow flexibility.

An increasing number of researches on distributed offloading techniques has been conducted to address these difficulties. In [24], [25], [26], [27], and [28], hybrid computation offloading architectures that adopt centralized training and distributed execution are proposed. A distributed DNN offloading algorithm (DDOA) is proposed in [24], multiple distributed DNNs working in parallel are used to create offloading decisions, these DNNs are enhanced further by using the newly generated decisions as a public training set and the back-propagation method with cross-entropy as the loss function. A Multi-Agent DRL algorithm is proposed in [25] to maximize energy consumption while simultaneously optimizing power control, resource allocation, and user equipment association, considering the cost of offloading and MEC server pricing. A multi-agent DRL algorithm is introduced in [26], it treats each user as a separate agent capable of deciding which tasks to offload instead of relying on a single centralized agent, each agent receives an instant reward for cooperating with other agents. In [27], a distributed application placement technique based on actor-critic is proposed, the process of developing experience trajectories is distinct from the comprehension of policy constraints. The distributed actors interact with their individual fog computing environments, each broker regularly gives the learner access to its local experience trajectories and in turn, the learner adjusts the target policy parameter and brokers update their local policy with the new information after every modification. A hybrid computation offloading architecture is proposed in [28], where all distributed actors are able to carry out tasks independently depending on their local observations and a centralized platform is introduced to collect global information. Multiple MDs can collaborate to make decisions following their independent local observations. A decentralized DRL-based scheduling algorithm is proposed in [29], it picks up knowledge about the environment during offloading, allowing it to adjust to changing conditions without knowing everything about it. Nevertheless, this algorithm requires user devices to monitor system status data to improve performance and achieve global optimization.

The key components of related work are identified and compared in Table 1 regarding their architectural characteristics and decision objectives. The identification and resolution

TABLE 1. Comparison of different DRL-based offloading decision algorithms.

Utilized Technique	Task Offloading	Task Number	Weight Parameters	Offloading Decision	Architectural Properties			Decision Objectives			
					MCC	MEC	ECC	Gain Ratio	Delay	Energy	Time Cost
DNN [16]	Centralized	Multiple	Constant	Locally	×	√	×	√	√	√	×
DNN [17]	Centralized	Multiple	Constant	Locally	×	×	√	√	√	√	×
DNN [18]	Centralized	Multiple	Constant	Locally	×	√	×	√	√	√	×
CNN [19]	Centralized	Multiple	Constant	Locally	×	×	√	×	√	√	×
DNN [20]	Centralized	Multiple	Constant	Locally	×	√	×	√	×	√	×
DNN [21]	Centralized	Single	Constant	Globally	×	√	×	×	√	√	×
CNN [22]	Centralized	Single	Constant	Globally	×	×	√	×	√	√	×
CNN [23]	Centralized	Multiple	Constant	Globally	×	×	√	√	√	×	×
DNN [24]	Distributed	Single	Constant	Locally	×	×	√	√	√	√	×
Muli-agent [25]	Hybrid	Single	Constant	Globally	×	√	×	×	√	√	√
Multi-agent [26]	Hybrid	Single	Adaptive	Globally	×	√	×	×	√	×	×
Actor-critic [27]	Hybrid	Single	Constant	Globally	×	×	√	×	√	√	×
Multi-agent [28]	Hybrid	Single	Constant	Globally	×	×	√	×	√	√	×
DNN [29]	Hybrid	Single	Constant	Globally	×	√	×	×	√	×	×
CNN [DRL-DO]	Distributed	Multiple	Adaptive	Globally	×	×	√	√	√	√	√

of significant task offloading challenges in heterogeneous ECC environments, where MEC and MCC collaborate to meet the demands of city IoT applications, has not received much attention in recent studies. Also, they optimize all system parameters simultaneously, which ultimately leads to the identification of impractical solutions as the optimal offloading choice. A DRL-DO framework is proposed in this paper to effeciently and effectively generate offloading decisions in ECC environments. DRL is used as a decision-maker to determine the optimal way to offload incoming dynamic workloads into central CS or ESs. Offloading decisions are made in a distributed manner to maintain the MDs' QoS while minimizing the weighted sum of task completion, delay, and energy consumption.

III. SYSTEM MODEL AND PROBLEM FORMULATION

The primary goal of the proposed DRL-DO framework is to generate a multi-class offload action. This action aims to efficiently manage the execution of independent tasks, allowing them to be processed locally on the MD or offloaded to one of the ESs or the central CS. This decision-making process considers the existing workload conditions of the MD, MEC servers, and the MCC infrastructure. The overarching objective is to maintain a harmonious balance in the mobile system, ensuring optimal task allocation and resource utilization across the various computing entities within the network. The framework employs DRL to dynamically adapt and optimize offloading decisions based on real-time workload conditions, contributing to the overall efficiency and performance of the mobile computing environment.

A. SYSTEM MODEL

Each MD sends the offload requests of some tasks to the OG directly connected to it. Then the OG makes offload decisions for more than one user and multiple tasks for each user at the same time. The OG can generate multiple outputs, where each output represents an offload decision for each task either locally, at one of the ESs or at central CS for further execution, the proposed DRL-DO system model is illustrated in Fig. 2. With the availability of more than one OG in the system, each OG serves multiple MDs and in turn makes offload decisions in a distributed manner, which preserves generalizability and quick adaptability to new environments.

The main challenge of DRL-DO is choosing the best offloading action, which minimize the total users' tasks delay and the corresponding energy consumption, and this can occur by minimizing the system utility of executing all the channel tasks at time t. The proposed DRL-DO utilizes DRL automata as a decision-maker to generate decisions about offloading the incoming dynamic workloads. Various notations of the proposed DRL-DO framework are shown in Table 2.

The proposed DRL-DO system model consists of a central CS, multiple ESs denoted by a set $k = \{1, 2, .., K\}$, multiple OGs denoted by a set $i = \{1, 2, .., I\}$, each OG is responsible for making offload action for multiple MDs denoted by a set $n = \{1, 2, .., N\}$, each MD requests offload of independent computational tasks denoted by $m = \{1, 2, .., M\}$, and each MD requests offload at multiple time periods denoted by a set $t = \{1, 2, \ldots, T\}$. The offload decision denoted by a numerical variable $a_{nm}(t) \in \{0, 1, .., K, K+1\}$,

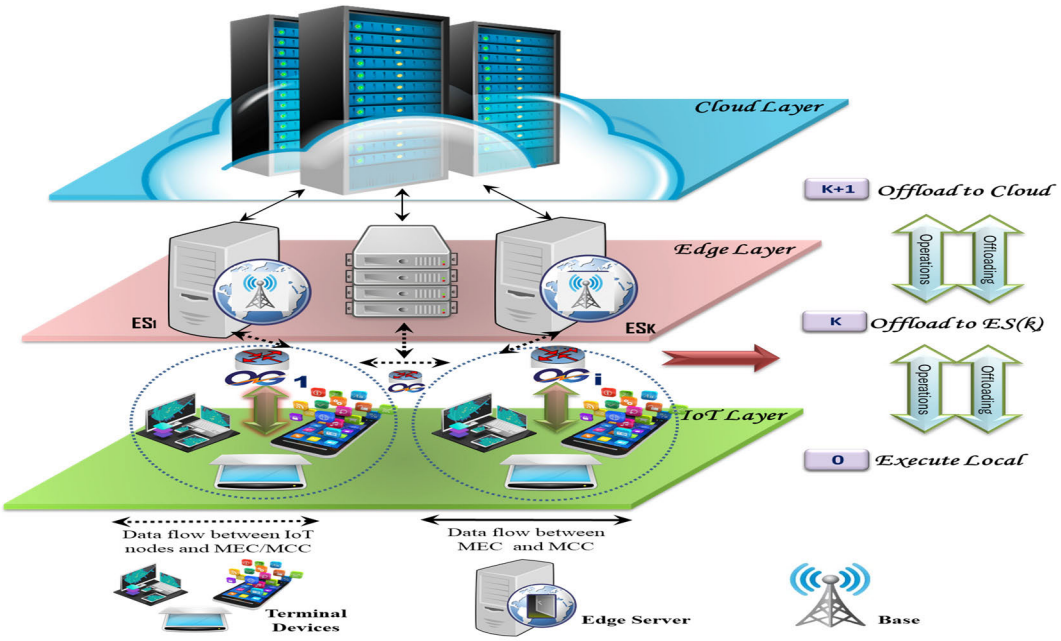


FIGURE 2. System model of the proposed DRL-DO algorithm.

which represents offloading decision for the m^{th} task of n^{th} MD in different cases, when the $a_{nm}(t)$ value equals 0, the decision is to execute the task locally, however the value of $a_{nm}(t) \in \{1, \ldots\ldots, K\}$ means that the offloading decision is to offload the task to ES(k), finally when $a_{nm}(t)$ equals $K+1$, the offloading decision is to offload the task to the central CS.

B. PROBLEM FORMULATION

The main goal of the proposed DRL-DO framework is to effectively produce offloading decisions over the ECC environments. The DRL is used as a decision maker to distribute offloading for the incoming dynamic workloads into MCC or ECC. Each OG is effectively aware of the workload information of the MDs and a particular server directly connected to it based on the most recent local and offloaded tasks completed on that MD and server.

IV. THE FRAMEWORK OF EACH OG

The Framework of each OG comprises four main phases, as shown in Fig. 3.

First, the data preparation phase involves reconstructing a 3D array from a 2D array that defines the properties of each channel task. It establishes the response time for each task, outlining the energy consumption across various offload scenarios for the distributed offloading of dynamic workloads to MCC or ECC. In the second phase, offloading actors are used to generate J candidate offloading actions, with one action per actor when the system state is provided. During the third phase, the optimal offloading action and the system state are selected from replay memory samples to train CNNs. Finally, in the last phase, the system workloads are updated depending on the best offloading action.

A. PHASE I: DATA PREPARATION

In this phase, numerous tasks with various features, as well as the system's current workload, are considered as input to CNN in order to produce offload decisions. Initially, OG makes an offload request of the system input channel $ch(t)$ which consists of m tasks of varied sizes from n MDs input requests, then a 3D array called $D_{in}(t)$ is created from $ch(t)$ at each time step t, as shown in Fig. 4. It defines the response time and energy consumption of each channel task at all offload possibilities for each corresponding server S_k, taking into account the previous processing time of MDs, ESs, and the central CS.

The response time of executing/offloading the m^{th} task of n^{th} MD on/to the corresponding assigned server (S_k) at time t ($RT_{nm}^{S_k}(t)$) is calculated by (1), it depends on the total CPU cycles ($Q_{nm}(t)$), the processing rate of S_k $\left(PR_n^{S_k}\right)$, the total offloading delay to S_k $\left(D_{nm}^{S_k}(t)\right)$, and the processing time of S_k at a previous time $\left(PT^{S_k}(t-1)\right)$.

$$RT_{nm}^{S_k}(t) = \frac{Q_{nm}(t)}{PR_n^{S_k}} + D_{nm}^{S_k}(t) + PT_n^{S_k}(t-1) \qquad (1)$$

where, $S_k \in \{S_0, S_1, \ldots, S_K, S_{K+1}\}$, S_0 denotes local MD, S_K denotes ES number, and S_{K+1} denotes the central CS. The ($Q_{nm}(t)$) required to execute the m^{th} task of the n^{th} MD at time t is defined by (2), it depends on the workload ($w_{nm}(t)$) and the positive coefficient of proportionality ($\delta_{nm}(t)$) [17].

$$Q_{nm}(t) = w_{nm}(t) \times \delta_{nm}(t) \qquad (2)$$

The total delay of offloading the m^{th} tasks of the n^{th} MD to the S_k ($D_{nm}^{S_k}(t)$) is calculated by (3), it depends on the

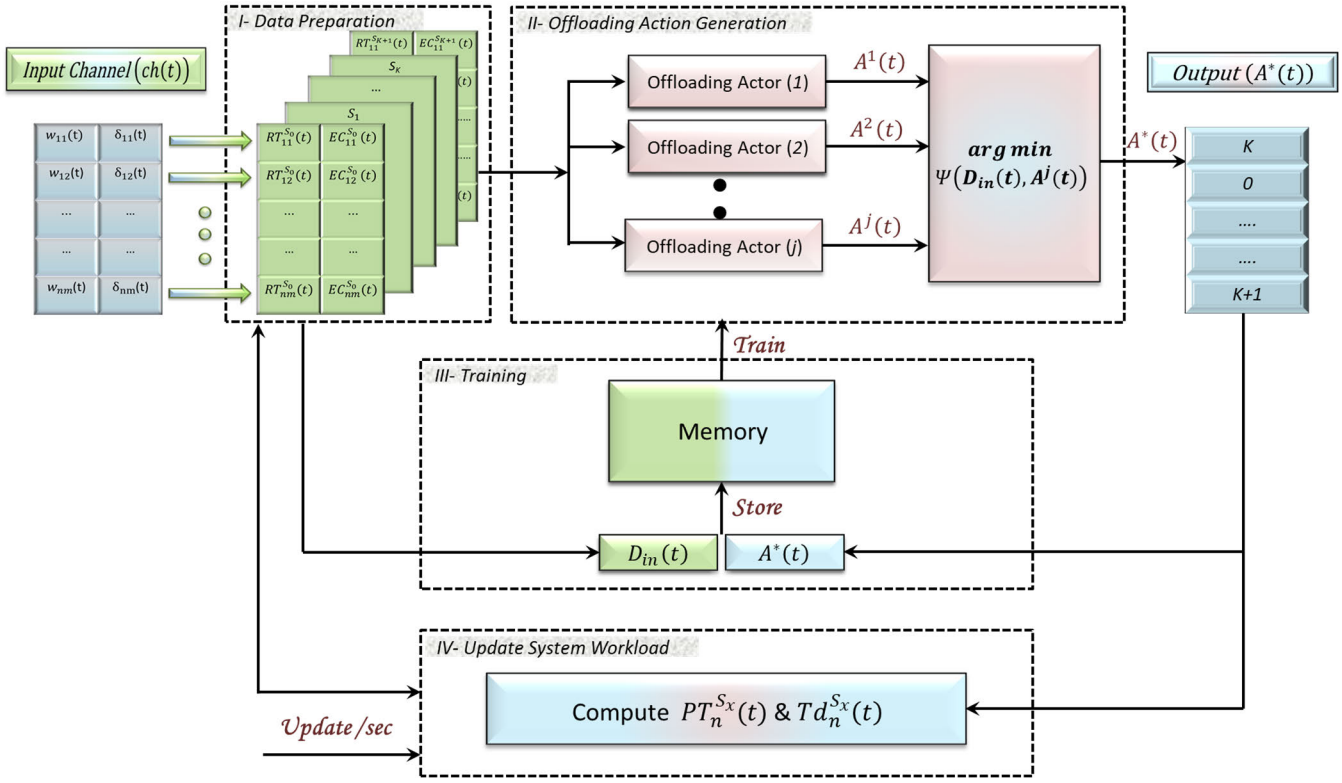


FIGURE 3. Procedure of the proposed DRL-DO framework.

offloading propagation delay of user n to S_k $\left(Pd_n^{S_k}\right)$, the transmission delay $(Td_{nm}(t))$, and the transmission delay of all offloaded tasks of the n^{th} MD at previous time t $(Td_n(t-1))$.

$$D_{nm}^{S_k}(t) = Pd_n^{S_k} + Td_{nm}(t) + Td_n(t-1) \tag{3}$$

where, the offloading transmission delay at time t is obtained by (4).

$$Td_{nm}(t) = \frac{w_{nm}(t)}{BN_n} \tag{4}$$

where BN_n is the bandwidth of the n^{th} user. Both $PT_n^{S_k}(t-1)$ and $Td_n(t-1)$ are calculated in the previous cycle.

The energy consumption parameter is now invested based on the task, MD, time, and execution/offloading decision. $EC_{nm}^{S_x}(t)$ represents the energy consumption by executing/offloading the m^{th} task of the n^{th} MD on/to the corresponding assigned server at time t, it is calculated by (5).

$$EC_{nm}^{S_k}(t) = \theta^T \times w_{nm}(t) + \theta^{S_k} \times Q_{nm}(t) \tag{5}$$

where θ^T represents the energy consumption per unit of uploading workload by each MD, its value is equal to 0, when S_k is equal to S_0. θ^{S_k} represents the corresponding assigned server's energy consumption per unit of workloads.

B. PHASE II: OFFLOADING ACTION GENERATION

At every time step t, the DRL-DO agent interacts with the environment, it perceives the current state of the environment $D_{in}(t)$ and selects an action $A(t)$ based on its policy, mapping state to action $A(t) = \{a_{11}(t), a_{12}(t), \ldots\ldots\ldots\ldots, a_{nm}(t)\}$, which represents the offloading action for all channel tasks at time t. The offloading decision for the m^{th} task of the n^{th} MD is represented by the numerical variable $a_{nm}(t) \in \{0, .., K, K+1\}$. The task is executed locally when the value of $a_{nm}(t) = 0$. On the other hand, if the value of $a_{nm}(t) \in \{1, .., K\}$, the task is offloaded to ES(k). Lastly, if the value of $a_{nm}(t) = \mathrm{K}+1$, the task is offloaded to the central CS.

As shown at phase II in Fig. 3, for each system state $D_{in}(t)$, J offloading actors are used to generate j candidate offloading actions with one action per actor, $j \in \{1, 2, \ldots.., J\}$. Inside each offloading actor, a three-dimensional input CNN which has multiple tasks with multiple features is used as input for generating multi-class offloading action represented by $A^j(\boldsymbol{t})$, which takes advantage of the translation in the variance of the CNN to capture the local feature of input data and accelerate the convergence of the offloading process, and it can be represented by a parametrized function $f_{\theta(j)}$, as in (6).

$$f_{\theta_{(j)}} : D_{in}(t) - -- \rightarrow A^j(\boldsymbol{t}) \tag{6}$$

where $\theta(j)$ denotes the parameters of the j^{th} offloading actor, all those actors have the same structure but with different parameter values. As seen in (7), the offloading action $A^*(\boldsymbol{t})$ with the least system utility $\left(\Psi^*(\mathrm{D}_{in}(t), A^j(\boldsymbol{t})\right)$ is selected among all J candidates to achieve the main DRL-DO goal by selecting the offloading action that achieves high

TABLE 2. Notations used in the DRL-DO algorithm.

Symbol	Definition
β	Constant weight cost of executing all the channel tasks.
β_t	Adaptive weight of response time.
β_e	Adaptive weight of energy consumption.
J	Number of offloading actors.
K	Number of ESs.
M	Number of tasks per MD.
N	Number of MDs connected directly to each OG.
T	Number of time cycles.
I	Number of OGs.
S_k	Corresponding server, $S_k \in \{S_0, S_1, ., S_K, S_{K+1}\}$.
θ^{S_k}	Energy consumption per unit of workload.
θ^T	Energy consumption by local MD for uploading workloads.
$D_{in}(t)$	3D array defines the response time and energy consumption for each channel task at all offload possibilities.
Bn_n	Bandwidth of the n^{th} MD.
$ch(t)$	Channel at time t, consists of more than one task per user.
$a_{nm}(t)$	Offloading decision for m^{th} task workload of n^{th} MD at t.
$A(t)$	Offloading action for all channel tasks at time t.
$PT^{S_k}(t)$	Processing time of corresponding assigned S_k at time t.
$Pd_n^{S_k}(t)$	Propagation delay when n^{th} user offloads task to S_k.
$RT_n^{S_k}(t)$	Response time of n^{th} MDs to finish its tasks offloading at t.
$RT_{nm}^{S_k}(t)$	Response time of m^{th} task of n^{th} MD executed by S_k at t.
$EC_n^{S_k}(t)$	Energy consumed by all tasks of the n^{th} MDs at time t.
$EC_{nm}^{S_k}(t)$	Energy consumption of executing/offloading task at t.
$w_{nm}(t)$	The m^{th} task workload of the n^{th} MD at time t.
$\delta_{nm}(t)$	Positive coefficient between workload and total CPU cycles.
$PR_n^{S_k}$	Processing rate of assigned S_k.
$Q_{nm}(t)$	Total CPU cycles of computing m^{th} task of n^{th} MD.
$TS_n(t)$	Total data size of all offloaded tasks of n^{th} MD at time t.
$D_{nm}^{S_k}(t)$	Total delay of offloading m^{th} task of n^{th} MD to S_k at time t.
$TL_n^{S_k}(t)$	Total length of all tasks processed by S_k concerning n^{th} MD.
$Td_n(t)$	Transmission delay of all offloaded tasks of n^{th} MD at t.
$Td_{nm}(t)$	Transmission delay of offloading m^{th} task of n^{th} MD at t.

performance regarding the total delay and the corresponding energy consumptions.

$$A^*(\boldsymbol{t}) = arg\ min\Psi\left(D_{in}(t), A^j(t)\right) \tag{7}$$

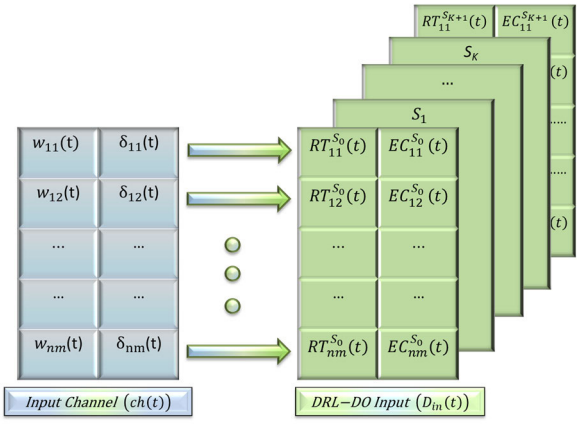


FIGURE 4. Construction of DRL-DO input from system input channel.

The $\Psi(D_{in}(t), A(t))$ is obtained by (8), it depends on the total response time ($RT(D_{in}(t), A(t))$) and the energy consumption ($EC(D_{in}(t), A(t))$) required to execute all channel tasks at time step t.

$$\Psi(D_{in}(t), A(t)) = ((\beta_t) * RT(D_{in}(t), A(t))) + ((\beta_e) * EC(D_{in}(t), A(t))) \tag{8}$$

As shown in (9) and (10), $\boldsymbol{\beta_t}$ and $\boldsymbol{\beta_e}$ are adaptive control parameters representing the weighted response time and energy consumption of all channel tasks, respectively.

$$\boldsymbol{\beta_t} = \frac{\beta * RT(D_{in}(t), A(t))}{(\beta * RT(D_{in}(t), A(t))) + ((1-\beta) * EC(D_{in}(t) A(t))} \tag{9}$$

$$\boldsymbol{\beta_e} = \frac{(1-\beta) * EC(D_{in}(t), A(t))}{(\beta * RT(D_{in}(t), A(t))) + ((1-\beta) * EC(D_{in}(t), A(t))} \tag{10}$$

where β is a constant weight between energy consumption and response time in the weighted execution cost of all the channel tasks.

The total response time and energy consumption are calculated by (11) and (13), they depend on the response time ($RT_n^{S_k}(t)$) and energy consumption $EC_n^{S_k}(t)$) of the n^{th} MD to finish executing/offloading its task on/to S_k at time t, respectively.

$$RT(D_{in}(t), A(t)) = \sum_{n=1}^{N} \max\left(Rt_n^{S_0}(t), RT_n^{S_1}(t), .., RT_n^{S_K}(t), RT_n^{S_{K+1}}(t)\right) \tag{11}$$

where,

$$RT_n^{S_k}(t) = \begin{cases} \sum_{m=1}^{M} RT_{nm}^{S_k}(t), & a_{nm}(t) = k \\ 0 & a_{nm}(t) \neq k \end{cases} \tag{12}$$

$$EC(D_{in}(t), A(t)) = \sum_{n=1}^{N} \sum_{k=0}^{K+1} ET_n^{S_k}(t), \tag{13}$$

where,

$$EC_n^{S_k}(t) = \begin{cases} \sum_{m=1}^{M} EC_{nm}^{S_k}(t), & a_{nm}(t) = k \\ 0 & a_{nm}(t) \neq k \end{cases} \tag{14}$$

The value of $a_{nm}(t)$ demonstrates whether the current task is assigned to S_k or not, and m denotes the number of independent computational tasks of the n^{th} MD.

C. PHASE III: TRAINING

All offloading actors are trained using the acquired labeled data at each training session. Once the highest performance offloading decision $A^*(\boldsymbol{t})$ is obtained, it is saved as a new entry of labeled data $(D_{in}(t), A^*(\boldsymbol{t}))$ in a finite-size memory structure, where the oldest data entry is discarded when the memory is full. The database improves efficiency since it

is updated frequently, resulting in newly created data that is more accurate than previously created data. However, it will take too much time to train each CNN using all of the labeled data from the database. Consequently, all CNNs have access to the same database, which they can use to randomly extract batches of data for neural network training.

In order to train the model in CNN using labeled data $(\mathrm{D}_{in}(t), A^*(\boldsymbol{t}))$, where $A^*(\boldsymbol{t}) = \left\{a^*_{11}(t), a^*_{12}(t), .., a^*_{nm}(t)\right\}$, the Adam optimizer [30] is performed to minimize the loss function [31] for each output in each offloading actor.

D. PHASE IV: UPDATE SYSTEM WORKLOAD

At every time step t, the optimal offloading decision $A^*(\boldsymbol{t})$ is obtained and the workload of the system is updated by recalculating both, the processing time of S_k $\left(PT_n^{S_k}(t)\right)$ and the transmission delay of all offloaded tasks $(Td_n(t))$ concerning the n^{th} MD, as shown in (15) and (17), respectively.

$$PT_n^{S_k}(t) = PT_n^{S_k}(t-1) + \frac{TL_n^{S_k}(t)}{PR_n^{S_k}} \tag{15}$$

where, $\left(PT^{S_k}(t-1)\right)$ is the previous processing time, $\left(TL_n^{S_k}(t)\right)$ is the total length of all tasks being processed, obtained by (16), and $\left(PR_n^{S_k}\right)$ is the processing rate of S_k.

$$TL_n^{S_k}(t) = \begin{cases} \sum_{m=1}^{M} Q_{nm}(t) & a_{nm}(t) = k,\ k = 0 \\ \sum_{n=1}^{N}\sum_{m=1}^{M} Q_{nm}(t) & a_{nm}(t) = k,\ k \neq 0 \\ 0 & a_{nm}(t) \neq k \end{cases} \tag{16}$$

$$Td_n(t) = Td_n(t-1) + \frac{TS_n(t)}{BN_n} \tag{17}$$

where, $(Td_n(t-1))$ is the transmission delay of all offloaded tasks of the n^{th} MD at the previous cycle, $\left(TS_n^{S_k}(t)\right)$ is the total data size of all offloaded tasks, calculated by (18), and BN_n is the Bandwidth of the n^{th} MD.

$$TS_n^{S_k}(t) = \begin{cases} \sum_{m=1}^{M} w_{nm}(t) & a_{nm}(t) \neq 0 \\ 0 & a_{nm}(t) = 0 \end{cases} \tag{18}$$

V. SIMULATION IMPLEMENTATION AND RESULTS

In the proposed DRL-DO algorithm, the database structure is continually updated and offloading decisions are generated as logits. The convergence here is the process of moving closer to the optimal. More precisely, GR, obtained by (19), refers to gain from the optimal weighted execution cost of executing all the channel tasks at time t $(\psi_2(\mathrm{D}_{in}(t), A(t)))$ to the minimum weighted execution cost of executing all the channel tasks at time t $(\psi_1(\mathrm{D}_{in}(t), A(t)))$ found by the DRL-DO algorithm.

$$GR = \frac{\psi_2(\mathrm{D}_{in}(t), A(t))}{\psi_1(\mathrm{D}_{in}(t), A(t))} \tag{19}$$

where, $0 < GR \leq 1$ indicates how close the solution found by the DRL-DO algorithm is to the greedy algorithm [21] which achieves the true optimal solution, nevertheless, it consumes a great deal of time, particularly when there are many MDs and tasks, as indicated in [17]. $\psi_2(\mathrm{D}_{in}(t), A(t))$ is determined by using a time-consuming greedy algorithm, in which all offloading decision combinations are enumerated, and then adopt the optimal one. Algorithm 1 shows the entire DRL-DO algorithm's progression for the ECC offloading model. The suggested DRL-DO framework automatically modifies the parameters to produce offloading decisions that are almost optimal by learning from previous offloading experiences. Because the generated offloading decisions are highly diverse, good convergence performance can be attained.

Algorithm 1 DRL-DO for One OG Over ECC Environment

Input: System input channel $ch(t)$ at time t
Output: Optimal offloading decision $A^*(t)$ at time t

Initialization:
  Initialize J offloading actors with random parameters
  Empty memory structure
for $t = 1, 2, \ldots, T$ **do**
  for each n user in a channel at time t **do**
    for each m task **do**
      Compute $RT_{nm}(t)$ and $EC_{nm}(t)$
    end for
  end for
  Compute system state $(D_{in}(t))$
  Replicate $(D_{in}(t))$ to all offloading actors
  Generate J offloading actions $A(t)$ from J actors
  Select the best offloading decision $A^*(t)$
  Compute gain ratio (convergence of DRL-DO)
  if database is not full **then**
    Store $\left(D_{in}(t), A^*(t)\right)$ into memory structure
  else
    Discard the oldest data and save new one
  end if
  if t % training interval $== 0$ **then**
    for $j = 1, 2, .., \mathrm{J}$ **do**
      Randomly sample batches from memory
      Train CNNs and update θ_j
    end for
  end if
end for

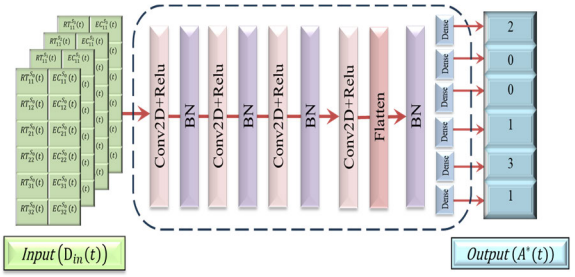


FIGURE 5. Configuration of an offloading actor.

TABLE 3. Simulation parameters.

Parameter	Value
β	0.5
J	5
K	2
M	2
N	3
T	15000
I	2, 3
S_k	$\in \{S_0, S_1, S_2, S_3\}$
θ^{S_k}	3.5×10^{-7}, 1.5×10^{-7}, or 1×10^{-7} j/bit depend on k
θ^{T}	1.42×10^{-7} j/bit
$D_{in}(t)$	3D array of shapes (6,2,4)
Bn_n	50 Mbps
$ch(t)$	An array of shapes (6,2)
$a_{nm}(t)$	$\in \{0, 1, 2, 3\}$
$Pd_n^{S_k}(t)$	2, 6, or 50 MS (depend on the value of S_k)
$w_{nm}(t)$	Randomly distributed between 1MB and 3MB
$PR_n^{S_k}$	5×10^8, 20×10^8, or 100×10^8 cycle/s depend on S_k

TABLE 4. Hyperparameters setting of Conv2D.

Conv2D	Number of Filters	Kernel Size
Layer 1	32	3×3
Layer 2	64	3×3
Layer 3	128	1×1
Layer 4	256	1×1

TABLE 5. Hyperparameters setting of DRL-DO.

Factor	Time frames	Training interval	Batch size	Memory size	Learning rate
Value	15000	10	192	1024	0.01

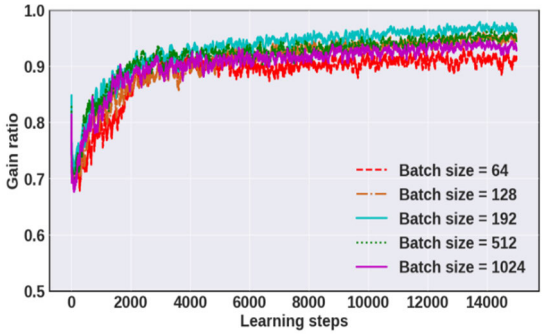


FIGURE 6. Relative optimality GR under varied batch sizes.

A. PARAMETER AND ENVIRONMENT SETTING

In this work, a heterogeneous ECC environment is constructed and the Keras ML library [32], [33], a high-level Application Programming Interface (API) for TensorFlow, is used to implement and evaluate the proposed DRL-DO algorithm and other offloading decision algorithms in Python experiments.

The number of time cycles is set to 15000 (T=15000), the number of ESs is assigned to 2 (k=2), the number of OGs is assigned to 2, or 3 (I$\in$ {2, 3}). The number of MDs directly connected to each OG is assigned to 3 (N=3), where each OG takes offloading action for 3 MDs, and each MD has two different tasks (M = 2). In addition, the processing rates for the n^{th} MD, ES, central CS are, $PR_{md}(n) = 500$ MHz, $\mathrm{PR_E} = 2$ GHz and $\mathrm{PR}_C = 10$ GHz, respectively, where $PR_{md}\ (n) < PR_E < PR_C$. The request workload of all tasks is randomly distributed between 1MB and 3MB. The offloading decision $a_{nm}(t) \in \{0, 1, 2, 3\}$ where, $a_{nm}(t) = 3$, $a_{nm}(t) = \{1,2\}$, and $a_{nm}(t) = 0$ represents that the n^{th} MD is offloaded its m^{th} task to the CS, ES1/2, or locally, respectively. The summary of the DRL-DO evaluation parameters values is presented in Table 3.

The DRL-DO offloading actor model makes use of an input Convolution (Conv) layer, three hidden Conv layers, and one multi-label fully connected layer, as illustrated in Fig. 5. The Conv2D [34] layer is the most common layer used to perform convolution operation on the DRL-DO input (D_{in}(t)) to extract features, which are used by dense layer to produce output ($A\ (t)$). Except of the output layer, each layer is followed by the Rectified Linear Unit (ReLU) activation function [35] and the Batch Normalization (BN) layer [36]. The Softmax activation function classifies the output into multiple classes [37]. The final Conv2D layer's output feature maps are usually flattened, or converted into a one-dimensional array of numbers. Then, they are linked to a multi-label fully connected layer, also known as a dense layer, where each input is connected to every output through a learnable weight. Once the features extracted by the Conv2D layers are generated, a fully connected layer maps them to the ultimate outputs of the network.

The first parameter Conv2D needs to know is how many learning filters it needs. The layers in the early network architecture closest to the real data input need to learn fewer Conv2D filters than the more profound layers closer to the output predictions. The second is the kernel size, an odd 2-tuple integer that defines the Conv2D window's width and height. Table 4 displays the hyperparameter settings used to tune the DRL-DO algorithm on one OG. J-offloading actors are first initialized with random values for the parameters θ_j. The algorithm converges to the best offloading actions by selecting $J \geq 2$.

B. CONVERGENCE PERFORMANCE

The convergence of the DRL-DO algorithm is demonstrated in distinct scenarios, where it converges to the optimal solution under a broad range of parameter settings. Based on the batch size, memory size, and number of CNNs, the convergence performance of the DRL-DO algorithm is analyzed.

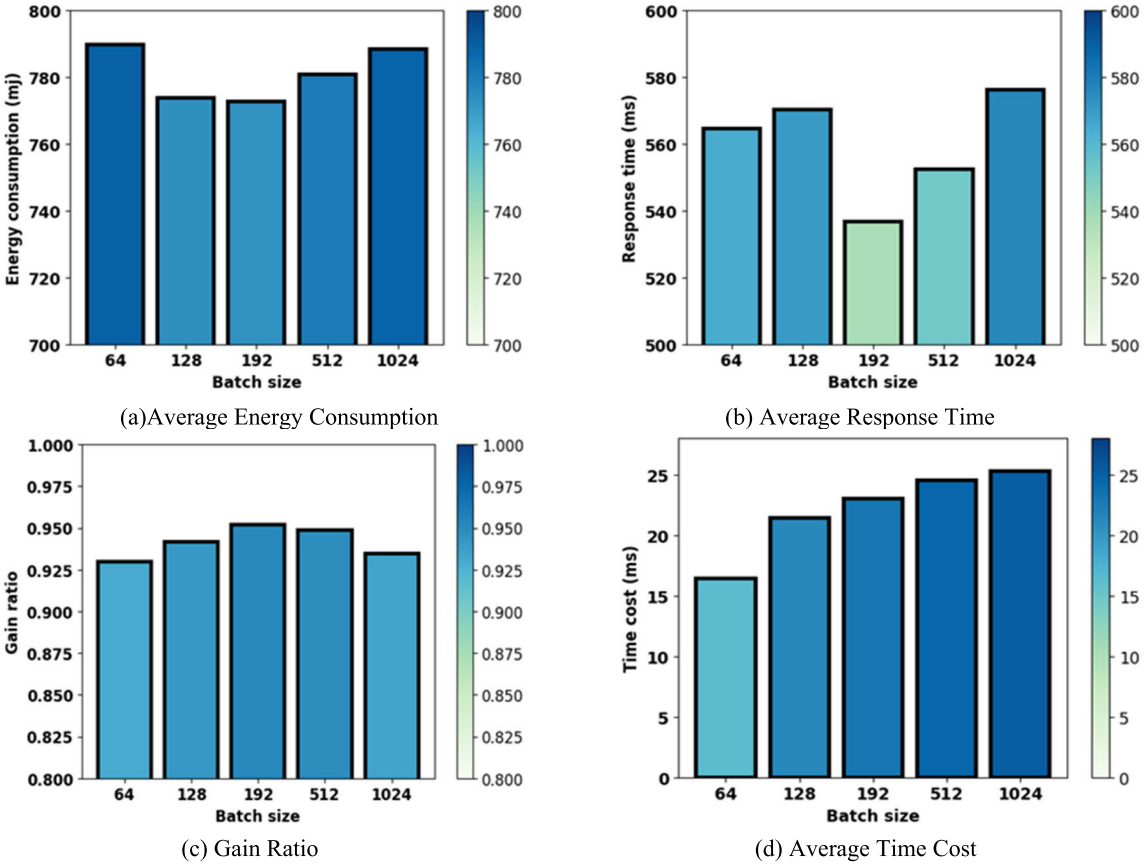


FIGURE 7. Performance analysis under different batch sizes.

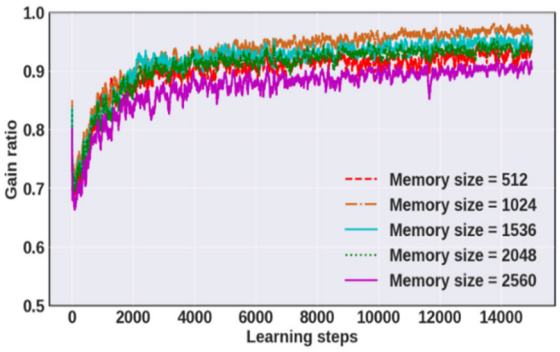


FIGURE 8. Relative optimality GR under varied memory sizes.

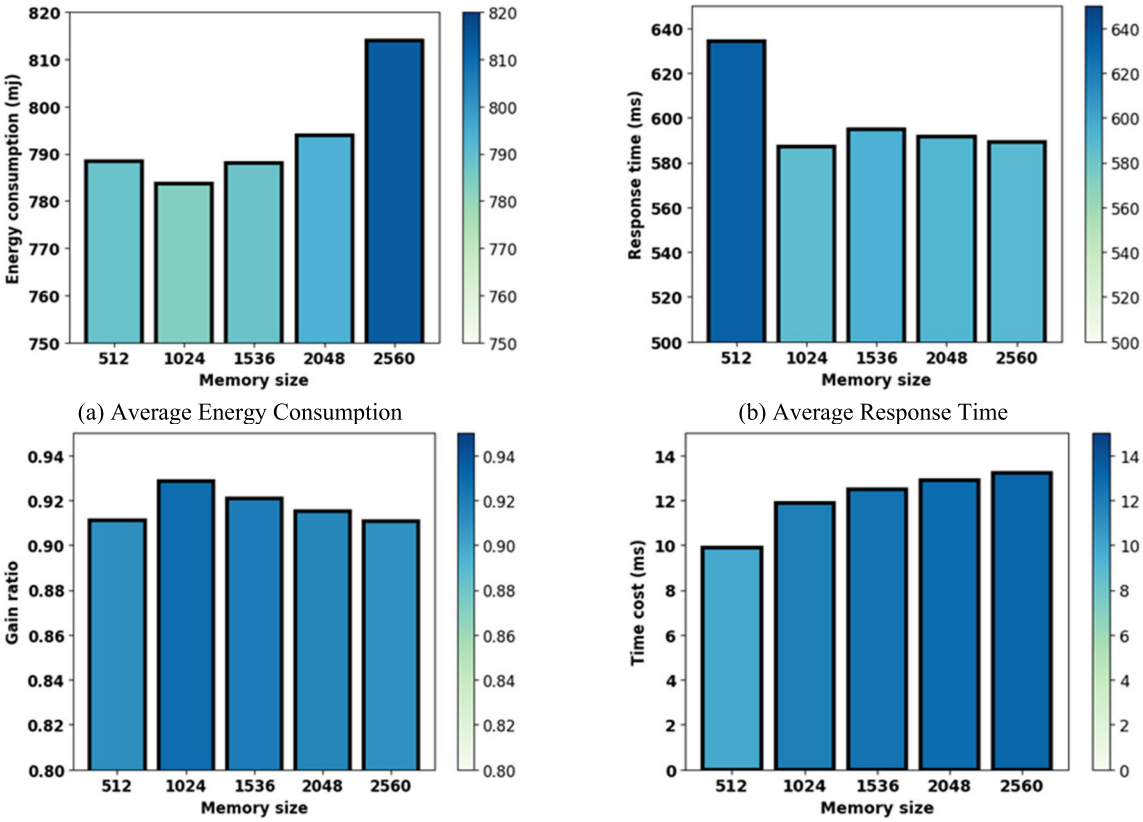
This proposal considers both a long-term optimization problem and a dynamic, concurrently executing environment [38]. Every task is executed simultaneously on the server, and the environment's network, location, and execution speed are all randomly changed. The adaptive learning rate optimization algorithm known as the Adam Optimizer [30] is used to dynamically modify the learning rate value in response to training progress and in turn, enable the model to learn more efficiently. The settings used for hyperparameters are shown in table 5.

1) IMPACT OF BATCH SIZE

The number of experience samples trained each time is determined by batch. To enhance the CNN, a batch of data samples is randomly sampled from the memory for each training procedure. Fig. 6 demonstrates the impact of batch size on the convergence GR of the proposed DRL-DO algorithm.

Fig. 7 shows the obtained results of DRL-DO algorithm in terms of the average Energy consumption, Response time, GR, and Time cost, respectively for each OG under different numbers of samples for training, 6000 iterations and 100 samples are used for the test.

Small batch size does not use all of the training data stored in memory, as illustrated in Fig. 6. Conversely, a large batch size frequently uses the old training data, slowing down convergence and lengthening the training period, as illustrated in Fig. 7. As the batch size increases, energy consumption and response time decrease, increasing the GR and causing the model to converge to a satisfactory solution; however, the average time cost increases by a small amount, which represents the computation time required by the DRL-DO algorithm to perform the offloading action. Therefore, in the following simulations, the training batch size is set to 192 as a trade-off between convergence speed and time cost.



(a) Average Energy Consumption

(b) Average Response Time

(c) Gain Ratio

(d) Average Time Cost

FIGURE 9. Performance analysis under different memory sizes.

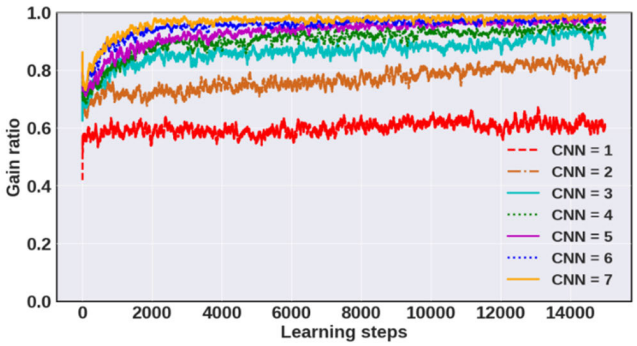


FIGURE 10. Relative optimality GR under varied CNNs.

optimum. The data is updated at a low rate when the database size is too large. Thus, the value of GR is at a minimum when the memory size is 2560.

As shown in Fig. 9, the case with a small memory size performs better than a large memory size regarding energy consumption, response time, and in turn, the DRL-DO algorithm utilizes these adaptive weights to minimize the weighted execution cost of executing all channel tasks and improves GR.

However, the increase of the memory size leads to the increase of the time cost. From the experimental observation, the 1024 memory size obtains the best results and therefore, an experience replay memory with size 1024 is selected in the ECC network.

2) IMPACT OF MEMORY SIZE

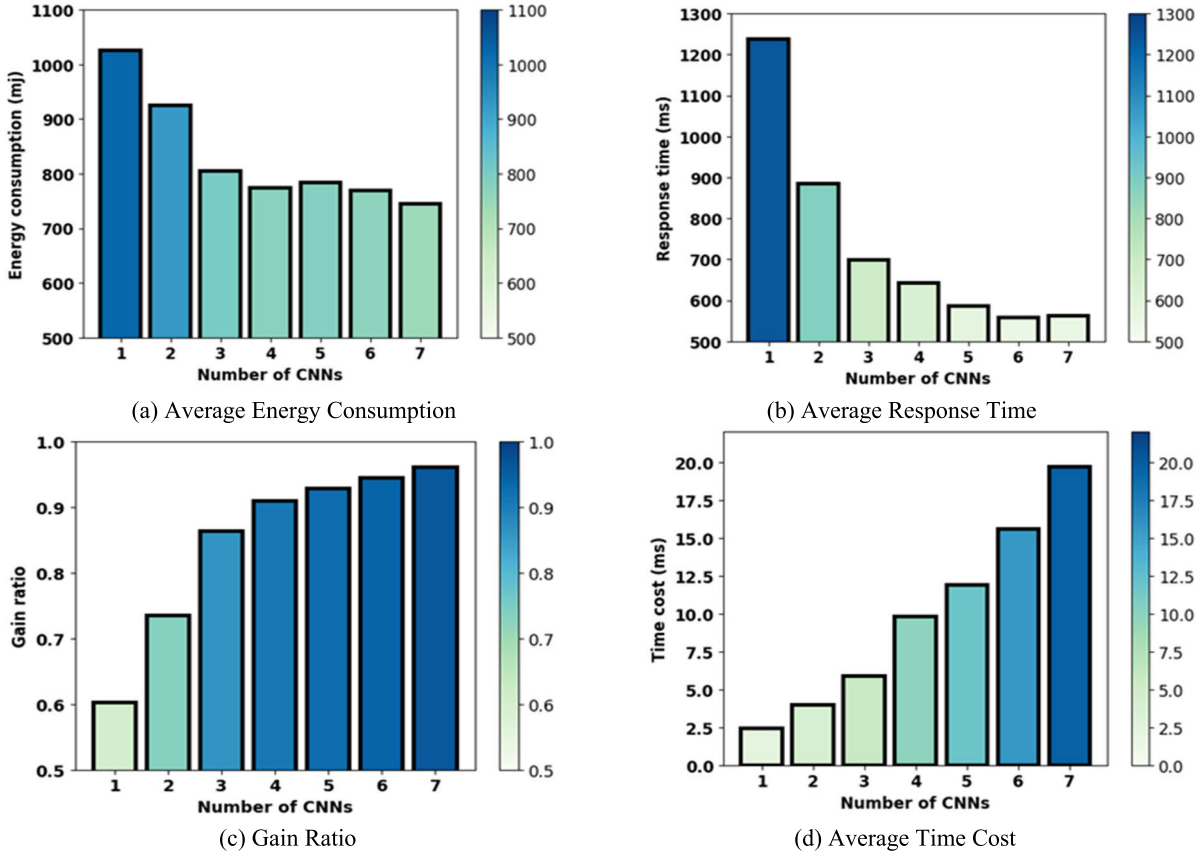
Another important factor affecting DRL-DO performance is memory size, it directly affects how many training samples are kept in replay memory. Fig. 8 shows how different memory sizes affect the GR under various learning steps. The energy consumption, response time, GR, and time cost are chosen, as indicated in Fig. 9, to assess how various memory sizes affect each OG in the DRL-DO algorithm, 3000 iterations and 100 samples are used for test.

As illustrated in Fig. 8, faster convergence is achieved with smaller memory sizes, but they fall on the risk of local

3) IMPACT OF CNN'S NUMBER

This section examines DRL-DO's convergence performance with varying numbers of offloading actors (CNNs). As the number of CNNs increases, the GR converges to one, as shown in Fig. 10. However, if only one CNN is used, the DRL-DO cannot converge and cannot learn anything from the data it generates. As a result, DRL-DO converges faster with more CNNs used and needs a minimum of two CNNs.

Fig. 11 shows the DRL-DO algorithm's results regarding the average energy consumption, response time, GR, and the



(a) Average Energy Consumption

(b) Average Response Time

(c) Gain Ratio

(d) Average Time Cost

FIGURE 11. Performance analysis under different CNNs.

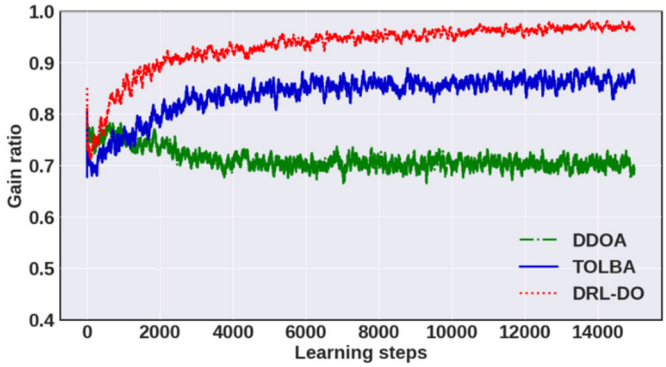


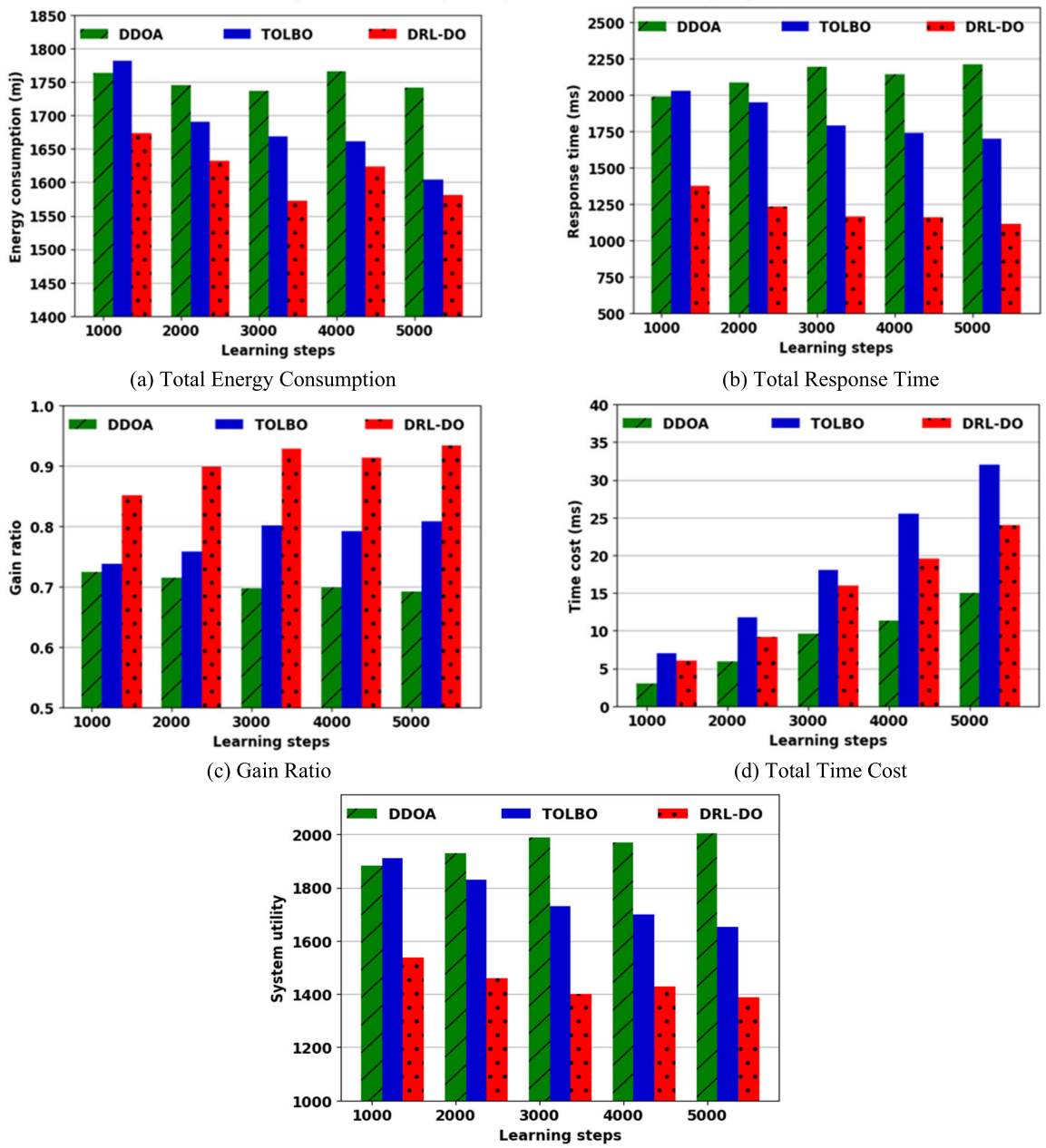
FIGURE 12. Relative optimality GR under varied learning steps.

time cost of each OG under various offloading actors at a 3000-time step number with 100 tested samples. As illustrated in Fig. 11, as the number of CNNs increases, the average energy consumption and response time decrease, leading to an increase in the GR, indicating that the model converges to a good solution. However, the number of CNNs also increases the average time cost by a significant amount; for this reason, the number of CNNs is set at five to balance convergence and time cost in the ECC network under consideration.

C. COMPARISON ANALYSIS

In comparison with DRL-DO that uses two OGs, each serving three MDs, the following offloading decision techniques are used in order to evaluate the efficacy and performance of the proposed DRL-DO algorithm in terms of total energy consumption, response time, GR, time cost, and system utility in various policy update iterations:

- TOLBO [22]: A centralized offloading algorithm, which uses a single center OG responsible for making offloading decisions simultaneously for 6 MDs.



(a) Total Energy Consumption

(b) Total Response Time

(c) Gain Ratio

(d) Total Time Cost

(e) System Utility

FIGURE 13. The comparisons of the three algorithms regarding the learning steps.

- DDOA [24]: A distributed DNN offloading algorithm, uses two OGs, each one simultaneously making an offloading decision for 3 MDs.

The offloading actor for different algorithms is implemented using the Conv2D class, batch size, memory size, learning rate, and CNN's numbers are set to 192, 1024, 0.01 learning rate, and 5, respectively.

Fig. 12 illustrates that when offload decisions are generated using DRL-DO, the GR reaches its maximum comparing to other algorithms. Also, as shown in Fig. 13, the DRL-DO model converges to the best result in terms of energy consumption, response time, GR and system utility, while achieving moderate time cost. This is because the proposed DRL-DO makes offloading decisions in a distributed manner, gathering data about the current load of the entire system by the last task executed locally or offloaded. As shown in Fig. 13, the average time cost of TOLBO is maximum because it requires global state observation before calculating optimal offloading decisions, and it has a single center OG that generates offload decisions, requiring a time delay from each MD to this center OG and results in excessive computational and communication overhead. DDOA achieves the

minimum time cost. However, all other performance metrics are degraded because it ignores the entire system's state.

VI. CONCLUSION AND FUTURE WORK

This paper proposes a DRL Distributed task Offloading (DRL-DO) framework based on DRL to generate offloading decisions in ECC environments efficiently and effectively. The primary goal of the proposed DRL-DO framework is to create a multi-class offload action that can be used to execute independent tasks locally and offload them to the central CS, or ESs, while taking into account the workload that the MD, MEC, and MCC are currently handling to maintain availability, preserve generalizability, and enable quick adaptation to new environments.

According to experimental results, the DRL-DO algorithm is accurate; in comparison to DDOA and TOLBO offloading algorithms, it achieves high performance, significantly lowers energy consumption by about 7.6%, and 3.8%, by about 43%, and 34% for response time, by about 26.2%, and 18.2% for system utility, and increases the GR by about 28.4 %, and 16.2% in contrast of DDOA and TOLBO, respectively, while achieving acceptable time cost. Thus, the proposed DRL-DO framework is well-positioned to contribute valuable insights to the field of real-time task offloading decisions in ECC environments and its applications in IoT.

More variables such as the bandwidth allocated to each task, learning on a smaller-shaped dataset, and constraints on energy consumption at the local MD level will be considered in future work to enhance the DRL-DO algorithm's ability and handle more realistic mobile offloading scenarios.

REFERENCES

[1] 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.

[2] B. Wang, C. Wang, W. Huang, Y. Song, and X. Qin, "A survey and taxonomy on task offloading for edge-cloud computing," *IEEE Access*, vol. 8, pp. 186080–186101, 2020.

[3] M. Y. Akhlaqi and Z. B. Mohd Hanapi, "Task offloading paradigm in mobile edge computing-current issues, adopted approaches, and future directions," *J. Netw. Comput. Appl.*, vol. 212, Mar. 2023, Art. no. 103568.

[4] R. F. Abdel-Kader, N. E. El-Sayad, and R. Y. Rizk, "Efficient energy and completion time for dependent task computation offloading algorithm in industry 4.0," *PLoS ONE*, vol. 16, no. 6, Jun. 2021, Art. no. e0252756.

[5] H. Nashaat, E. Ahmed, and R. Rizk, "IoT application placement algorithm based on multi-dimensional QoE prioritization model in fog computing environment," *IEEE Access*, vol. 8, pp. 111253–111264, 2020.

[6] M. Goudarzi, H. Wu, M. Palaniswami, and R. Buyya, "An application placement technique for concurrent IoT applications in edge and fog computing environments," *IEEE Trans. Mobile Comput.*, vol. 20, no. 4, pp. 1298–1311, Apr. 2021.

[7] M. Goudarzi, M. Palaniswami, and R. Buyya, "A distributed application placement and migration management techniques for edge and fog computing environments," in *Proc. 16th Conf. Comput. Sci. Intell. Syst.*, Sep. 2021, pp. 37–56.

[8] Q. Deng, M. Goudarzi, and R. Buyya, "FogBus2: A lightweight and distributed container-based framework for integration of IoT-enabled systems with edge and cloud computing," in *Proc. Int. Workshop Big Data Emergent Distrib. Environments*, Jun. 2021, pp. 1–8.

[9] M. Aazam, S. U. Islam, S. T. Lone, and A. Abbas, "Cloud of Things (CoT): Cloud-fog-IoT task offloading for sustainable Internet of Things," *IEEE Trans. Sustain. Comput.*, vol. 7, no. 1, pp. 87–98, Jan. 2022.

[10] I. H. Sarker, "Machine learning: Algorithms, real-world applications and research directions," *Social Netw. Comput. Sci.*, vol. 2, no. 3, pp. 1–21, May 2021.

[11] B. Kar, W. Yahya, Y.-D. Lin, and A. Ali, "A survey on offloading in federated cloud-edge-fog systems with traditional optimization and machine learning," 2022, *arXiv:2202.10628*.

[12] A. Heuillet, F. Couthouis, and N. Díaz-Rodríguez, "Explainability in deep reinforcement learning," *Knowl.-Based Syst.*, vol. 214, Feb. 2021, Art. no. 106685.

[13] 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.

[14] X. Chen, S. Hu, C. Yu, Z. Chen, and G. Min, "Real-time offloading for dependent and parallel tasks in cloud-edge environments using deep reinforcement learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 35, no. 3, pp. 391–404, Mar. 2024.

[15] T. H. Binh, D. B. Son, H. Vo, B. M. Nguyen, and H. T. T. Binh, "Reinforcement learning for optimizing delay-sensitive task offloading in vehicular edge-cloud computing," *IEEE Internet Things J.*, vol. 11, no. 2, pp. 2058–2069, Jul. 2023.

[16] L. Huang, X. Feng, A. Feng, Y. Huang, and L. P. Qian, "Distributed deep learning-based offloading for mobile edge computing networks," *Mobile Netw. Appl.*, vol. 27, no. 3, pp. 1123–1130, Nov. 2018.

[17] H. Wu, Z. Zhang, C. Guan, K. Wolter, and M. Xu, "Collaborate edge and cloud computing with distributed deep learning for smart city Internet of Things," *IEEE Internet Things J.*, vol. 7, no. 9, pp. 8099–8110, Sep. 2020.

[18] L. Huang, X. Feng, L. Zhang, L. Qian, and Y. Wu, "Multi-server multi-user multi-task computation offloading for mobile edge computing networks," *Sensors*, vol. 19, no. 6, p. 1446, Mar. 2019.

[19] Y. Wang, M. Li, R. Ji, M. Wang, Y. Zhang, and L. Zheng, "A convolutional operation-based online computation offloading approach in wireless powered multi-access edge computing networks," *Comput. Electron. Agricult.*, vol. 197, Jun. 2022, Art. no. 106967.

[20] S. Bi, L. Huang, H. Wang, and Y. A. Zhang, "Lyapunov-guided deep reinforcement learning for stable online computation offloading in mobile-edge computing networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 11, pp. 7519–7537, Nov. 2021.

[21] Y. Tu, H. Chen, L. Yan, and X. Zhou, "Task offloading based on LSTM prediction and deep reinforcement learning for efficient edge computing in IoT," *Future Internet*, vol. 14, no. 2, p. 30, Jan. 2022.

[22] L. Yan, H. Chen, Y. Tu, and X. Zhou, "A task offloading algorithm with cloud edge jointly load balance optimization based on deep reinforcement learning for unmanned surface vehicles," *IEEE Access*, vol. 10, pp. 16566–16576, 2022.

[23] W. Hashem, R. Attia, H. Nashaat, and R. Rizk, "Advanced deep reinforcement learning protocol to improve task offloading for edge and cloud computing," in *Proc. Int. Conf. Adv. Mach. Learn. Techn. Appl.* Springer, 2022, pp. 615–628.

[24] Z. Yang and W. Bai, "Distributed computation offloading in mobile fog computing: A deep neural network approach," *IEEE Commun. Lett.*, vol. 26, no. 3, pp. 696–700, Mar. 2022.

[25] J. Xue, Q. Wu, and H. Zhang, "Cost optimization of UAV-MEC network calculation offloading: A multi-agent reinforcement learning method," *Ad Hoc Netw.*, vol. 136, Nov. 2022, Art. no. 102981.

[26] Z. Zhang, C. Li, S. Peng, and X. Pei, "A new task offloading computing in edge computing," *EURASIP J. Wireless Commun. Netw.*, vol. 2021, no. 1, pp. 1–21, Jan. 2021.

[27] M. Goudarzi, M. Palaniswami, and R. Buyya, "A distributed deep reinforcement learning technique for application placement in edge and fog computing environments," *IEEE Trans. Mobile Comput.*, vol. 22, no. 5, pp. 2491–2505, May 2023.

[28] Z. Qin, H. Yao, T. Mai, D. Wu, N. Zhang, and S. Guo, "Multi-agent reinforcement learning aided computation offloading in aerial computing for the Internet-of-Things," *IEEE Trans. Services Comput.*, vol. 16, no. 3, pp. 1976–1986, Jul. 2022.

[29] Y. Fan, J. Ge, S. Zhang, J. Wu, and B. Luo, "Decentralized scheduling for concurrent tasks in mobile edge computing via deep reinforcement learning," *IEEE Trans. Mobile Comput.*, pp. 1–15, Apr. 2023, doi: 10.1109/TMC.2023.3266226.

[30] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[31] Y. Ho and S. Wookey, "The real-world-weight cross-entropy loss function: Modeling the costs of mislabeling," *IEEE Access*, vol. 8, pp. 4806–4813, 2020.
[32] M. Abadi et al., "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," 2016, *arXiv:1603.04467*.
[33] T. Carneiro, R. V. M. D. Nobrega, T. Nepomuceno, G.-B. Bian, V. H. C. De Albuquerque, and P. P. R. Filho, "Performance analysis of Google colaboratory as a tool for accelerating deep learning applications," *IEEE Access*, vol. 6, pp. 61677–61685, 2018.
[34] C. Shi, R. Xia, and L. Wang, "A novel multi-branch channel expansion network for garbage image classification," *IEEE Access*, vol. 8, pp. 154436–154452, 2020.
[35] B. Ding, H. Qian, and J. Zhou, "Activation functions and their characteristics in deep neural networks," in *Proc. Chin. Control Decis. Conf. (CCDC)*, 2018, pp. 1836–1841.
[36] T. Sledevic, "Adaptation of convolution and batch normalization layer for CNN implementation on FPGA," in *Proc. Open Conf. Electr., Electron. Inf. Sci.*, Apr. 2019, pp. 1–4.
[37] M. A. Anjum, J. Amin, M. Sharif, H. U. Khan, M. S. A. Malik, and S. Kadry, "Deep semantic segmentation and multi-class skin lesion classification based on convolutional neural network," *IEEE Access*, vol. 8, pp. 129668–129678, 2020.
[38] R. Rizk and H. Nashaat, "Smart prediction for seamless mobility in F-HMIPV6 based on location based services," *China Commun.*, vol. 15, no. 4, pp. 192–209, Apr. 2018.

HEBA NASHAAT (Member, IEEE) received the B.Sc. and M.Sc. degrees in computer and control engineering from Suez Canal University, in 2001 and 2006, respectively, and the Ph.D. degree in computer and control engineering from Port Said University, Egypt, in 2011. She is currently an Assistant Professor of computer and control with the Electrical Engineering Department, Port Said University. She is also the former Executive Director of the Network Infrastructure Center, Port Said University. Recently, she has been the Executive Director of the Software Engineering Unit (SWEU), Faculty of Engineering, Port Said University, and the Manager of the Electronic Learning Center (ELC), Port Said University. Her research interests include computer networking, including mobile networks, cloud computing, and the Internet of Things.

WALAA HASHEM received the B.Sc. degree in computer and control engineering from Suez Canal University, in 2006, and the M.Sc. degree in computer and control engineering from Port Said University, in 2018. Her research interests include computer networking, cloud computing, and machine learning.

RAWYA RIZK (Senior Member, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees in computers and control engineering from Suez Canal University, in 1991, 1996, and 2001, respectively. She has been the Chief Information Officer (CIO) with Port Said University (PSU), Egypt, since 2014. She has been the Vice President for Graduate Studies and Research with PSU, since 2021. She is currently a Professor of computers and control with the Electrical Engineering Department, PSU. Her research interests include computer networking, including mobile networking, wireless, ATM, sensor networks, ad hoc networks, QoS, traffic and congestion control, handoffs, and cloud computing. She is a Reviewer of many international communication and computer journals, such as IEEE Intelligent Transportation Systems Transactions, IEEE Access, and *IET Communications*.

RADWA ATTIA received the B.Sc. and M.Sc. degrees in computer and control engineering from Suez Canal University, in 2004 and 2009, respectively, and the Ph.D. degree in computer and control engineering from Port Said University, Egypt, in 2015. She is currently an Assistant Professor of computer and control with the Electrical Engineering Department, Port Said University. Her current research interests include computer networking, including mobile networking, wireless, sensor networks, and ad hoc networks. She is a Reviewer of many international communication and computer journals, such as *KSII Transactions on Internet and Information Systems*, *Wireless Personal Communications*, and *ANTE Annals of Telecommunications*.