IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON CLOUD - FOG - EDGE COMPUTING IN CYBER-PHYSICAL-SOCIAL SYSTEMS (CPSS)

Received February 25, 2020, accepted March 6, 2020, date of publication March 17, 2020, date of current version March 26, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2981434

Task Offloading and Resource Allocation for Mobile Edge Computing by Deep Reinforcement Learning Based on SARSA

TAHA ALFAKIH[1], MOHAMMAD MEHEDI HASSAN[1,2], (Senior Member, IEEE), ABDU GUMAEI[1], CLAUDIO SAVAGLIO[3], AND GIANCARLO FORTINO[3], (Senior Member, IEEE)

[1]Department of Information Systems, College of Computer and Information Sciences, King Saud University, Riyadh 11543, Saudi Arabia
[2]Research Chair of Smart Technologies, King Saud University, Riyadh 11543, Saudi Arabia
[3]Department of Informatics, Modeling, Electronics and Systems, University of Calabria, 87036 Rende, Italy

Corresponding author: Mohammad Mehedi Hassan (mmhassan@ksu.edu.sa)

This work was supported by the Deanship of Scientific Research at King Saud University through the Vice Deanship of Scientific Research Chairs. Chair of Smart Technologies. The authors also thank the RSSU at King Saud University for their technical support.

ABSTRACT In recent years, computation offloading has become an effective way to overcome the constraints of mobile devices (MDs) by offloading delay-sensitive and computation-intensive mobile application tasks to remote cloud-based data centers. Smart cities can benefit from offloading to edge points in the framework of the so-called cyber–physical–social systems (CPSS), as for example in traffic violation tracking cameras. We assume that there are mobile edge computing networks (MECNs) in more than one region, and they consist of multiple access points, multi-edge servers, and N MDs, where each MD has M independent real-time massive tasks. The MDs can connect to a MECN through the access points or the mobile network. Each task be can processed locally by the MD itself or remotely. There are three offloading options: nearest edge server, adjacent edge server, and remote cloud. We propose a reinforcement-learning-based state-action-reward-state-action (RL-SARSA) algorithm to resolve the resource management problem in the edge server, and make the optimal offloading decision for minimizing system cost, including energy consumption and computing time delay. We call this method OD-SARSA (offloading decision-based SARSA). We compared our proposed method with reinforcement learning based Q learning (RL-QL), and it is concluded that the performance of the former is superior to that of the latter.

INDEX TERMS Mobile devices, edge computing, mobile edge computing, edge cloud computing, virtual machines, access points.

I. INTRODUCTION

In recent years, the massive growth of computationally intensive and delay sensitive mobile applications, such as online gaming, image or signal processing (e.g., facial recognition), augmented reality, and real-time translation services, have been imposing heavy computation demands on resource-constrained mobile devices (MDs). As MDs are limited in terms of computation, battery, and storage capacity, there is a growing trend to offload or transfer computation intensive tasks to powerful remote computing platforms. This method is referred to as computation offloading. It reduces energy consumption for local processing and therefore prolongs battery life.

Mobile cloud computing (MCC) [1] is a well-known computation offloading model for MDs [1]. In MCC, user devices can utilize the resources of dedicated remote cloud servers for executing their tasks. These servers have high power, CPU, and storage capabilities. However, the long distance between the MDs and the cloud server lead to substantial communication costs in terms of latency and energy, negatively influencing real-time applications [2]. Therefore, in recent years, the computation and storage capabilities of the remote cloud have partially migrated to the edge server (near the MDs). This concept is called mobile edge computing (MEC) [3].

MEC provides information technology services and cloud computing capabilities at the mobile network edge. MEC is

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

implemented by a dense deployment of computational servers or by strengthening already deployed edge entities, such as small cell base stations (BS) with computation and storage resources. The objective of MEC is to ensure efficient network operation and service distribution, reduce latency, and offer an enhanced user experience [3], [4]. MEC offloads computation intensive applications to the cellular network edge. Smart cities can benefit from offloading to edge servers in the framework of the so-called cyber–physical–social systems (CPSSs), as in traffic violation tracking cameras, or drone services for delivery or geological survey purposes. Each edge node processes the data itself rather than forwarding them to a central remote cloud. Consequently, MEC can improve user experience quality (QoE) and meet service quality (QoS) requirements, such as low latency and energy consumption. Moreover, unlike MCC, MEC pursues a decentralized framework where the edge servers are deployed in a distributed manner.

Despite the great potential of MEC, there remain several challenges. As discussed before, real-time mobile applications are highly sensitive in terms of latency and energy consumption. However, owing to the randomness and dynamics of mobile edge networks, the long execution time of these applications can lead to high energy consumption. Most studies indicate that the long execution time is one of the major challenges in MEC [5], [6]. Hence, there is a need for an efficient computation-offloading framework for MEC. Furthermore, MDs determine when offloading should be performed, and what part of a given task should be offload to an edge server. However, developing an effective dynamic partitioning method for accurate offloading decision making is a challenge in MEC. Moreover, determining where to offload a task in a multi-edge network for minimizing the latency of service computing (close proximity edge or adjacent edge network or remote cloud) is another challenge. In addition, the limited computational resources of mobile edge servers should be efficiently utilized so that QoS requirements may be met (e.g., latency requirement). Furthermore, user mobility, the heterogeneity of edge node resources, and the physical distribution of MDs impose additional challenges for computation offloading in edge computing.

A number of methods have been developed to overcome some of these challenges [13]–[16]. However, these studies did not consider the benefit of using adjacent edges to serve offloadable tasks when the nearest edge server cannot serve these tasks. Another limitation is that all these studies used off-policy-based reinforcement learning techniques for resource allocation management, such as the Q-Learning method. This technique depends on the previous workload state, ignoring the current state. Moreover, current studies lack an efficient dynamic multi-objective optimization decision scheme for selecting the tasks to be offloaded. In the present study, we will resolve these issues and improve the offloading performance by proposing a dynamic framework that considers both servers and users' standpoints. In particular, we are concerned with 1) Computation offloading to the mobile edge using the system utility of the MEC network to balance processing delay and energy consumption, 2) determining which part/module or process of a mobile application should be offloaded using deep reinforcement on-policy learning such as state-action-reward-state-action (SARSA), 3) determining where to offload the part/module or process in a multi-edge network, and 4) ensuring efficient resource management in the MEC servers.

In this study, we address the question of developing an efficient resource management model for the selected MEC server in a multi-edge network by proposed an offloading decision-based SARSA method (OD-SARSA). Additionally, we consider the problem of managing mobility when the MDs move from one region to another. Accordingly, we should design and develop an efficient resource management model to enhance MEC server utilization through task scheduling and load balancing. As MEC suffers from limited computational resources, compared with central MCC, it becomes imperative to allocate these resources efficiently. The proposed resource allocation will enable meeting QoS requirements (e.g., latency) with minimal effort. Therefore, the main contributions of this study as follows:

- We propose a MEC system model considering both computing time delay and power consumption, and we formulate it as an optimization problem. In particular, we propose an offloading decision-based SARSA (OD-SARSA) using reinforcement learning to make the optimal offloading decision for reducing system cost in terms of energy consumption and computing time delay.
- We compared our proposed OD-SARSA with RL-QL and concluded that the former performs better than the latter.
- We analyzed the effect of optimal offloading decision factors and reduced cost by changing the main parameters and analyzing the results, leading to real-world application.

This paper is organized as follows. Section 2 reviews related work. Section 3 describes RL based on SARSA. Sections 4 describes the system model of MEC as a communication model, model of the task, and model of computation. Section 5 then describes the SARSA learning method based autonomic computation offloading. Finally, Section 6 presents the performance evaluation results and our conclusions.

II. RELATED WORK

With the rapid advancement of communication technology, MEC is emerging as a promising technology. An MD uses remote execution (offloading) to enhance a mobile use's QoS by reducing energy consumption and increasing performance. We will focus on previous studies concerned with the offloading process (how and where to offload), the partition of mobile applications, and resource allocation, which affect offloading efficiency (performance) and energy consumption. Few studies have focused on computation offloading in MEC, although several options can be used on the MEC servers depending on the conditions of the mobile network.

Thus, an efficient cloud-path selection method is required to select the best resource.

Reducing execution time (T) is one of the objectives of computation offloading in MEC. Execution time is the sum of local execution time (T_l) and remote execution time (T_o). The latter can be further divided into transmission delay to the ME (T_{od}), processing time at the ME (T_{op}), and receiving time from the ME (T_{or}) An offloading decision is not taken unless $T_l > T_o$. The aim is to minimize computation time, as discussed in [7]. This is achieved by using a one-dimensional search method, so that an effective offloading decision can be made depending on the queuing state buffer of the application, available energy in the MD and the MEC server, and the communication status between the MEC server and the MD. This algorithm was compared with greedy offloading, local execution, and cloud execution. The simulation demonstrated that execution time can be reduced by up to 81% and 44% as the arrival of the applications. The limitation of this method is that to make a decision, the MD as a client requires feedback from the MEC. In [8], the low-complexity Lyapunov optimization dynamic computation offloading algorithm was proposed. In [9], proposed system to leverage from the ability of computing and storage capacity available in the edge servers. In [10], a new computation offloading model in MEC was introduced. Its principle is to enable the use of virtual resources in the edge cloud to reduce resource and energy consumption and improve the performance of the application. In [11], the authors proposed a novel framework for computation offloading from an MD to an edge server considering CPU availability so that execution time may be reduced in both the MD and the server. In [12], an opportunistic computation offloading scheme was proposed for data mining in MDs and the edge network to reduce execution time and power consumption. In [12], the authors developed a distributed computation offloading algorithm that can attain a Nash equilibrium so that superior performance may be achieved, and user size may be reduced through server mode selection [5]. In [13], a computation offloading method to a small cell cloud was analyzed, and its performance was evaluated.

Minimizing energy consumption (E) and achieving an acceptable execution time is one of the objectives of computation offloading in MEC. If an MD executes all computations locally, E_l denotes the energy consumption; otherwise, the computation is carried out remotely by offloading to the edge. In this case, (E_o) is the energy consumption and is the sum of the transmission energy to the ME (E_{od}), the energy for processing at the ME (E_{op}), and the energy for receiving the result from the edge (E_{or}) The offloading decision is not made unless $E_l > E_o$ *when* $T_l > T_o$ when $T_l > T_o$. In [14], the authors proposed computation offloading to reduce energy consumption in the MD when the computation time constraint is satisfied. A constrained Markov decision process was proposed to solve the optimization problem. The author of [15] proposed an energy-efficient computation offloading algorithm in which the decision making is performed according to the following principles, 1) the MD considers its execution time and power consumption constraints, offloading to the ME is performed when the MD cannot satisfy the computation time constraint, and local execution is selected when the power depletion is below the determined threshold and the execution constraint is satisfied, 2) the offloading priority is high. Third, given the radio resource allocation priorities, experiments demonstrated that this algorithm can reduce energy consumption by up to 15%. Using the cloud radio access network (C-RAN) service, the authors of [17] presented a computation offloading algorithm from mobile to remote cloud radio heads to reduce energy consumption and improve user QoE by minimizing the response time of the app. The Lyapunov optimization algorithm makes the offloading decision depending on the frequencies of the CPU-cycle for mobile execution and the transmission energy for computation offloading [8]. In [16], the authors designed an autonomous and energy-efficient offloading scheme that uses a mathematical model for the energy consumption at the ME for the mobile application, considering the energy consumed by the interaction among the tasks in the same application. In [17], the authors proposed a new game theoretic approach to enhance the edge computing throughput and reduce energy consumption on the edge server.

In [20], it was proposed that the computation offloading decision should satisfy the trade-off between delay and energy consumption at the ME and UE. This study used the Nash equilibrium distributed computation offloading algorithm, in which the computation offloading decision depends on certain weight parameters, and the effective channel is chosen to transmit data. The numerical results demonstrated that this algorithm has superior performance when the application is computed at the MEC server rather than locally. In [18], the authors developed a code offloading model and decision-making process that reduce the applicatio's response time and the MDs' energy consumption. The offloading decision is made based on the method of Lagrange multipliers, and a nonlinear optimization solver is used instead of solving a complex linear optimizing problem.

Proper resource allocation should follow the decision-making regarding partial or full offloading. Resource allocation is influenced by the partitioned and paralleled computation offloading ability of the application. If offloading is impossible, then the partitioned and paralleled applications are allocated only one node for the computing. The number of offloaded applications to the ME should satisfy the computing time energy consumption requirements [19]. The application should determine where offloadable task should be placed, depending on the computing resources available at the ME. Reference [20] is similar to [22]; however, it not only minimizes computation time but also reduces energy consumption at the ME. The authors propose several hotspots in the density area of the UEs, which enable the MDs to access the ME using the enhanced node B (eNB). The proposed efficient policy by equivalent discretion is

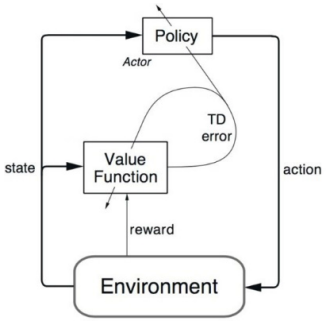


FIGURE 1. Reinforcement learning architecture.

called Markov decision processes (MDP). Reference [21] is similar to [19] and [20], as the main objective is to reduce computation time and energy consumption, as well as reduce channel overload, resource consumption, and computation cost of virtual machine (VM) migration. In [21], the authors use enhanced small cells (SCeNBs) as service nodes at the ME, and each MD is allocated a VM at an SCeNB. This reduces the communication delay because the SNeNBs are characterized by high-quality data transmission.

III. REINFORCEMENT LEARNING BASED ON SARSA LEARNING

RL is a part of machine learning [22]. It consists of taking appropriate action to increase the reward in specific states. Various programs and machines/devices use it to find the best behavior or possible path in a given state. RL differs from supervised learning in that the learning data contain the answer key. Thus, in supervised learning, the model is trained on the correct answer itself, whereas in RL, there is no answer, but the reinforcement agent determines how a certain task is to be carried out. When a dataset is not available, learning is performed through experience. The basic principle of RL is the following: The input must be an initial state from which the models start. The output consists of several potential results because there are several solutions to a specific problem. Training depends on the input, the model will return the value of the state, and the user will decide to punish or reward the model based on its results or output. The model learns continuously, and the best solutions are determined based on the maximum reward. RL involves an environment and agent, where the agent selects the most appropriate action from the environment states. The environment generates the next state based on an action obtained from another policy and rewards the generated state when the agent takes the action, as shown in Fig. 1. SARSA and Q-learning are two commonly used model-free RL techniques. They have different exploration policies and similar exploitation policies. Q-learning is an off-policy technique in which the agent learns based on the action by another policy, whereas SARSA is an on-policy technique, where learning is based on the current action by the current policy. RL has proved efficient in resource allocation [23], cloud computing, and computation offloading [22]. The policy π estimates the next (s, a) based on the current a state-action (s, a). To do this, we use temporal-difference (TD) to update the rule applied at every timestamp by allowing the agent to transition from one pair of state-action to another pair.

TABLE 1. Reward comparison between RL algorithms.

	Q	SARSA	R	AC
Q	-	10:44	13:41	17:38
SARSA	10:44	-	30:23	28/25
R	41:13	23:30	-	25:26
AC	38:17	25:28	26:25	-

To solve complex, large state-space problems, the deep SARSA function is updated as

$$Q(S_t, A_t) = R(S_t, A_t) + \gamma Q(S_{t+1}, A_{t+1}) \tag{1}$$

where $Q(S_t, A_t)$ is the value of Q for the action A in system state S at time t, $R(S_t, A_t)$ is the reward when the agent selects the action A_t at state S_t, and γ denotes the discount factor; the epsilon-greedy policy is used to select the best action A_{t+1} in the current state S_{t+1}.

Numerous traditional reinforcement learning models have been used for computation offloading. For example, in [24], an RL technique was used for complicated video games, and several different RL approaches, such as SARSA learning, Q learning, GQ, actor-critic, and R learning, were compared. The results are shown in Table 1, which is reproduced from that paper and shows that the SARSA outperforms other RL algorithms, as it obtained the greatest rewards.

Markov decision processes (MDPs) are used in RL for appropriately increasing the reward in the training task of an agent interacting with the environment [25]. Therefore, the future reward at time t is define as

$$R_t = \sum_{k=0}^{T} \alpha^k r_{t+k+1} \tag{2}$$

where $\alpha \in (0, 1]$ is a discount factor, and r_t is the reward when action a is taken at time t. When the agent takes the action a under the policy π in state S at the time t, denoted by $Q^\pi(s, a)$. Thus,

$$\begin{aligned} Q^\pi(s, a) &= E_\pi \{R(t) \mid s_t = \mathrm{s}, \quad a_t = a\} \\ &= E_\pi \left\{ \sum_{k=0}^{\infty} \alpha^k r_{t+k+1} \mid s_t = \mathrm{s}, \quad a_t = a \right\} \end{aligned} \tag{3}$$

where E is the expected reward, and π is the policy function for the action A_t. The aim of the training task is to acquire the maximum rewards and obtain the optimal state and action of $Q^\pi(s, a)$. There are two methods in RL. One is called Q-learning, and the other SARSA [26]. In this study, we will use SARSA, as it has been demonstrated that this method can select a safe path. This is considered appropriate in the present study, which is concerned with the selection of an optimal and safe path for offloading intensive tasks to the edge cloud. SARSA is an on-policy technique, that is, the next action a^* depends on the value of the current state s_t and current action a_t. The equation for updating state and action values is

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s^*, a^*) - Q(s, a)] \tag{4}$$

In SARSA learning, the training task is a quinary (s, a, r, s^*, a^*), which is updated sequentially.

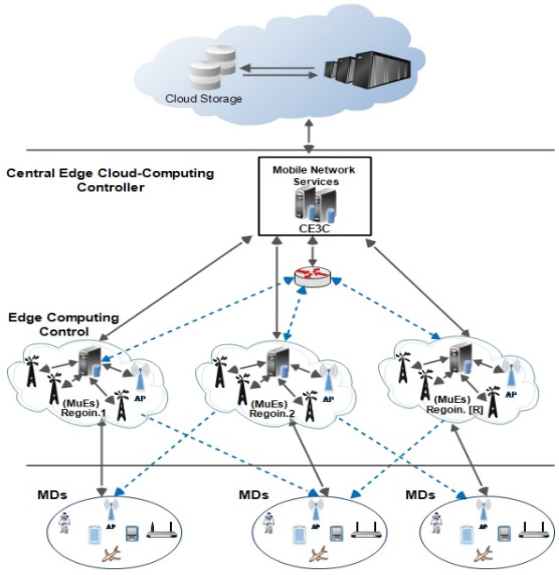


FIGURE 2. High-level overview of computation offloading in MEC model.

IV. SYSTEM MODEL OF MOBILE EDGE COMPUTING

The mobile edge system (MES) model is shown in Fig. 2. MES is constructed on a telecommunication infrastructure, such as BS/LTE. The MDs (e.g., smartphones, tablets, robots, and drones) connect to the edge computing control at the BS/LTE in the adjacent location (region) to the computation offloading. The edge computing controller in each region manages multi-edge mobile computing, receives the offloaded tasks from the MDs, and chooses an effective edge node to address them as a task model. In the mobility status, when an MD moves from one region to another, the processing results of the offloaded tasks are sent to the corresponding MD over the central edge cloud-computing controller (CE3C) and edge-computing controller for the adjacent region. The components of the edge network have high storage and computation capabilities, which are used to create a virtual server offering mobile edge services as a computing model. If a workload demands resources beyond what the edge server can support, the request is redirected over the main network (CE3C) to the cloud services on the other side of the network as the resource management model.

A. COMMUNICATION MODEL

We assume that there are MEC networks (MECNs) in more than one region, as shown in Fig. 2, which consist of multiple APs, multi-edge servers, and n MDs denoted by $n = \{1, 2, \ldots..,n\}$. An MD can connect to the MECN through an AP or mobile network. Depending on certain parameters such as edge servers' workloads, response time, or latency and energy consumption, the MDs should find an efficient location in the network to perform offloading. The offloading action is denoted by $A = \{a_1, a_2, \ldots..,a_n\}$, depending on the offloading decision, where X_n represent the offloading decision $X_n = \{0, 1, 2, 3\}$ (nearest the edge server: $X_n = 1$, adjacent to the edge server: $X_n = 2$, remote cloud: $X_n = 3$, or local computing: $X_n = 0$). The offloading decision is influenced by the bandwidth B_n and computing delay, which depends on the processing frequency f_n.

The communication bandwidths between MDs and offloading location are denoted by B^e, B^a, B^c, which represent edge server bandwidth, adjacent edge server bandwidth, and cloud bandwidth, respectively, as the end-to-end bandwidth. Moreover, the total communication delay for a certain MD is denoted by T_n. Additionally, p_n^t represents the power consumption for task transmission, and p_n^r the receiving power consumption. Therefore, depending on certain parameters such as edge servers' workloads, response time, or latency and energy consumption, the MD should find an efficient location (nearest the edge server or adjacent to the edge server or remote cloud) to offload its tasks. Eventually, after the offloading process to the nearest edge server or adjacent edge server has been completed, an efficient resource allocation method is required on the edge server.

B. TASK MODEL

We assume that each MD has $\boldsymbol{M}$ independent massive real-time tasks, which can be executed locally in the MD or remotely in the MEC network by the computation offloading. Therefore, tasks cannot be partitioned into subtasks to be processed in multiple devices [27]. Task size is denoted by $\boldsymbol{D_n}$ (transferred data size), and $\boldsymbol{R_n}$ denotes the computation resources required to serve this task (CPU cycles number). Therefore, $\boldsymbol{D_n}$ and $\boldsymbol{R_n}$ are positively related:

$$\boldsymbol{R_n = \theta D_n, \theta\ constant.}$$

Regardless of whether the task is executed locally by the MD or in the MEC network, $\boldsymbol{D_n}$ does not change.

C. COMPUTATION MODEL

1) LOCAL PROCESSING TIME

When the decision unit decides to process a task in the MD ($\boldsymbol{X_n = 0}$), the time processing per task is denoted by $\boldsymbol{T^l}$. This includes the computing delay of the local CPU. Therefore, the processing time is

$$\boldsymbol{T^l_{nm} = \frac{R_{nm}}{\mathcal{F}^L_N}} \tag{5}$$

Similarly, the corresponding power consumption for task $\boldsymbol{M_n}$ of user $\boldsymbol{n}$ is denoted by $\boldsymbol{P^l}$ and is defined as

$$\boldsymbol{P^l_{nm} = D^i_{nm} p_l} \tag{6}$$

where p_l denotes the power consumption when the task is processed in the MD. Therefore, the cost of local processing is the combination of the local processing time and local power consumption:

$$C^l_n = \sum\nolimits_{n=1}^{N} (\alpha T^l_n + \beta P^l_n) \tag{7}$$

where α and β are constant weighting parameters corresponding to the time and power cost of the task.

2) EDGE PROCESSING TIME

When the decision unit decides to offload the task to an edge server ($\boldsymbol{X_n = 1}$), the time processing per task is denoted by $\boldsymbol{T^e}$. This includes the transmission delay and computing delay. The computing delay depends on the CPU frequency of the edge server and other resources. Therefore, the processing time is

$$\boldsymbol{T^e_{nm} = \frac{1}{\mathcal{F}^e \mathcal{B}_n}(\mathcal{B}^e_n R^e_n + \mathcal{F}^e D_n)} \tag{8}$$

where $\boldsymbol{\mathcal{F}^e}$ and $\boldsymbol{\mathcal{B}_n}$ denote the CPU frequency of the edge server and the communication bandwidth, respectively. Similarly, the corresponding power cost for task $\boldsymbol{M_n}$ of user $\boldsymbol{n}$ is denoted by $\boldsymbol{p^e}$ and is defined as

$$\boldsymbol{P^e_{nm} = T^e_{nm} p_e} \tag{9}$$

Therefore, the processing cost of edge computing is the combination of edge computing time and power consumption, as follows:

$$\boldsymbol{C^e_n = \sum_{n=1}^{N} (\alpha T^e_n + \beta P^e_n)} \tag{10}$$

3) PROCESSING TIME OF ADJACENT EDGE SERVER

When the decision unit decides to offload a task to an adjacent edge server ($X_n = 2$), the time processing per task is denoted by T^a. This include the transmission delay and computing delay. The computing delay depend on the CPU frequency of the adjacent edge server and other resources. Therefore, the processing time is

$$T^a_{nm} = \frac{1}{\mathcal{F}^a \mathcal{B}_n}(\mathcal{B}^a_n R^a_n + \mathcal{F}^a D_n) \tag{11}$$

where $\mathcal{F}^a$ and $\mathcal{B}_n$ represents the CPU frequency of the adjacent edge server and communication bandwidth, respectively. Similarly, the corresponding power cost for task M_n of user n is denoted by p^a and is defined as

$$P^a_{nm} = T^a_{nm} p_a \tag{12}$$

Therefore, the processing cost of an adjacent edge computing server is the combination the corresponding computing time and power consumption, as follows:

$$C^a_n = \sum_{n=1}^{N} (\alpha T^a_n + \beta P^a_n) \tag{13}$$

4) REMOTE PROCESSING TIME

When it is decided to offload a task to the remote cloud server ($X_n = 3$), the time processing per task is denoted by T^c. This includes the transmission delay and the computing delay. The former corresponds to two directions: from the *MD* to the edge server or adjacent edge server ($T_{m,e}$ *or* $T_{m,a}$), and from the edge server to the remote cloud ($T_{e,c}$ *or* $T_{a,c}$). We assume that $T_{m,e}$ and $T_{m,\mathrm{a}}$ are similar, and thus we neglect one of them. The computing delay depends on the CPU frequency of the assigned remote server and other resources.

We can compute the task processing time in the cloud by the following equation, as in [28]:

$$T^c_{nm} = \frac{1}{\mathcal{F}^c \mathcal{B}_n}\left(\mathcal{B}^c_n R^c_n + \mathcal{F}^c D_n\right) \tag{14}$$

where $\mathcal{F}^c$ denotes the CPU frequency for processing in the cloud for each user. The total time cost involving the processing and transmission delay is:

$$T^c_n = T^c_{nm} + T^e_{nm} \tag{15}$$

Similarly, the corresponding power cost for task M_n of user n is denoted by p^c and is defined as

$$P^c_{nm} = T^c_{nm} p_c \tag{16}$$

Therefore, the processing cost of a remote cloud server is the combination of computing time and power consumption, as follows:

$$C^c_n = \sum_{n=1}^{N} (\alpha T^c_n + \beta P^c_n) \tag{17}$$

The total cost C_{total} of the MEC offloading system can expressed as

$$\begin{aligned} \mathrm{C_{total}} = \sum_{\mathrm{n=1}}^{\mathrm{N}} \Big(& \frac{\mathrm{C^l_n}\,(1-\mathrm{X_n})\,(2-\mathrm{X_n})\,(3-\mathrm{X_n})}{6} \\ & + \frac{\mathrm{X_n}\,(2-\mathrm{X_n})\,(3-\mathrm{X_n})\,\mathrm{C^e_n}}{2} - \frac{\mathrm{X_n}\,(\mathrm{X_n}-1)\,(\mathrm{X_n}-3)\,\mathrm{C^a_n}}{2} \\ & + \frac{\mathrm{X_n}(\mathrm{X_n}-1)(\mathrm{X_n}-2)\mathrm{C^c_n}}{6} \Big) \end{aligned} \tag{18}$$

We assume that there are five MDs in the network. MDs 1 and 5 choose to execute tasks locally, that is, $X_n = 0$, MD chooses to offload tasks to the edge point, that is, $X_n = 1$, MD 3 chooses to offload tasks to an adjacent edge, that is, $X_n = 2$, and MD 4 chooses to offload tasks to the remote cloud server, that is, $X_n = 3$. We use formula (14) to calculate the computing time and power consumption, that is, $C_{total} = C^l_n + C^e_n + C^a_n + C^c_n$. The notations used in this study are defined in Table 2.

D. OPTIMIZATION PROBLEM FORMULATION

Our objective to minimize the processing and transmission delay and reduce the power consumption for these two operations. The minimized cost is denoted by Q_{min}. We assume that the transmission and receiving bandwidth are equal $\beta^t_n = \beta^r_n$. The optimization problem of system utilization is formulated as follows:

$$\begin{aligned} Q_{min} = \mathit{minimize} \sum_{n=1}^{N} \Big(& \frac{C^l_n\,(1-X_n)\,(2-X_n)\,(3-X_n)}{6} \\ & + \frac{X_n\,(2-X_n)\,(3-X_n)\,C^e_n}{2} - \frac{X_n\,(X_n-1)\,(X_n-3)\,C^a_n}{2} \\ & + \frac{X_n\,(X_n-1)\,(X_n-2)\,C^c_n}{6} \Big) \end{aligned} \tag{19}$$

under the constraints

$$\sum_{n=1}^{N} B^t_n \le \beta^t; \sum_{n=1}^{N} \beta^r_n \le \beta^r, (\beta^t_n, \beta^r_n) \ge 0, \forall n \tag{20}$$

TABLE 2. Notation list.

Notation	Definition
D_n	Uploaded real-time massive tasks
R_n	Computation resources required to execute tasks
T_l	Local time processing of task
T_e	Time processing of task on the edge server
T_a	Time processing of task on adjacent edge server
T_c	Time processing of task on remote cloud server
p_n^l	Power consumption when task is processed locally
p_n^e	Power consumption when task is processed on edge server
p_n^a	Power consumption when task is processed on adjacent edge server
β_n^t	Uploading bandwidth
β_{nm}^r	Downloading bandwidth
X_{nm}	Offloading decision {0,1,2,3}
C_{total}	System utility cost services
$\mathcal{F}^c$	Processing Frequency in the cloud for each user
S_{nm}^c	Local power consumption per task
p_l	Local processing time
$\mathcal{F}^c$	Number of processing cycles for each uploaded task
C_n	Combined cost for computing time and power consumption

where

$$\beta_{n>0}, \quad \forall n \in N$$
$$X_{n\in\{0,1,2,3\}}, \quad \forall n \in N$$

$X_n = \{X_1, X_2, \ldots, X_n\}$ is the offloading decision; it has four modes and takes four values: 0, 1, 2, and 3. Additionally, the bandwidth is limited by constraint (16) on transmission tasks and receiving results to prevent congestion on the server, which may cause significant delays. The optimization problem (15) is considered a mixed-integer problem, which is generally difficult to solve. To minimize the system utilization cost, we propose a reinforcement learning technique based on deep SARSA.

V. SARSA LEARNING AUTONOMIC COMPUTATION OFFLOADING

We assume that there multiple options for executing an offloadable task at the nearest edge, at an adjacent edge, or in the remote cloud. To determine the optimal location, we used deep reinforcement learning (SARSA). Thus, the performance of the edge server (ES) depends on the resource allocation mechanism and improves the simultaneous execution of tasks. However, the scheduling and resource allocation on the edge server are NP-hard scheduling problems. Most current studies use game theory and reinforcement learning. Therefore, we will develop an efficient resource allocation mechanism to enhance MEC server utilization owing to the limited power and computational resources compared with cloud-computing servers. In our mechanism, the offloading decision algorithm (OD-SARSA) will be used for solving the resource management problem on the ES based on parameters derived from its environment, such as data size, bandwidth, edge-server workload, signal strength, and energy consumption. OD-SARSA is an effective method to achieve high utilization on the ES, owing to its ability to function as an on-policy technique, that is, it considers the current resource consumption state in the ES environment, which is highly important for resource management. For example, the current state obtained from the SARSA algorithm is used to determine whether current VMs should be employed or new VMs should be created on the ES. In the latter case, the VM manager on the ES is responsible for creating the VMs and assigns VMs to each offloaded task. One approach for the VM manager could be to activate VMs only on a few servers, depending on the offloaded tasks, whereas the other servers are put into sleep mode to save energy. However, the VM manager should also consider the users' latency requirements, as the servers may be overload with many offloaded tasks, resulting in a load balancing issue. This will be more challenging when there is uncertainty in task arrival, and there is no central controller.

A. OFFLOADING-DECISION-BASED SARSA METHOD (OD-SARSA)

We should solve the optimization problem (19) and meet the QoS (e.g., energy consumption, or delay) requirements so that a deep SARSA function may be used to make an efficient decision X_{nm} for offloading of each task to the appropriate location . The input of the SARSA function is the uploading bandwidth β_{nm}^t and downloading bandwidth β_{nm}^r as states. The output of the system is the value of Q for each state S_t of the corresponding action A_t. Each time, the agent selects the suitable action with regard to the Q value. The result of the action is to make identical adjustment to the offloading decision X_{nm} and determine the appropriate location (nearest to the edge server or adjacent to the edge server or remote cloud), as well as resource allocation β_{nm}^t and β_{nm}^r.

The SARSA function considers an on-policy mechanism, which implies that the agent learns based on its up-to-date action as a consequence of the current policy. OD-SARSA is described in Algorithm 1. It performs offloading and is trained through deep leaning. In SARSA, an epsilon-greedy policy is used for state transition; the Q value in the preceding state is updated by Equation (15), where the next action is selected by an epsilon-greedy policy. In the system, there are a target network and an evaluation network. The input system is the current state, and the following or next state are obtained after the selection of an action. We can choose the action based on the epsilon greedy (ε) policy. We use a probability of $1 - \varepsilon$ and select the best action, and thus the output of the target network is changed according to the reward, and the parameters are updated in each state, and a new policy is imposed.

Therefore, the actions a of the agent can be defined as offloading to valid locations (nearest, adjacent, and remote). We assume 10 possible actions, as a follows: A_l is local processing, A_N is offloading to the nearest edge, A_a is offloading to an adjacent edge, A_R is offloading to the remote cloud, A_{NA} is migration from the nearest edge to an adjacent edge, A_{AN} is migration from an adjacent edge to the nearest edge, A_{NR} is migration from the nearest edge to the remote cloud,

Algorithm 1 OD-SARSA

1: Input: Number of MDs N and task size D_n
2: Output: efficient offloading decision, cost reduction and bandwidth allocation
3: Initialize the network parameters with upload and download bandwidth, and processing cycle number
4: Initialize the number of iterations (episodes), let I = 100
5: for iteration I < 1,2,3,…., I do
6: Select *"Action"* randomly.
7: Compute *"Current State"* according to formula No. 3
8: **if** *"Current state"* < $(St + 1)$ **then**
9: Set $r_t = 1$
10: **else if** $S_t > S_{t+1}$ **then**
11: Set $r_t = -1$
12: **else**
13: Set $r_t = 0$
14: **end if**
15: Obtain reward r_t and next state S_{t+1} after execution of a_t.
16: Set this as (S_t, a_t, r_t, S_{t+1}).
17: Compute the Q-value y_t from the target deep QL $y_t = r_{t+1} + \gamma QS_{t+1}, a_{t+1}$
18: Execute the algorithm of gradient descent to reduce $(y_t - q(s_{t+1}, a_{t+1}); \alpha)^2$
19: Update q-value: $q^*(s,a) = (1-\alpha)\, q(s,a) + \alpha(R_{t+1} + \gamma q(s_{t+1}, a_{t+1}))$
20: **end for**

A_{RN} is migration from the remote cloud to the nearest edge, A_{AR} is migration from an adjacent edge to the remote cloud, and A_{AR} is migration from the remote cloud to an adjacent edge. Thus, the actions of the agent can be represented as $A(t) = \{A_1(t), A_2(t) \ldots\ldots\ldots\ldots A_k(t)\}$, where $A_k(t)$ denotes the k-th offloading decision. If $A_k(t) = 0$, the task is processed locally, if $A_k(t) = 1$, the task is offloadable and processed on the edge server, if $A_k(t) = 2$, task is executed at the adjacent node, and if the $A_k(t) = 3$, the task is processed on the remote server. The agent learning state S can be defined as the resources of the edge computing: processing (S_p), memory (S_m), and network bandwidth (S_b). Thus, the current system state can be represented as $(t) = \{S_1(t), S_2(t) \ldots\ldots S_n(t)\}$, where $S_i = (S_{pi}, S_{mi}, S_{bi})$, $i = 1 \ldots\ldots n$.

In this system, a particular learning agent does not have information regarding the overall state of all nearest edges; the agent only has information regarding its local state. There is collaboration and communication between the agents to offload tasks to appropriate locations at the edge network (nearest or adjacent edge) or in a public cloud.

Reward function: The main objective of computation offloading is to reduce the processing delay of intensive tasks. This primarily depends on the capability of the edge network, that is, processing, memory, and bandwidth. CE3C determines its processing capability by detecting its state, estimates the response time, and chooses the appropriate location accordingly. After an action is performed, result $S(t)$ is obtained. If $S(t)$ is smaller than $S(t-1)$, a positive reward $R(t) = +1$ is given. If S_t is larger than S_{t+1}, we give a negative reward $R(t) = -1$; otherwise, $R(t) = 0$. The reward allows the agent to learn efficient decision making for resource allocation and offloading for reduced energy consumption.

To update the value of Q for the state after an action, we use the Bellman equation as follows:

$$q^*(s,a) = E[R_{t+1} + \alpha\, q^*(s_{t+1}, a_{t+1}) \tag{21}$$

The value of Q for a given state and action should be as close to the right-hand of the Bellman equation as possible so that the Q-value will finally converge to a safe value q^*.

$$q^*(s_{t+1}, a_{t+1}) - q(s,a) < 0, \text{ system state; non} - \text{offloading} \tag{22}$$

$$E[R_{r+1} + \alpha q^*(s_{t+1} - a_{t+1}) - E[\sum_{I=0}^{\infty} \propto^I R_{t+I+1}] \tag{23}$$

The method for computing the new value of Q for the state and action pair (s, a) at a certain time is

$$q^*(s,a) = (1-\alpha)\, q(s,a) + \alpha(R_{t+1} + \gamma q(s_{t+1}, a_{t+1})) \tag{24}$$

B. PERFORMANCE EVALUATION

We will now evaluate the proposed OD-SARSA algorithm. The model uses N task of M users to determine if the best action at a given time is to offload or not (local processing). We give data sizes as input and output for each user. We aim to find an optimal policy offloading function π. The offloading size can by expressed by NM, which increase with the number of tasks M per user N in MEC networks.

We assume that the number of mobile users is $N = 5$, and each user has five tasks. Table 1 shows all parameters that are used in reinforcement learning. We set the local processing time for an MD to $3.75 \times 10^{-7} s/bit$, and the corresponding power consumption to 3.55×10^{-6} J/bit. We assume that the size of all tasks is distributed between 10 and 35 MB. Regarding the other network parameters, such as bandwidth, we assume that the bandwidth for both uplink and downlink between a user and an edge server is 150 MB and may change depending on network conditions. The rate of the CPU of an edge sever is 9×10^8 cycle/s. The MDs' transmission and receiving energy consumption are both 1.60×10^{-6} J/bit. We train the model using 100 episodes.

There is close similarity between Q-learning and SARSA, but SARSA uses an on-policy technique. This encouraged us to use it for improved offloading performance to the ES, particularly because it does not depend on explicitly learning the agent's policy function. The results are shown in Fig. 3 and demonstrate that SARSA outperforms Q-learning, with an improvement rate of up to 8%. When the number of iterations increases, the improvement increases as well. It is noted that QL is better than SARSA in a faster training scenario

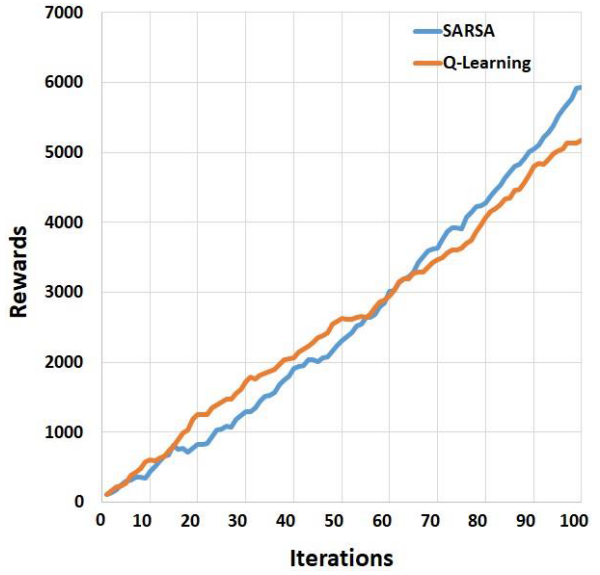


FIGURE 3. Performance of SARSA and Q-Learning.

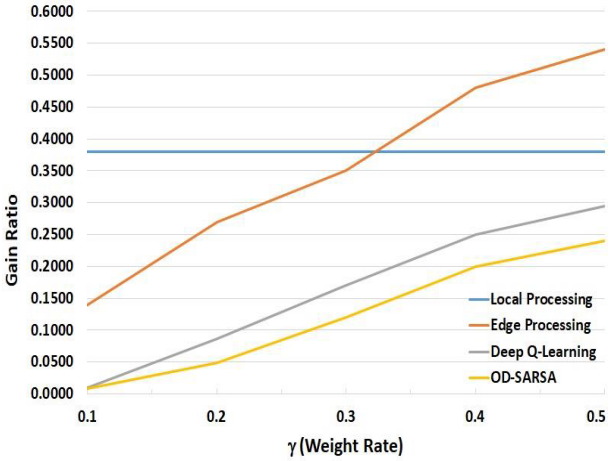


FIGURE 4. System utility under different parameters γ and μ.

(when the number of iterations is less than 50), but for more than 60 iterations, SARSA is consistently better than QL. We can conclude for increased the training iterations, the gap between SARSA and Q widens, with increased gain rewards. This affects performance in favor of the SARSA method.

The system utility under the different parameters γ and μ, which denote learning rate and weight rate respectively, are shown in Fig. 4 by comparing the proposed OD-SARSA with other algorithms: Q-learning, edge processing, and local processing. The results indicate the superiority of OD-SARSA to the other algorithms. The main problem with deep learning modules is choosing a learning rate and optimizer (the hyper-parameters). Therefore, we study our algorithm under different learning rates. After 100 iterations, we notice that a learning rate of 0.001 is stable and appropriate for the proposed method (Fig. 5). In contrast, the results for other values are unstable, and large dispersion are observed, particularly when LR = 0.01.

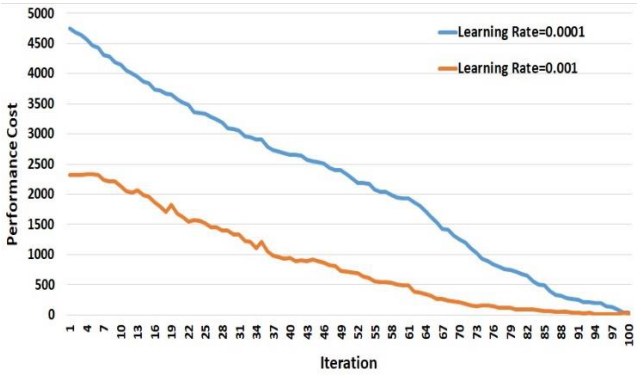


FIGURE 5. DO-SARSA performance under different learning rates.

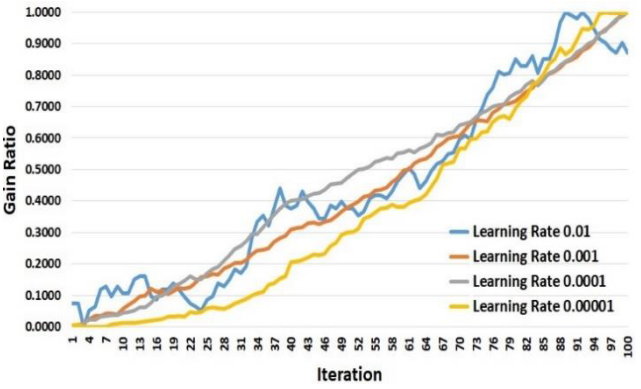


FIGURE 6. Performance of OD-SARSA under different learning rates.

Based on the comparison between various different learning rates, we studied the performance cost corresponding to 0.001 and 0.0001. We notice that the total cost of DO-SARS for a learning rate of 0.001 is significantly lower than that for 0.0001 (Fig. 6). At the beginning of the training, we notice that the gap is large owing to the increased performance cost. Nevertheless, as iterations increase, this gap decreases, and the costs are equal in the last iteration.

We observe that Q-learning correctly selects the optimal path in several applications, but it occasionally fails in critical stages, which require an important and critical decision, owing to the ε-greedy action selection. In our study, we demonstrated that SARS is better at making decisions in critical situations, as it is considered stable, particularly because it learns the safe path. This is highly important in making critical decisions. To attain better results in practice with on-policy RL techniques, the epsilon parameter should be reduced over time. Fig. 7 shows the effect of varying epsilon on the offloading decision. We notice that when $\varepsilon = 0.80$, we obtain satisfactory results, and maximum rewards are achieved; thus, this value was adopted in this study. Degenerate levels (0.20: 75) of course yield suboptimal results. It is conceivable that this caused by the short timescale the agent actions.

The result of the optimization problem (eq. 19) is shown in Fig. 8, where the number of offloadable and non-offloadable tasks can be seen. We notice that as the

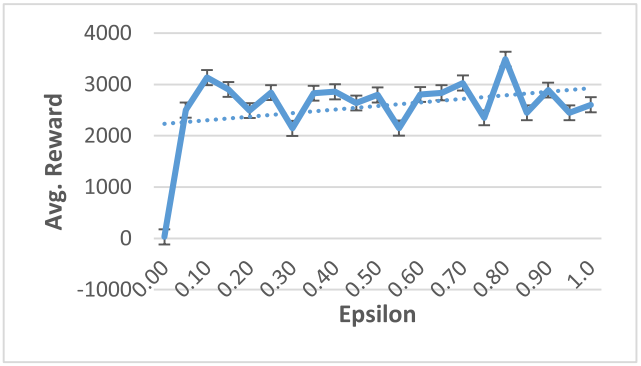


FIGURE 7. Effect of epsilon on acquiring rewards using OD-SARSA.

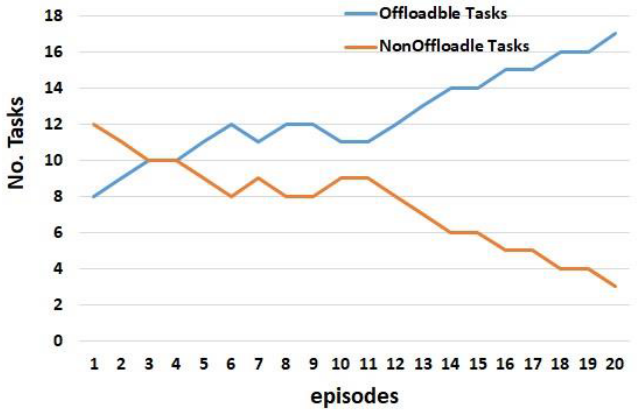


FIGURE 8. Number of offloaded tasks vs. non-offloadable tasks.

training iterations increase, the "offloadable" decisions increase, regardless of the offloading location (edge server, adjacent edge server, or a remote server). At the beginning of the training, the difference between these numbers is small, but subsequently, it gradually increases.

VI. CONCLUSION

In this paper, we assumed that there are MECNs in more than one region, consisting of multiple APs, multi-edge servers, and N MDs, where each MD has independent massive real-time tasks. The MD can connect to an MECN through an AP or a mobile network. Each task can be processed locally by the MD itself or remotely. There are three offloading options: nearest edge server, adjacent edge server, and remote cloud. We propose a reinforcement-learning-based SARSA method to solve the optimization problem for making decisions regarding offloading to one of the previously mentioned locations to reduce system cost, including energy consumption and computing time delay. It was demonstrated that on this problem, OD-SARSA performed better than RL-QL. Therefore, in offloading to adjacent edge servers, the proposed method resolves most challenges faced by CPSSs and achieves optimal results in terms of volume, variety, velocity, and veracity. In future, we will consider the code offloading on edge devices with GPUs that connected with mobile devices.

ACKNOWLEDGMENT

This work was supported by the Deanship of Scientific Research at King Saud University through the Vice Deanship of Scientific Research Chairs. Chair of Smart Technologies. The authors also thank the RSSU at King Saud University for their technical support.

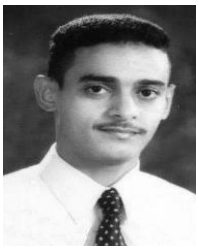
REFERENCES

[1] M. Satyanarayanan, "Mobile computing: The next decade," in *Proc. 1st ACM Workshop Mobile Cloud Comput. Services, Social Netw. Beyond (MCS)*. New York, NY, USA: ACM, Jun. 2010, pp. 1–6.
[2] G. H. Forman and J. Zahorjan, "The challenges of mobile computing," *Commun. ACM*, vol. 36, no. 7, pp. 75–84, 1993.
[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st MCC Workshop Mobile Cloud Comput.*, 2012, pp. 13–16.
[4] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—A key technology towards 5G," *ETSI White Paper*, vol. 11, no. 11, pp. 1–16, 2015.
[5] S. Ranadheera, S. Maghsudi, and E. Hossain, "Computation offloading and activation of mobile edge computing servers: A minority game," 2017, *arXiv:1710.05499*. [Online]. Available: http://arxiv.org/abs/1710.05499
[6] D. Mazza, D. Tarchi, and G. E. Corazza, "A cluster based computation offloading technique for mobile cloud computing in smart cities," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–6.
[7] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," 2016, *arXiv:1604.07525*. [Online]. Available: http://arxiv.org/abs/1604.07525
[8] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3590–3605, Dec. 2016.
[9] K. Habak, M. Ammar, K. A. Harras, and E. Zegura, "Femto clouds: Leveraging mobile devices to provide cloud service at the edge," in *Proc. IEEE 8th Int. Conf. Cloud Comput.*, Jun. 2015, pp. 9–16.
[10] X. Wei, S. Wang, A. Zhou, J. Xu, S. Su, S. Kumar, and F. Yang, "MVR: An architecture for computation offloading in mobile edge computing," in *Proc. IEEE Int. Conf. Edge Comput. (EDGE)*, Jun. 2017, pp. 232–235.
[11] F. Messaoudi, A. Ksentini, and P. Bertin, "On using edge computing for computation offloading in mobile network," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2017, pp. 1–7.
[12] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.
[13] J. Dolezal, Z. Becvar, and T. Zeman, "Performance evaluation of computation offloading from mobile device to the edge of mobile network," in *Proc. IEEE Conf. Standards for Commun. Netw. (CSCN)*, Oct. 2016, pp. 1–7.
[14] M. Kamoun, W. Labidi, and M. Sarkiss, "Joint resource allocation and offloading strategies in cloud enabled cellular networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2015, pp. 5529–5534.
[15] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang, "Energy-efficient offloading for mobile edge computing in 5G heterogeneous networks," *IEEE Access*, vol. 4, pp. 5896–5907, 2016.
[16] C. Luo, S. Salinas, M. Li, and P. Li, "Energy-efficient autonomic offloading in mobile edge computing," in *Proc. IEEE 15th Intl Conf Dependable, Autonomic Secure Comput., 15th Intl Conf Pervas. Intell. Comput., 3rd Intl Conf Big Data Intell. Comput. Cyber Sci. Technol. Congr. (DASC/PiCom/DataCom/CyberSciTech)*, Nov. 2017, pp. 581–588.
[17] A. Kaur and R. Kaur, "An efficient framework for improved task offloading in edge computing," in *Proc. Int. Conf. Intell., Secure, Dependable Syst. Distrib. Cloud Environ.* Cham, Switzerland: Springer, 2018, pp. 94–101.
[18] M. E. Khoda, M. A. Razzaque, A. Almogren, M. M. Hassan, A. Alamri, and A. Alelaiwi, "Efficient computation offloading decision in mobile cloud computing over 5G network," *Mobile Netw. Appl.*, vol. 21, no. 5, pp. 777–792, Oct. 2016.
[19] T. Zhao, S. Zhou, X. Guo, Y. Zhao, and Z. Niu, "A cooperative scheduling scheme of local cloud and Internet cloud for delay-aware mobile cloud computing," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Dec. 2015, pp. 1–6.
[20] X. Guo, R. Singh, T. Zhao, and Z. Niu, "An index based task assignment policy for achieving optimal power-delay Tradeoff in edge cloud systems," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–7.

[21] V. Di Valerio and F. Lo Presti, "Optimal virtual machines allocation in mobile Femto-Cloud computing: An MDP approach," in *Proc. IEEE Wireless Commun. Netw. Conf. Workshops (WCNCW)*, Apr. 2014, pp. 7–11.
[22] A. E. Eshratifar and M. Pedram, "Energy and performance efficient computation offloading for deep neural networks in a mobile cloud computing environment," in *Proc. Great Lakes Symp. VLSI (GLSVLSI)*, 2018, pp. 111–116.
[23] C.-F. Liu, M. Bennis, M. Debbah, and H. V. Poor, "Dynamic task offloading and resource allocation for ultra-reliable low-latency edge computing," *IEEE Trans. Commun.*, vol. 67, no. 6, pp. 4132–4150, Jun. 2019.
[24] A. Defazio and T. Graepel, "A comparison of learning algorithms on the arcade learning environment," 2014, *arXiv:1410.8620*. [Online]. Available: https://arxiv.org/abs/1410.8620
[25] D. Zhao, H. Wang, K. Shao, and Y. Zhu, "Deep reinforcement learning with experience replay based on SARSA," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Dec. 2016, pp. 1–6.
[26] G. A. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems," Ph.D. dissertation, Dept. Eng., Univ. Cambridge, Cambridge, U.K., 1994.
[27] L. Huang, X. Feng, L. Qian, and Y. Wu, "Deep reinforcement learning-based task offloading and resource allocation for mobile edge computing," in *Proc. Int. Conf. Mach. Learn. Intell. Commun.* Cham, Switzerland: Springer, 2018, pp. 33–42.
[28] J. Xu, Z. Hao, and X. Sun, "Optimal offloading decision strategies and their influence analysis of mobile edge computing," *Sensors*, vol. 19, no. 14, p. 3231, 2019.

TAHA ALFAKIH received the B.S. degree in computer science from the Computer Science Department, Hadhramout University, Yemen, and the M.Sc. degree from the Computer Science Department, King Saud University (KSU), Riyadh, Saudi Arabia. He is currently pursuing the Ph.D. degree with the Information Systems Department, KSU. He also works as a Researcher with the Computer Science College, KSU. His research interests include machine learning, mobile edge computing, and the Internet of Things (IoT).

MOHAMMAD MEHEDI HASSAN (Senior Member, IEEE) received the Ph.D. degree in computer engineering from Kyung Hee University, South Korea, in February 2011. He is currently an Associate Professor with the Information Systems Department, College of Computer and Information Sciences (CCIS), King Saud University (KSU), Riyadh, Saudi Arabia. He has authored or coauthored more than 180 publications, including refereed IEEE/ACM/Springer/Elsevier journals, conference papers, books, and book chapters. His research interests include cloud computing, edge computing, the Internet of Things, body sensor networks, big data, deep learning, mobile cloud, smart computing, wireless sensor networks, 5G networks, and social networks. Recently, his four publications have been recognized as the ESI Highly Cited Papers. He was a recipient of number of awards, including the Best Journal Paper Award from the IEEE Systems Journal, in 2018, the Best Paper Award from CloudComp 2014 Conference, and the Excellence in Research Award from King Saud University, in 2015 and 2016. He has served as the Chair and a Technical Program Committee Member of numerous reputed international conferences/workshops, such as IEEE CCNC, ACM BodyNets, and IEEE HPCC.



ABDU GUMAEI received the B.S. degree from the Computer Science Department, Al-Mustansiriya University, Baghdad, Iraq, the master's degree from the Computer Science Department, King Saud University, Riyadh, Saudi Arabia, and the Ph.D. degree from King Saud University, in 2019, all in computer science. He has worked as a Lecturer and taught many courses, such as programming languages at the Computer Science Department, Taiz University. He is currently an Assistant Professor with the College of Computer and Information Sciences, King Saud University. He has several types of research in the field of image processing. His research interests include software engineering, image processing, computer vision, machine learning, networks, and the Internet of Things (IoT). He has received a patent from the United States Patent and Trademark Office (USPTO), in 2013.

CLAUDIO SAVAGLIO received the Ph.D. degree in information and communication technology from the Department of Informatics, Modeling, Electronics and Systems (DIMES), University of Calabria. He is currently a Research Fellow with the Department of Informatics, Modeling, Electronics and Systems (DIMES), University of Calabria. He has been a Visiting Scholar with the Eindhoven University of Technology, The Netherlands; The University of Texas; the New Jersey Institute of Technology, USA; and the Universitat Politècnica de València, Spain. He is also the author of more than 30 articles in international journals, conferences, and book chapters. His research interests include autonomic and cognitive Internet of Things systems, cyber-physical networks, edge computing, and agent-oriented middleware and development methodologies. He has also served with different roles (the Chair, an Organizer, a Program Committee Member, a Guest Editor, and a Reviewer) in international journals, conferences, and book series.

GIANCARLO FORTINO (Senior Member, IEEE) received the Ph.D. degree in computer engineering from the University of Calabria (Unical), Italy, in 2000. He is currently a Full Professor of computer engineering with the Department of Informatics, Modeling, Electronics, and Systems, Unical. He is also a Guest Professor with the Wuhan University of Technology, Wuhan, China, a High-End Expert with HUST, China, and a Senior Research Fellow with the Italian National Research Council ICAR Institute. He is also the Director of the SPEME Laboratory, Unical, and the Co-Chair of Joint labs on IoT established between Unical and WUT and SMU Chinese universities, respectively. His research interests include agent-based computing, wireless (body) sensor networks, and the Internet of Things. He is the author of over 400 articles in international journals, conferences, and books. He is currently a member of the IEEE SMCS BoG and the IEEE Press BoG. He is also the Chair of the IEEE SMCS Italian Chapter. He is a cofounder and the CEO of SenSysCal S. r. l., a Unical spinoff focused on innovative IoT systems. He is a (founding) Series Editor of the IEEE Press Book Series on *Human-Machine Systems* and the EiC of *Internet of Things* (Springer) series and an AE of many international journals, such as the IEEE Transactions on Automatic Control (TAC), the IEEE Transactions on Human-Machine Systems (THMS), the IEEE Internet of Things Journal (IoTJ), the IEEE Systems Journal (SJ), IEEE SMCM, *Information Fusion*, JNCA, and EAAI.