This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3270242

Deep Reinforcement Learning-based Task Assignment for Cooperative Mobile Edge Computing

Li-Tse Hsieh, Hang Liu, *Senior Member, IEEE*, Yang Guo, Robert Gazda, *Member, IEEE*

Abstract—Mobile edge computing (MEC) integrates computing resources in wireless access networks to process computational tasks in close proximity to mobile users with low latency. This paper investigates the task assignment problem for cooperative MEC networks in which a set of geographically distributed heterogeneous edge servers not only cooperate with remote cloud data centers but also help each other to jointly process user tasks. We introduce a novel stochastic MEC cooperation framework to model the edge-to-edge horizontal cooperation and the edge-to-cloud vertical cooperation. The task assignment optimization problem is formulated by taking into consideration dynamic network states, uncertain node computing capabilities and task arrivals, as well as the heterogeneity of the involved entities. We then develop and compare three task assignment algorithms, based on different deep reinforcement learning (DRL) approaches, value-based, policy-based, and hybrid approaches. In addition, to reduce the search space and computation complexity of the algorithms, we propose decomposition and function approximation techniques by leveraging the structure of the underlying problem. The evaluation results show that the proposed DRL-based task assignment schemes outperform the existing algorithms, and the hybrid actor-critic scheme performs the best under dynamic MEC network environments.

Index Tems—Mobile edge computing (MEC), edge server cooperation, task assignment, stochastic optimization, deep reinforcement learning

1 Introduction

Cloud computing has been widely adopted as a cost-effective platform to provide powerful computing capabilities and storage resources for computation-intensive applications. However, many emerging mobile applications such as intelligent transportation, smart cities, industrial robotics, and augmented/virtual reality (AR/VR) require not only intensive data processing and high network bandwidth but also very low latency. Cloud data centers are centralized and often located far away from the mobile users over the Internet. Thus, they are difficult to satisfy the low-latency requirements of these interactive applications and/or to adapt data processing to the local wireless network context. In addition, the proliferation of Internet of Things (IoT) devices generates a large amount of data at the network edge that needs to be efficiently handled and processed. It is highly inefficient to move large volumes of data collected from many edge sources, e.g. video sensors, to a centralized data center and perform remote computations. Mobile edge computing (MEC) is an emerging paradigm that provides computing services at the edge of the mobile radio access network (RAN), addressing the shortcomings of cloud data centers [1–6]. MEC edge servers, also called edge nodes, with computing, storage, and communication capabilities are co-located or integrated with base stations (BSs), routers, and gateways in a wireless access network, allowing the execution of applications in close proximity to mobile users. Mobile devices with excessive computing resources can also join the MEC network and offer services to other devices and applications. MEC can reduce transmission latency, alleviate network congestion, and provide real-time local context-aware services required by emerging mobile applications. It also allows network operators to provide additional value-added services and bring a better quality of experience to mobile users. MEC technology has attracted a lot of attentions in academia and industry, and it is considered as an important component of next-generation (5G and beyond) mobile networks. The standardization effort on the MEC technology is ongoing in the European Telecommunications Standards Institute (ETSI) [7] and 3GPP [8]. An industry consortium has also been created to promote open edge computing technology [40].

In spite of the fact that MEC can address the drawbacks of cloud computing, it is challenging to manage edge servers due to geographically distributed deployment of these edge servers and their heterogeneous computing resources [9, 10]. Unlike cloud computing, user requests for MEC computational tasks may arrive at any edge server, instead of a gateway or a master node. The computational tasks may be queued with a long delay because of insufficient processing resources at the edge server, and bounded buffer sizes may cause task queue overflows. Furthermore, the workload received by edge servers exhibits temporal and spatial fluctuations due to user mobility, the bursty nature of mobile applications, and unexpected events, e.g., a traffic accident. The limited resources of individual edge servers can be over

L. Hsieh and H. Liu are with the Department of Electrical Engineering and Computer Science, the Catholic University of America, Washington, DC 20064, USA.
Y. Guo is with the National Institute of Standards and Technology, Gaithersburg, MD 20878, USA.
R. Gazda is with InterDigital Communications, Inc., Conshohocken, PA 19428, USA.
This work is partially supported by the National Science Foundation under Grants CNS-1910348 and CNS-1822087, and InterDigital Communications, Inc.

Authorized licensed use limited to: NIST Virtual Library (NVL). Downloaded on August 25,2023 at 18:48:25 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3270242

or under-utilized over time. Based on these observations, a cooperative MEC network can be considered to tackle the problems, where edge servers without enough resources can forward their partial tasks to other nearby edge servers and/or remote cloud data centers for execution. By exploiting the horizontal cooperation among geographically distributed heterogeneous edge servers as well as the vertical cooperation between edge servers and cloud data centers to jointly process computational tasks, MEC system performance can be significantly improved. However, there are non-trivial challenges to assign the tasks to be executed at different edge servers and the cloud for a cooperative MEC network to achieve optimal performance: a) the edge-to-edge horizontal cooperation and the edge-to-cloud vertical cooperation should be considered. A framework is needed to model the complex interactions and heterogeneity of the involved entities as well as the shipping cost of the tasks from multiple edge servers to multiple edge servers or the cloud. b) The fluctuation in computing demands as well as the computation resource availability at different edge servers and network communication delay between the servers should be taken into consideration. c) The user task arrivals, available computing resources, and network conditions are non-stationary and unknown beforehand in many MEC scenarios. Thus, a stochastic framework, instead of a deterministic one, is necessary in order to fully capture the underlying dynamics and explore the synergy among the MEC entities for the optimal performance of joint task processing.

Although considerable work has been done to design MEC systems and algorithms, most research efforts have focused on the problem of offloading tasks from mobile devices to edge servers [11-15] or the vertical cooperation where MEC edge servers help cloud data centers process delay-sensitive user tasks for improved quality of service (QoS) [16-19]. Less research attention has been given to investigate the horizontal cooperation among MEC edge servers for joint task processing. The authors in [41] proposed a scheme that allows an edge server to forward its tasks to other edge servers to balance the workload. However, they made many assumptions in assigning the tasks to the edge servers, such as a fixed task arrival rate at each edge server as well as a pre-known task processing time of each edge server and pre-known transmission delay between the edge servers. Their task assignment algorithm utilizes the classical convex optimization method based on these assumptions under a static MEC environment. Such existing schemes are too idealized for real deployment scenarios and fail to characterize system dynamics and impacts of the performance.

In this paper, we investigate the task assignment and scheduling for cooperative mobile edge computing networks under varying task arrival statistics, node computing capabilities, and network states. We cast the task assignment as a dynamic and stochastic optimization problem and develop new deep reinforcement learning (DRL)-based algorithms which are able to dynamically assign the tasks requested without presuming the state of the network and the ability of the servers to be known. The assignment considers edge-to-edge cooperation and edge-to-cloud cooperation, and takes the heterogeneity of edge servers and the fluctuation of the network into account. Our contributions include,

(1) A novel stochastic framework is proposed to model the horizontal cooperation of edge servers as well as the vertical cooperation between edge servers and cloud data centers, in which the time-varying computation resources and network communication delays are considered. The stochastic task assignment problem is formulated as a Markov decision process (MDP).
(2) We derive and compare three new task assignment algorithms by seeking different deep reinforcement learning (DRL) approaches (value-based, policy-based, and hybrid), which can learn the optimal policy for the task assignment matrix to the edge servers and cloud data center without requirement for prior knowledge of task arrival statistics, node computation capabilities, and network dynamics.
(3) In order to deal with the state/action explosion and to improve the learning algorithm efficiency, we introduce decomposition and function approximation techniques by leveraging the structure of the underlying problem.
(4) Numerical results show that all three of our proposed online DRL-based task assignment schemes improve the MEC QoS performance, compared to the existing representative algorithms, and the hybrid scheme achieves the best performance under dynamic MEC environments.

To the best knowledge of the authors, this is the first work to solve the task assignment optimization problem with edge-to-edge horizontal cooperation and edge-to-cloud vertical cooperation under stochastic and dynamic MEC network environments by employing a deep reinforcement learning approach.

The remainder of the paper is organized as follows: Section 2 describes the system model. Section 3 formulates the problem of stochastic task assignment optimization. In Section 4, we simplify the problem and derive the deep reinforcement learning-based algorithms in detail. Section 5 provides the numerical experimental results under various settings. Section 6 reviews the related work. Finally, the conclusions are given in Section 7.

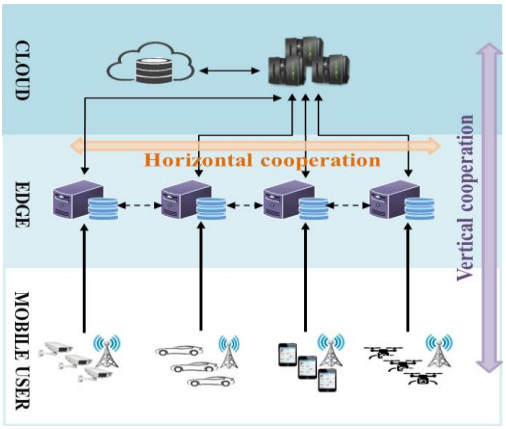

Fig. 1. System model.

2 System Model

Fig. 1 illustrates the system model under consideration in this paper. A MEC network consists of geographically distributed edge servers (note that we use edge servers and edge nodes interchangeably in the paper), deployed in a radio access network (RAN) covering a certain area. The edge servers are equipped with computing resources and are co-located or integrated with base stations or Wi-Fi access points. They receive the computing tasks from their associated mobile users over the wireless network. These edge servers connect to the cloud data centers through Internet. In this paper, we model a data center as a special node with powerful resources but far away from a RAN in terms of network transmission

Authorized licensed use limited to: NIST Virtual Library (NVL). Downloaded on August 25,2023 at 18:48:25 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3270242

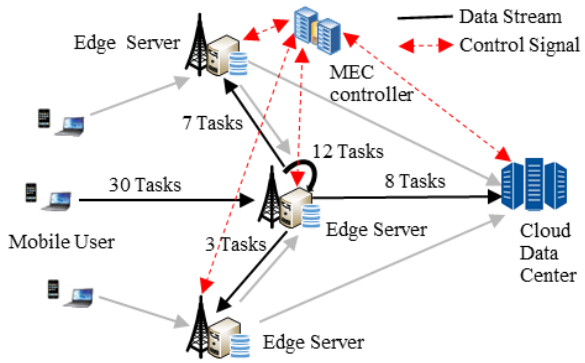

Fig. 2. An example of task processing and edge server cooperation.

distance and consider it as an extension of the MEC network. We use the term "MEC system" or "MEC network" to refer to the networked system that includes the MEC edge servers and remote cloud data centers, unless otherwise stated. Note that some user devices may, opportunistically, become MEC edge servers and be part of a MEC system by contributing their computing resources to help execute the tasks.

Mobile users/smart devices/sensors connect to nearby MEC edge servers to submit their computational tasks to be processed. A user task is a computational job request along with associated data, for example, recognizing an image captured by a mobile device or analyzing the data captured by a sensor. The MEC nodes (edge servers and a cloud data center) help each other to jointly process the computational tasks. As an example, shown in Fig. 2, an edge server receives the tasks from its associated devices, it may process them locally, or forward part or all of its unprocessed tasks to other edge servers and/or the cloud data center for processing to optimize the QoS, which is based on the task assignment decision. We consider a software-defined MEC network with a centralized control plane and a distributed data plane [21, 22]. Software-defined networks (SDNs) have attracted a lot of interest from network service providers because they can be flexibly controlled and programmed. In an SDN-based MEC network, a control plane connects the edge servers to a software-defined programmable MEC controller that coordinates the task assignment decisions by taking into consideration the dynamic workload and network conditions. The edge servers will forward, receive, and execute the tasks on the data plane based on the task assignment decisions received on the control plane from the controller. The MEC controller resides in the RAN and could be one of the edge servers with dedicated control plane connectivity, thus the control latency is minimal.

It is assumed that the computational tasks from the associated devices arrive at MEC edge servers randomly, and the distribution is not known beforehand. The network delay between two nodes is time-varying and unknown in advance due to dynamic network conditions, traffic load, and many other uncertain factors. In addition, the task processing capability of a node is also time-varying because the CPU cycles may be adjusted based on the environments such as heat and power status, and the task complexity varies.

Consider a MEC network that consists of N edge servers in the area of consideration, labeled as $\mathcal{N} = \{1, 2, \ldots, N\}$ and a remote cloud data center modeled as a special node n_c that has very powerful capability to process the tasks but incurs a high network delay. Note that this model can be easily extended to multiple data centers. We assume that the system operates over discrete scheduling slots of equal time duration, and the task scheduling decision is performed every time slot. The values of a task assignment matrix $\boldsymbol{\Phi}^t = [\phi_{n,j}^t : n, j \in \mathcal{N} \cup n_c\}$ should be determined at the beginning of each time slot t, where $\phi_{n,j}^t$ specifies the number of tasks that edge server n will send to edge server j or cloud data center n_c for processing in slot t, and $\phi_{n,n}^t$ is the number of tasks that edge server n will buffer for processing by itself. $\boldsymbol{\phi}_n^t = [\phi_{n,j}^t, \phi_{j,n}^t : j \in \mathcal{N} \cup n_c\}$ represents the task assignment vector regarding edge server n. An edge server may either have extra computing resources to help other nodes to process the tasks or may need to forward its tasks to other nodes for processing due to overload. We assume that the data center n_c will process all the received tasks by itself, not forwarding them to the edge servers, i.e., $\phi_{n_c,j}^t = 0, j \in \mathcal{N}$. We also assume that there is no dynamic joining or quitting of the edge servers, whose impacts will be part of our future study. For convenience, Table I summarizes the major notations used in this paper.

TABLE I. LIST OF MAJOR NOTATIONS

Symbol	Definition
$\mathcal{N}$	The set of edge servers
n_c	Cloud data center
t	Time slot
$\phi_{n,j}^t$	# of tasks that node n will forward to node j in slot t
$\boldsymbol{\Phi}^t$	Task assignment matrix, $\boldsymbol{\Phi}^t = [\phi_{n,j}^t : n, j \in \mathcal{N} \cup n_c\}$
$\boldsymbol{\phi}_n^t$	Task assignment vector for node n, $\phi_n^t = [\phi_{n,j}^t, \phi_{j,n}^t : j \in \mathcal{N} \cup n_c\}$
$\mathcal{p}_{n,j}^t$	Probability that node n will forward its tasks to node j in slot t
$\boldsymbol{\mathcal{P}}^t$	Task assignment probability matrix, $\boldsymbol{\mathcal{P}}^t = [\mathcal{p}_{n,j}^t : n, j \in \mathcal{N} \cup n_c\}$
$\boldsymbol{\mathcal{p}}_n^t$	Task assignment vector for node n, $\boldsymbol{\mathcal{p}}_n^t = [\mathcal{p}_{n,j}^t, \mathcal{p}_{j,n}^t : j \in \mathcal{N} \cup n_c\}$
A_n^t	Task arrivals of node n in slot t
$\boldsymbol{A}^t$	Task arrival matrix, $\boldsymbol{A}^t = \{A_n^t : n \in \mathcal{N}\}$
q_n^t	Queue size of node n at the beginning of slot t
$\boldsymbol{q}^t$	Queue state, $\boldsymbol{q}^t = \{q_n^t : n \in \mathcal{N} \cup n_c\}$,
$q_n^{(max)}$	Maximum queue length of node n
s_n^t	Task processing capability of node n (max number of tasks that node n can process) in slot t
$\boldsymbol{S}^t$	Task processing capability vector, $\boldsymbol{S}^t = \{s_n^t : n \in \mathcal{N} \cup n_c\}$
$c_{n,j}^t$	Network communication delay for shipping a task from node n to node j.
$\boldsymbol{c}_n^t$	Network communication delay vector for node n, $\boldsymbol{c}_n^t = (c_{n,j}^t, c_{j,n}^t : j \in \mathcal{N} \cup n_c)$
$\boldsymbol{C}^t$	Network communication delay matrix, $\boldsymbol{C}^t = \{\boldsymbol{c}_n^t : n \in \mathcal{N} \cup n_c\}$
$\boldsymbol{\chi}_n^t$	Local network state of node n, $\boldsymbol{\chi}_n^t = (Q_n^t, s_{n,}^t \boldsymbol{c}_n^t)$ at the beginning of time slot t
$\boldsymbol{\chi}^t$	Global MEC network state, $\boldsymbol{\chi}^t = (\boldsymbol{\chi}_n^t : n \in \mathcal{N} \cup n_c) = (\boldsymbol{Q}^t, \boldsymbol{S}^t, \boldsymbol{C}^t)$
d_n	Service delay of node n, including network delay and queuing delay
$d^{(\max)}$	Maximum tolerance threshold of the service delay
o_n	Task queue overflow rate (number of tasks over the maximum queue size during a time slot) for node n
$o^{(\max)}$	Maximum tolerance threshold of the task queue overflow rate

Authorized licensed use limited to: NIST Virtual Library (NVL). Downloaded on August 25,2023 at 18:48:25 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3270242

Symbol	Definition
$U_n^{(d)}(\cdot)$	Quality satisfaction related to service delay for node *n*
$U_n^{(o)}(\cdot)$	Quality satisfaction related to task queue overflow rate for node *n*
$U(\cdot)$	Utility function related to service delay and task queue overflow rate.
w_d, w_o	Weight factors in the utility function related to service delay and task queue overflow rate, respectively
$V(\boldsymbol{\chi}, \boldsymbol{\Phi})$	Expected discounted long-term utility (state value function)
γ, α	Parameters in state value function, γ is a discount factor, and $\alpha = (1-\gamma)$
$\boldsymbol{\Phi}^*$	Optimal task assignment policy
$V^*(\boldsymbol{\chi})$	Optimal state value function
$\tilde{Q}_n, \tilde{s}_n, \tilde{\boldsymbol{c}}_n$	Post-decision states of queue, task processing capability, and network delay, respectively, for node *n*
$\tilde{\boldsymbol{Q}}, \tilde{\boldsymbol{S}}, \tilde{\boldsymbol{C}}$	Post-decision states of queue, task processing capability, and network delay, respectively, for MEC network
$\tilde{\boldsymbol{\chi}}$	Post-decision state of MEC network, $\tilde{\boldsymbol{\chi}} = (\tilde{\boldsymbol{Q}}, \tilde{\boldsymbol{S}}, \tilde{\boldsymbol{C}})$
$\tilde{V}^*(\tilde{\boldsymbol{\chi}})$	Optimal post-decision state value function
ε^t	Learning rate

3 Problem Formulation

In this section, we first formulate the problem of stochastic task assignment optimization and then discuss the approaches to solve the optimization problem. Let A_n^t be the number of the new tasks randomly arrived at edge server *n*, $n \in \mathcal{N}$ from its associated users in time slot *t*, and $\boldsymbol{A^t} = \{A_n^t : n \in \mathcal{N}\}$. The distribution of A_n^t is not known beforehand. Q_n^t represents the task queue length of node *n* at the beginning of time slot *t*. Let s_n^t be the task processing capability of node *n* in slot *t*, which is defined as the number of tasks that node *n* can serve/process in slot *t*. Different tasks may consume different amount of resource such as CPU clock cycle. In other words, a node may have different rates to process different types of tasks. A node may receive different type of tasks. Thus, s_n^t varies in time and is unknown in advance. We modeled s_n^t as a Markov model with multiple states. Each state represents a type of tasks. In this way, a node does not have to know what types of tasks will arrive at it. It can learn its process capability for the type of the tasks arrived based on the Markov model. The queue evolution of node *n* can be written as $q_n^{t+1} = \max\{0, \min[q_n^t + A_n^t + \sum_{i\in e_n}(\phi_{i,n}^t - \phi_{n,i}^t) - s_n^t, Q_n^{(max)}]\}$, where $\sum_{i\in e_n}\phi_{n,i}^t$ with $e_n = \{\mathcal{N} \cup n_c\}\backslash\{n\}$ represents the number of tasks that edge server *n* offloads to other edge servers and the cloud, and $\sum_{i\in e_n}\phi_{i,n}^t$ is the number of tasks that edge server *n* receives from other edge servers in slot *t*. $Q_n^{(max)}$ is the maximum buffer size at node *n*.

The local state of a node is characterized by its task queue size, its task processing capability, and its network delay to other nodes. For a node *n*, $n \in \mathcal{N} \cup n_c$, at the beginning of time slot *t*, we measure its local state as $\boldsymbol{\chi}_n^t = (Q_n^t, s_{n,}^t \boldsymbol{c}_n^t)$ where $\boldsymbol{c}_n^t = (c_{n,j}^t, c_{j,n}^t : j \in \mathcal{N} \cup n_c)$ with $c_{n,j}^t$ being the network delay for shipping a task from node *n* to node *j*, $c_{j,n}^t$ being to the network delay for shipping a task from node *j* to node *n*, and $c_{n,n}^t = 0$. As the network delay between two nodes is related to the network bandwidth, transmission distance (the number of hops along the path between the two nodes), traffic conditions in the network, and many other unpredicted factors, it varies in time and its distribution is unknown as well. Thus, at the beginning of each scheduling time slot *t*, the global MEC network state is represented $\boldsymbol{\chi^t} = (\boldsymbol{\chi}_n^t : n \in \mathcal{N} \cup n_c) = (\boldsymbol{Q^t}, \boldsymbol{S^t}, \boldsymbol{C^t}) \in X$, where $\boldsymbol{Q^t} = \{Q_n^t : n \in \mathcal{N} \cup n_c\}$, $\boldsymbol{S^t} = \{s_n^t : n \in \mathcal{N} \cup n_c\}$, and $\boldsymbol{C^t} = \{\boldsymbol{c}_n^t : n \in \mathcal{N} \cup n_c\}$. X represents the whole MEC system state space. Note that we model the cloud data center n_c as a special node and its state can then be represented in the same way as an edge server, except that its task processing capability $s_{n_c}^t$ is assumed to be large.

For a given MEC network state $\boldsymbol{\chi^t}$ at the beginning of a time slot *t*, a task assignment $\boldsymbol{\Phi}^t = \boldsymbol{\Phi}(\boldsymbol{\chi^t}) = \{\phi_{n,j}(\boldsymbol{\chi^t}) : n, j \in \mathcal{N} \cup n_c\}$ is made, and the MEC network achieves an instantaneous utility that is related to the QoS. We consider delay-sensitive applications, where the QoS is measured by the task service delay and the task queue overflow rate. The task service delay is defined as the period from the time that a task arrives at an edge server to the time that the task has been served in the unit of scheduling slot duration. For an edge server *n*, $n \in \mathcal{N}$, the service delay d_n depends on the delay incurred by the queue Q_n if the edge server *n* processes the task locally or consists of the network delay $c_{n,j}^t$ and the queueing delay due to the queue Q_j at the service provider *j* if a task is sent from node *n* to node *j* for processing. The task queue overflow rate o_n is defined as the number of tasks overflowed per time slot due to the limited buffer size.

The instantaneous MEC network utility under the state $\boldsymbol{\chi^t}$ and task assignment decision $\boldsymbol{\Phi}(\boldsymbol{\chi^t})$ at time slot *t* is defined as,

$$U(\boldsymbol{\chi^t}, \boldsymbol{\Phi}(\boldsymbol{\chi^t})) = \sum_{n\in\mathcal{N}} [w_d U_n^{(d)}(\boldsymbol{\chi^t}, \boldsymbol{\Phi}(\boldsymbol{\chi^t})) + w_o U_n^{(o)}(\boldsymbol{\chi^t}, \boldsymbol{\Phi}(\boldsymbol{\chi^t}))], \quad (1)$$

where $U_n^{(d)}(.)$ and $U_n^{(o)}(.)$ measure the satisfactions of the service delay and task queue overflow rate, respectively. w_d and w_o are the weight factors indicating the importance of delay and task queue overflow in the utility function of the MEC system, respectively. For an edge server, we consider there is a maximal tolerance threshold, $d^{(\max)}$ for the service delay, i.e. $d_n \le d^{(\max)}$. Correspondingly, let $o^{(\max)}$ be the maximal tolerance threshold for the task queue overflow rate, i.e. $o_n \le o^{(\max)}$. In addition, we choose the utility function to be the exponential functions, namely $U_n^{(d)} = \exp(-d_n/d^{(\max)})$ and $U_n^{(o)} = \exp(-o_n/o^{(\max)})$ [23, 38].

Stochastic user task arrivals and dynamic MEC system states present challenges and make naive one-shot optimization schemes unstable and unable to achieve the optimal network performance on a longer timescale. Therefore, we want to develop a stochastic optimization framework for the cooperative task assignment, which maximizes the expected long-term utility of a MEC system while guaranteeing the service delay and task queue overflow rate are within their respective acceptable thresholds.

The task assignment matrix $\boldsymbol{\Phi}(\boldsymbol{\chi^t})$ is determined according to the control policy $\boldsymbol{\Phi}$ after observing the network state $\boldsymbol{\chi^t}$ at the beginning of a time slot *t*, which induces a probability distribution over the set of possible global MEC network states in the following time slot $\boldsymbol{\chi^{t+1}}$, and hence a probability distribution over the set of per-slot utility $U(\boldsymbol{\chi^t}, \boldsymbol{\Phi}(\boldsymbol{\chi^t}))$. For simplicity, we assume that the task

Authorized licensed use limited to: NIST Virtual Library (NVL). Downloaded on August 25,2023 at 18:48:25 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3270242

processing capability of a node and the network delay each can be modelled as a finite-state discrete-time Markov chain across the time slots, i.e. the probability of a state in the subsequent slot depends only on the state attained in the present slot. Given a control policy $\boldsymbol{\Phi}$, the random process $\boldsymbol{\chi^t}$ is thus a controlled Markov chain [24, 25] with the following state transition probability,

$$\Pr\{\boldsymbol{\chi^{t+1}}|\boldsymbol{\chi^t}, \boldsymbol{\Phi}(\boldsymbol{\chi^t})\} = \Pr\{\boldsymbol{Q^{t+1}}|\boldsymbol{\chi^t}, \boldsymbol{\Phi}(\boldsymbol{\chi^t})\} \Pr\{\boldsymbol{S^{t+1}}|\boldsymbol{S^t}\} \Pr\{\boldsymbol{C^{t+1}}|\boldsymbol{C^t}\}, \quad (2)$$

where $\Pr\{.\}$ denotes the probability of an event. We assume that the task processing capability and the network communication delay are independent. For a controlled Markov chain, the transition probability from a present state $\boldsymbol{\chi^t}$ to the next state $\boldsymbol{\chi^{t+1}}$ depends only on the present state $\boldsymbol{\chi^t}$ and the control policy $\boldsymbol{\Phi}(\boldsymbol{\chi^t})$ acted on the present state. Take discounted expectation with respect to the per-slot utilities $U(\boldsymbol{\chi^t}, \boldsymbol{\Phi}(\boldsymbol{\chi^t}))$ over a sequence of network states $\boldsymbol{\chi^t}$, the discounted expected value of the long-term utility of a MEC network can be defined as [25, 26],

$$V(\boldsymbol{\chi}, \boldsymbol{\Phi}) = \mathrm{E}\left[\alpha \cdot \sum_{t=1}^{\infty} \gamma^{t-1} U(\boldsymbol{\chi^t}, \boldsymbol{\Phi}(\boldsymbol{\chi^t}))| \boldsymbol{\chi^1}\right], \quad (3)$$

where $\alpha, \gamma \in [0, 1)$ are the parameters. γ is a discount factor that discounts the utility rewards received in the future, and $(\gamma)^{t-1}$ denotes the discount to the (t −1)-th power. $\boldsymbol{\chi^1}$ is the initial network state. $V(\boldsymbol{\chi}, \boldsymbol{\Phi})$ is also termed as the state value function of the MEC network in state $\boldsymbol{\chi}$ under task assignment policy $\boldsymbol{\Phi}$. α is multiplied only for analysis convenience. We let $\alpha = 1 - \gamma$, the expected undiscounted long-term average utility, $\bar{U}(\boldsymbol{\chi}, \boldsymbol{\Phi}) = \mathrm{E}\left[\lim_{T\to\infty} \frac{1}{T} \cdot \sum_{t=1}^{\infty} \gamma^{t-1} U(\boldsymbol{\chi^t}, \boldsymbol{\Phi}(\boldsymbol{\chi^t}))| \boldsymbol{\chi^1}\right]$ can be considered as a special case of (3) when γ approaches 1 and $\alpha = (1 - \gamma)$ approaches 0 [25]. On the other hand, if γ is set to be 0, then $V(\boldsymbol{\chi}, \boldsymbol{\Phi}) = U(\boldsymbol{\chi^1}, \boldsymbol{\Phi}(\boldsymbol{\chi^1}))$, that is, only the immediate utility performance is considered. We therefore consider the expected discounted long-term utility performance in (3) as a general QoS indicator in this paper.

The objective is to design an optimal task assignment control policy $\boldsymbol{\Phi}^*$ that maximizes the expected discounted long-term utility performance, that is,

$$\boldsymbol{\Phi}^* = arg \max_{\boldsymbol{\Phi}} \big(V(\boldsymbol{\chi}, \boldsymbol{\Phi})\big) \quad (4)$$

$V^*(\boldsymbol{\chi}) = V(\boldsymbol{\chi}, \boldsymbol{\Phi}^*)$ is the optimal state value function. The stochastic task assignment optimization in (4) can be considered as a MDP with the discounted utility criterion as the network states follow a controlled Markov process [26]. The optimal task assignment control policy achieving the maximal state value function can thus be obtained by solving the following Bellman's optimality equation [26,27],

$$V^*(\boldsymbol{\chi}) = \max_{\boldsymbol{\Phi}} \Big\{(1-\gamma)\, U(\boldsymbol{\chi}, \boldsymbol{\Phi}(\boldsymbol{\chi})) + \gamma \sum_{\boldsymbol{\chi'}} \Pr\{\boldsymbol{\chi'}|\boldsymbol{\chi}, \boldsymbol{\Phi}(\boldsymbol{\chi})\}\, V^*(\boldsymbol{\chi'})\Big\}, \quad (5)$$

where $\boldsymbol{\chi'} = \{\boldsymbol{Q'}, \boldsymbol{S'}, \boldsymbol{C'}\}$ is the MEC network state in the subsequent time slot, and $\Pr\{\boldsymbol{\chi'}|\boldsymbol{\chi}, \boldsymbol{\Phi}(\boldsymbol{\chi})\}$ represents the state transition probability that making the task assignment $\boldsymbol{\Phi}(\boldsymbol{\chi})$ in state $\boldsymbol{\chi}$ will produce the next state $\boldsymbol{\chi'}$. $\boldsymbol{Q'} = \{Q'_n : n \in \mathcal{N} \cup n_c\}$, $\boldsymbol{S'} = \{s'_n : n \in \mathcal{N} \cup n_c\}$, and $\boldsymbol{C'} = \{\boldsymbol{c'}_n : n \in \mathcal{N} \cup n_c\}$ are the queue, task processing capability, and network delay states in the subsequent time slot.

Solving (5) is generally a challenging problem. Traditional approaches are based on value iteration, policy iteration, and dynamic programming [27, 28], but these methods require full knowledge of the network state transition probabilities and task arrival statistics, which for our problem, cannot be obtained in advance. Thus, we seek the online reinforcement learning techniques to solve the problem which does not have such requirements. Conventional Q-learning [29, 30] defines an evaluation function, called Q function, $Q(\boldsymbol{\chi}, \boldsymbol{\Phi}) = (1 - \gamma)\, U(\boldsymbol{\chi}, \boldsymbol{\Phi}) + \gamma \sum_{\boldsymbol{\chi'}} \Pr\{\boldsymbol{\chi'}|\boldsymbol{\chi}, \boldsymbol{\Phi}\}\, Q(\boldsymbol{\chi'}, \boldsymbol{\Phi})$ and learns an optimal state-action value table in a recursive way to decide the optimal task assignment control policy for each time slot. However, for the cooperative MEC network, the task assignment decision-making for a node depends on not only its own resource availability and queue state, but also is affected by the resource availability and queue states of other nodes as well as the network delay between the nodes. The system state space and control action space will grow rapidly as the number of involved nodes increases. The conventional Q-learning process will search and update a large state-action value table, which incurs high memory usage and computation complexity and cannot handle the large state space well. Deep reinforcement learning incorporates reinforcement learning with deep neural networks (DNNs) to address the state and action space explosion issues of the conventional Q-learning [29, 31]. We will design DRL-based task assignment algorithms below.

4 Problem Simplification and Deep Reinforcement Learning Algorithms

In this section, we focus on developing efficient algorithms to achieve the optimal task assignment policy with no assumption for prior knowledge of the statistical information about the network state transitions and task arrivals based on recent advances in deep reinforcement learning. There are three categories of the state-of-the-art DRL techniques, including value-based, policy-based, and hybrid approaches. Specifically, we design three sets of novel algorithms leveraging the underlying structure of our task assignment optimization problem and the three DRL approaches to tackle the aforementioned challenges and to maximize the long-term MEC network utility. We will evaluate the advantages and disadvantages of the designed algorithms and gain insights to each DRL approach for our problem in the next section. In addition, it can be observed that the MEC network utility function is additive, which motivates us to linearly decompose the state value function, and incorporate the decomposition technique into the DRL-based algorithms to lower the complexity.

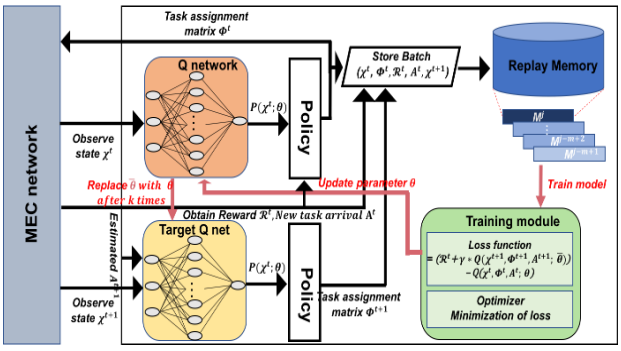

Fig. 3. DDQN-based cooperative MEC task assignment.

Authorized licensed use limited to: NIST Virtual Library (NVL). Downloaded on August 25,2023 at 18:48:25 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3270242

4.1 Value-based DRL Task Assignment Algorithm

For the value-based DRL approach, an agent learns the state value function, and the optimal policy is determined according to the learned value function. Double deep Q networks (double DQN or DDQN) utilize double Q-learning with two deep neural networks [32, 33], which can reduce the overestimation errors of traditional Q-learning by separating the action selection and action evaluation. Unfortunately, the conventional DDQN algorithms cannot be directly applied to solve our problem because it outputs the Q values corresponding to the state-action pairs and selects the action with the maximum Q value that depends on the total tasks received in the current time slot. However, we do not know the number of the new task arrivals in a time slot at the beginning of the time slot. To solve the problem, we modified the standard DDQN to output a probability matrix, which indicates the probabilities that an edge server forwards its tasks to other edge servers for processing in a time slot, i.e., the portion of the tasks shipped from one edge server to the others in the time slot. We further truncate the probabilities to a discrete set of values to reduce the action space and simplify the learning model.

The modified DDQN is used to approach the optimal state value function in (5) and select the best action. Fig. 3 illustrates the proposed DDQN-based reinforcement learning scheme for the collaborative MEC task assignment. The system consists of two DNNs, Q evaluation network (Q-eval) and Q target network (Q-tar), to learn the optimal state value function and decide the optimal action. The Q-eval is used to select the task assignment matrix $\boldsymbol{\Phi^t}(\boldsymbol{\chi^t}, \theta)$ based on the collected network states $\boldsymbol{\chi^t}$ at the time slot *t*, and the Q-tar is used to estimate the value of the current task assignment policy and decide the target task assignment matrix $\bar{\boldsymbol{\Phi}}^{\boldsymbol{t+1}}(\boldsymbol{\chi^{t+1}}, \bar{\theta})$ for the following scheduling slot. The parameters θ and $\bar{\theta}$ are updated iteratively. We redefine the state value function (5) to be (6) as shown at the bottom of this page, where $\boldsymbol{\mathcal{P}}(\boldsymbol{\chi^t}, \theta^t)$ and $\boldsymbol{\mathcal{P}'}(\boldsymbol{\chi^{t+1}}, \bar{\theta}^t)$ are the probability matrices calculated by Q-eval and Q-tar networks, respectively. In the conventional DDQN algorithm, the state value will be updated in each time slot and used to determine the optimal action. To simplify the updates, in our implementation, the state value obtained from (6) is stored in a replay memory for training and updating θ and $\bar{\theta}$ in the learning process so that the Q-eval and Q-tar can select the optimal task assignment probability matrices directly and accurately. The loss function for updating the parameters θ of Q-eval is defined in (7) at the bottom of this page. The parameters $\bar{\theta}$ will be updated by copying θ after a predefined number of steps.

Specifically, at the beginning of each time slot *t*, the MEC controller determines the task assignment matrix $\boldsymbol{\Phi^t}(\boldsymbol{\chi^t})$ based on the observed network states and informs the edge servers of the task assignment decision. The task assignment matrix $\boldsymbol{\Phi}^t = [\phi^t_{n,j}: n, j \in \mathcal{N} \cup n_c\}$ at the beginning of scheduling slot *t* is determined as,

$$\boldsymbol{\Phi}^t = \boldsymbol{\mathcal{P}}^t(\boldsymbol{\chi^t}; \theta^t) \tag{8}$$

An edge server then offloads the tasks to other nodes or receives tasks from other nodes and processes these tasks based on the task assignment decision. The new task arrivals $\boldsymbol{A^t}$ will be counted at the end of the time slot *t* and the new network state is collected and updated to $\boldsymbol{\chi^{t+1}}$ by the controller. The MEC network receives a utility $\boldsymbol{U}^t = U(\boldsymbol{\chi^t}, \boldsymbol{\Phi^t}(\boldsymbol{\chi^t}, \boldsymbol{\mathcal{P}}(\boldsymbol{\chi^t}; \theta^t)))$ by performing the task processing. The Q-tar network is used to calculate $\boldsymbol{\Phi^{t+1}}$. As shown in Fig. 5, the DDQN includes a replay memory that is used to store a pool of the most recent *M* transition experiences, $\Omega = \{\boldsymbol{m}^{t-M+1}, \dots, \boldsymbol{m}^t\}$, where each experience $\boldsymbol{m^t} = (\boldsymbol{\chi^t}, \boldsymbol{\Phi^t}, \boldsymbol{U^t}, \boldsymbol{\chi^{t+1}}, \boldsymbol{\Phi^{t+1}})$ is occurred at the transition of two consecutive slots *t* and *t* + 1 during the learning process. At a slot *t*, the *k* previous experiences are randomly sampled as a batch from the memory pool Ω to train the DDQN online. The approximated overall state value for each experience in the batch is calculated and the parameters θ is updated with a goal to minimize the loss function (7). Once the state value function is converged, we can obtain the optimal parameters θ^* for Q-eval. The optimal policy will thus be,

$$\boldsymbol{\Phi}^* = \boldsymbol{\mathcal{P}}^*(\boldsymbol{\chi}; \theta^*) \tag{9}$$

The MEC network utility in (1) is the summation of the service delay and task queue overflow rate satisfactions of the edge servers. The task arrival statistics and task processing capabilities of the edge servers are independent each other. We can then decompose (6) into per server utility and separate the satisfactions regarding the service delay and the task queue overflows [34]. We first rewrite (8) as

$$\boldsymbol{\Phi}^t = \{\boldsymbol{\phi}^t_n(\boldsymbol{\chi}^t_n): n \in \mathcal{N}\} = \{\boldsymbol{\mathcal{P}}^t_n(\boldsymbol{\chi^t_n}; \theta^t_n): n \in \mathcal{N}\}. \tag{10}$$

where $\boldsymbol{\mathcal{P}}_n(.)$ is the task assignment probability related to server n. n agents $n \in \mathcal{N}$ can be employed and each agent learns the respective optimal state value function through a per server sub-DDQN. The optimal joint task assignment decision is thus made to maximize the aggregated state value function from all the agents. The state value function in (6) can be decomposed and expressed as in (11) and (12)

$$V^t(\boldsymbol{\chi^t}) = \sum_{n \in \mathcal{N}} V^t_n(q^t_n, s^t_n, \boldsymbol{c}^t_n), \tag{11}$$

$$\begin{aligned} V^t_n(\boldsymbol{\chi}^t_n) = {} & (1-\gamma^t) U\left(\boldsymbol{\chi}^t_n, \boldsymbol{\Phi}^t\left(\boldsymbol{\chi}^t_n, \boldsymbol{\mathcal{P}}_n(\boldsymbol{\chi}^t_n; \theta^t_n)\right)\right) + \\ & \gamma^t [\Pr\{\boldsymbol{\chi}^{t+1}_n | \boldsymbol{\chi}^t_n, \boldsymbol{\Phi}^t(\boldsymbol{\chi}^t_n, \boldsymbol{\mathcal{P}}_n(\boldsymbol{\chi}^t_n; \theta^t_n))\} \\ & U\left(\boldsymbol{\chi}^{t+1}_n, \Phi^{t+1}(\boldsymbol{\chi}^{t+1}_n, \boldsymbol{\mathcal{P}}'_n\left(\boldsymbol{\chi}^{t+1}_n; \bar{\theta}^t_n\right)\right)\right)] \end{aligned} \tag{12}$$

With the linear decomposition, the problem to solve a complex Bellman's optimality equation (6) is broken into simpler MDPs and the computation complexity is lowered. In order to derive a task assignment policy based on the global MEC network state, $\boldsymbol{\chi} = (\boldsymbol{\chi}_n: n \in \mathcal{N} \cup n_c)$ with $\boldsymbol{\chi}_n = (q_n, s_n, \boldsymbol{c}_n)$ and $\boldsymbol{c}_n = \left(c_{n,j}, c_{j,n} : j \in \mathcal{N} \cup n_c\right)$, at least $\prod_{n \in \mathcal{N} \cup n_c} \prod_{j \in \mathcal{N} \cup n_c} (|q_n||s_n||c_{n,j}||c_{j,n}|)$ states should be trained. Using linear decomposition, only $(N + 1)\,|q_n||s_n|\,\prod_{j \in \mathcal{N} \cup n_c}\left(\left|c_{n,j}\right|\left|c_{j,n}\right|\right)$ states need to be trained, significantly reducing the number of training states and resulting in much simplified task assignment decision

$$V^t(\boldsymbol{\chi^t}) = \max_{\boldsymbol{\Phi}} \{(1-\gamma^t) U(\boldsymbol{\chi^t}, \boldsymbol{\Phi^t}(\boldsymbol{\chi^t}, \boldsymbol{\mathcal{P}}(\boldsymbol{\chi^t}; \theta^t))) + \gamma^t [\Pr\{\boldsymbol{\chi^{t+1}} | \boldsymbol{\chi^t}, \boldsymbol{\Phi^t}(\boldsymbol{\chi^t}, \boldsymbol{\mathcal{P}}(\boldsymbol{\chi^t}; \theta^t))\} U\left(\boldsymbol{\chi^{t+1}}, \boldsymbol{\Phi^{t+1}}(\boldsymbol{\chi^{t+1}}, \boldsymbol{\mathcal{P}'}\left(\boldsymbol{\chi^{t+1}}; \bar{\theta}^t\right))\right)]\} \tag{6}$$

$$\mathbb{L}(\theta) = E\left[\left((1-\gamma) U(\boldsymbol{\chi}, \boldsymbol{\Phi}(\boldsymbol{\chi}, \boldsymbol{\mathcal{P}}(\boldsymbol{\chi}; \theta))) + \gamma [U(\boldsymbol{\chi}', \boldsymbol{\Phi}'(\boldsymbol{\chi}', \boldsymbol{\mathcal{P}}'(\boldsymbol{\chi}'; \bar{\theta})))] - V(\boldsymbol{\chi})\right)^2\right] \tag{7}$$

Authorized licensed use limited to: NIST Virtual Library (NVL). Downloaded on August 25,2023 at 18:48:25 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3270242

makings especially when the number of MEC servers $\mathcal{N}$ is large. The online DDQN-based algorithm to estimate the optimal state value function and determine the optimal task assignment policy is summarized in Algorithm 1.

Algorithm 1. Online DDQN-based Cooperative MEC Task Assignment

1. Initialize the Q-eval and Q-tar with two sets of θ^t and $\bar{\theta}^t$ random parameters for t = 1; allocate the replay memory Ω for experience replay.
2. At the beginning of scheduling slot t, the MEC controller observes the network state, $\boldsymbol{\chi}^t = \{\boldsymbol{\chi}_n^t : n \in \mathcal{N}\}$ with $\boldsymbol{\chi}_n^t = (q_n^t, s_n^t, \boldsymbol{c}_n^t)$, and the Q-eval with parameters θ^t, and then determines the task assignment matrix, $\boldsymbol{\Phi}^t = [\boldsymbol{\phi}_n^t : n \in \mathcal{N}]$.
3. The edge servers offload and process the tasks according to the above task assignment decision, and the new tasks $\boldsymbol{A}^t = \{A_n^t : n \in \mathcal{N}\}$ will be counted at the end of slot t.
4. The controller determines the MEC network utility U^t and calculates the state value V^t according to (11) and (12)
5. The network state transits to $\boldsymbol{\chi}^{t+1} = \{\boldsymbol{\chi}_n^{t+1} : n \in \mathcal{N}\}$ where $\boldsymbol{\chi}_n^{t+1} = (q_n^t + A_n^t, s_n^{t+1}, \boldsymbol{c}_n^{t+1})$, which is taken as input to the Q-tar with parameter $\bar{\theta}^t$ to estimate the value of the current task assignment policy and decide the target task assignment matrix $\bar{\boldsymbol{\Phi}}^{t+1} = \{\bar{\boldsymbol{\phi}}_n^{t+1}, n \in \mathcal{N}\}$ for the following scheduling slot t+1.
6. The replay memory Ω is updated with most recent transition $\boldsymbol{m}^t(\boldsymbol{\chi}^t, \boldsymbol{\Phi}^t, \boldsymbol{U}^t, \bar{\boldsymbol{\Phi}}^{t+1}, \boldsymbol{\chi}^{t+1})$.
7. Once the replay memory collects $\overline{M}$ transitions, the controller updates the Q-eval parameter θ^t with a randomly sampled batch of transitions to minimize the loss function (7).
8. The target DQN parameter $\bar{\theta}^t$ are reset every k time slots, and otherwise $\bar{\theta}^t = \bar{\theta}^{t-1}$
9. The scheduling slot index is updated by $t \leftarrow t + 1$.
10. Repeat from step 2 to 9.

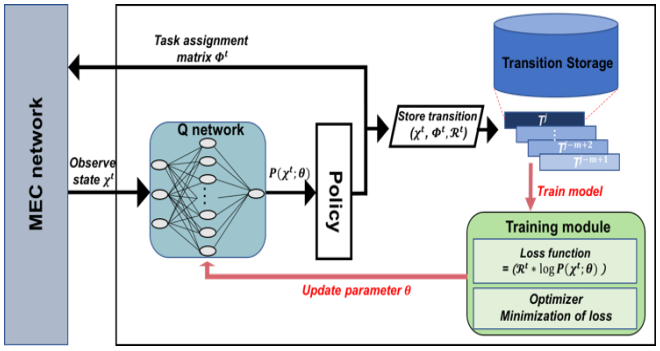

Fig. 4. Policy-Gradient-based cooperative MEC task assignment.

4.2 Policy-based DRL Task Assignment Algorithm

For the value-based DRL, it obtains the optimal policy and make the action decision by calculating and solving the state value function with the maximal Q-value. This approach is not well applicable to problems with continuous or large action space [35]. On the other hand, policy-based DRL algorithms are designed to tackle this issue by searching directly in the action space. In this section, we investigate the feasibility of solving the task assignment problem by designing a policy-based DRL algorithm, which uses a DNN to directly approximate the optimal policy and makes the decision. Motivated by [36], we adopt a stochastic policy gradient (PG) method to design our policy-based DRL task assignment algorithm, which finds the optimal policy by adjusting the possibility of trajectory with the highest accumulative rewards of an episode.

Fig. 4 depicted the PG-based DRL scheme for the collaborative MEC task assignment, which consists of a fully connected neural network, called Q-network, assigned with a set of parameters θ that is updated episodically for the decision making. The Q-network directly outputs the task assignment probability matrix $\boldsymbol{\mathcal{P}}_g$.

Specifically, the controller observes the network states $\boldsymbol{\chi}^t = \{\boldsymbol{\chi}_n^t : n \in \mathcal{N}\}$ and derive the task assignment probability matrix $\boldsymbol{\Phi}^t(\boldsymbol{\chi}^t) = \boldsymbol{\mathcal{P}}_g^t(\boldsymbol{\chi}^t; \theta^t)$ by the Q-network at the beginning of each time slot t, and the edge servers will process tasks according to its task assignment probability decisions, $\boldsymbol{\mathcal{P}}_g^t = \{\boldsymbol{\mathcal{P}}_{g,n}^t : n \in \mathcal{N}\}$. At the end of the time slot, the new task arrivals $\boldsymbol{A}^t$ will be counted and the MEC network utility for the time slot $U^t = U\left(\boldsymbol{\chi}^t, \boldsymbol{\Phi}^t(\boldsymbol{\chi}^t, \boldsymbol{\mathcal{P}}_g^t(\boldsymbol{\chi}^t; \theta^t))\right)$ is calculated. The network state then transits from $\boldsymbol{\chi}^t$ to $\boldsymbol{\chi}^{t+1}$. Similarly, the PG algorithm also includes a transition storage that is used to store a pool of the most recent $\bar{\mathcal{T}}$ transition experiences, $\mho = \{\boldsymbol{\mathcal{T}}^{t-M+1}, \ldots, \boldsymbol{\mathcal{T}}^t\}$, where each experience $\boldsymbol{\mathcal{T}}^t(\boldsymbol{\chi}^t, \boldsymbol{\Phi}^t, U^t)$ is occurred at the transition of two consecutive slots t and t + 1 during the learning process. k experiences are randomly sampled as a batch from the memory pool $\mho$ to train the PG algorithm online by using the learned utility at each time slot to update the Q-network parameters, which changes the possibility of the actions. Different from the DDQN-based algorithm, the loss function used for the parameter updating is as follows, which approaches a policy with the minimal regret.

$$\mathbb{L}(\theta, \boldsymbol{\mathcal{T}}) = E\left[\left(\log \boldsymbol{\mathcal{P}}_g(\boldsymbol{\chi}; \theta) U\right)^2\right], \tag{13}$$

The online policy-gradient DRL task assignment algorithm is summarized in Algorithm 2.

Algorithm 2. Online Policy-Gradient-based DRL Cooperative MEC Task Assignment

1. Initialize θ^t of Q-network with random parameters for t = 1 and allocate the transition storage $\mho$.
2. At the beginning of scheduling slot t, the MEC controller observes the network state, $\boldsymbol{\chi}^t = \{\boldsymbol{\chi}_n^t : n \in \mathcal{N}\}$ with $\boldsymbol{\chi}_n^t = (Q_n^t, s_n^t, \boldsymbol{c}_n^t)$, which is taken as input to the Q-network with parameter θ^t to select the task assignment matrix $\boldsymbol{\Phi}^t = \{\boldsymbol{\phi}_n^t, n \in \mathcal{N}\}$.
3. After offloading and processing the tasks according to the above task assignment decision, the new tasks $\boldsymbol{A}^t = \{A_n^t : n \in \mathcal{N}\}$ is counted at the end of slot t.

Authorized licensed use limited to: NIST Virtual Library (NVL). Downloaded on August 25,2023 at 18:48:25 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3270242

4. The controller calculates the utility U^t and the network state transits to $\boldsymbol{\chi}^{t+1} = \{\boldsymbol{\chi}_n^{t+1} : n \in \mathcal{N}\}$ where $\boldsymbol{\chi}_n^{t+1} = (q_n^t + A_n^t, s_n^{t+1}, \boldsymbol{c}_n^{t+1})$
6. The new transition $\boldsymbol{\mathcal{T}^t}(\boldsymbol{\chi^t}, \boldsymbol{\Phi^t}, U^t)$ is added to the transition storage $\mathcal{U}$.
7. Once the transition storage collects $\overline{\mathcal{T}}$ transitions, the Q-network parameter θ^t is updated with a batch of transitions to minimize the loss function (13)
8. The scheduling slot index is updated by $t \leftarrow t + 1$.
9. Repeat from step 2 to 8.

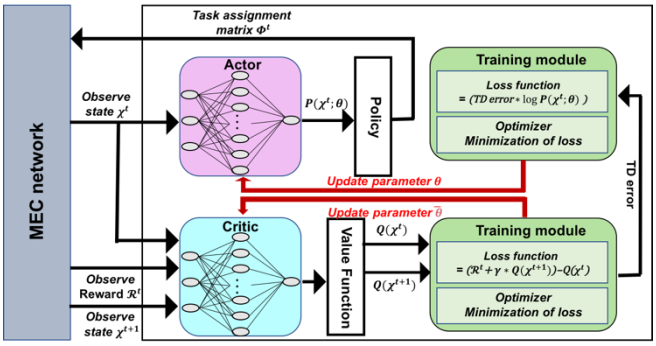

Fig. 5. Actor-Critic-based cooperative MEC task assignment.

4.3 Hybrid DRL Task Assignment Algorithm

The two proposed algorithms, valued-based and policy-based DRL algorithms, have their own benefits, but also incur some shortcomings under different scenarios. The value-based DRL method can address the state space explosion problem, but it may overestimate the action as the state space is huge. The policy-based DRL method performs efficiently in continuous or large action space but may suffer from a large variance in action and stagnate prematurely at local optima [35]. Actor-critic (AC) DRL methods [37] recently attracted research interests in the machine learning community, which adopt a hybrid value and policy learning approach. They typically consist of two DNNs, an actor network to select actions and a critic network to estimate the state value function and criticize the actions made by the actor. The critic learns about and critiques whatever policy is currently being followed by the actor with a temporal difference (TD) error to drive the learning in both actor and critic for achieving the optimal policy.

Fig. 5 illustrates the AC-based task assignment algorithm. The MEC controller observes the network states $\boldsymbol{\chi^t}$ and runs the actor to select the task assignment probability matrix $\boldsymbol{\Phi^t}(\boldsymbol{\chi^t}; \theta_A)$, and the edge servers then offload and process the tasks according to the task assignment probability. The new task arrivals $\boldsymbol{A^t}$ will be counted at the end of the time slot, and the network states transits from $\boldsymbol{\chi^t}$ to $\boldsymbol{\chi^{t+1}}$. On the other hand, the task assignment decision is evaluated by the critic. To approximate the state value and perform temporal difference error learning to update the parameters θ_C of the critic, the critic will first take the network state $\boldsymbol{\chi^t}$ as the input to approximate the state value $Q(\boldsymbol{\chi^t})$ of time slot t. Once it receives the utility U^t and the network state $\boldsymbol{\chi^{t+1}}$ at time slot t+1, TD learning is performed according to the loss function as defined in (14) at the bottom of this page. Finally, the actor updates its parameters θ_a in the same way as the policy gradient algorithm except that the utility is replaced with TD error we calculated in the critic, by (15). The algorithm is summarized in Algorithm 3.

Algorithm 3. Online Actor-Critic-based DRL Cooperative MEC Task Assignment

1. Initialize the Actor and the Critic networks with two sets of θ_A^t and θ_C^t random parameters for $t = 1$.
2. At the beginning of scheduling slot t, the MEC controller observes the network state, $\boldsymbol{\chi}^t = \{\boldsymbol{\chi}_n^t : n \in \mathcal{N}\}$ where $\boldsymbol{\chi}_n^t = (q_n^t, s_n^t, \boldsymbol{c}_n^t)$, and the Actor with parameters θ_A^t determines the task assignment probability matrix, $\boldsymbol{\Phi}^t = [\boldsymbol{\phi}_n^t : n \in \mathcal{N}]$.
3. After offloading and processing the tasks according to the above task assignment decision, the new tasks $\boldsymbol{A}^t = \{A_n^t : n \in \mathcal{N}\}$ are counted at the end of slot t.
4. The MEC network utility U^t is calculated. The network state transits to $\boldsymbol{\chi}^{t+1} = \{\boldsymbol{\chi}_n^{t+1} : n \in \mathcal{N}\}$ where $\boldsymbol{\chi}_n^{t+1} = (q_n^t + A_n^t, s_n^{t+1}, \boldsymbol{c}_n^{t+1})$.
5. The Critic calculates TD error with $Q(\boldsymbol{\chi})$ and $Q(\boldsymbol{\chi}')$
6. The Critic network updates the parameters θ_C^t to minimize the loss function (14).
7. The Actor network updates the parameters θ_A^t to minimize the loss function (15)
8. The scheduling slot index is updated by $t \leftarrow t + 1$.
9. Repeat from step 2 to 9.

5 Evaluation

In this section, we evaluate the stochastic task assignment performance achieved by our three derived deep reinforcement learning schemes and compare them with each other and with several other baseline algorithms to gain insights.

5.1 General Setup

We compare the performance of the proposed DRL-based schemes with the following baselines:

1) Edge Server Self-Processing: An edge server processes all the tasks it receives from the associated users by itself. There is no task offloading.
2) Cloud Execution: An edge server offloads all its received tasks to the cloud data center for processing.
3) Subgradient with Dual Decomposition: task assignment optimization based on subgradient with dual decomposition as proposed in [41].
4) Q-learning: task assignment optimization based on conventional Q-learning.

$$\mathbb{L}_A(\theta_A) = E\left[\left((1-\gamma)Q(\boldsymbol{\chi}, \boldsymbol{\Phi}(\boldsymbol{\chi}; \theta_A)) + \gamma[Q(\boldsymbol{\chi}', \boldsymbol{\Phi}(\boldsymbol{\chi}'; \theta_A))] - Q(\boldsymbol{\chi}, \boldsymbol{\Phi}(\boldsymbol{\chi}; \theta_A))\right)^2\right] \quad (14)$$

$$\mathbb{L}_C(\theta_C) = E\left[\left(\left((1-\gamma)Q(\boldsymbol{\chi}, \boldsymbol{\Phi}(\boldsymbol{\chi}; \theta_A)) + \gamma[Q(\boldsymbol{\chi}', \boldsymbol{\Phi}(\boldsymbol{\chi}'; \theta_A))] - Q(\boldsymbol{\chi}, \boldsymbol{\Phi}(\boldsymbol{\chi}; \theta_A))\right) * (\log \boldsymbol{\mathcal{P}}(\boldsymbol{\chi}; \theta_A))\right)^2\right] \quad (15)$$

Authorized licensed use limited to: NIST Virtual Library (NVL). Downloaded on August 25,2023 at 18:48:25 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3270242

We simulated multiple MEC network scenarios with different numbers of edge servers, various task arrival rates, node processing, network delay, and other system parameters. Due to the page limit, we present the results for several typical settings. In the simulations, we assume the slot duration is 20 ms for the following considerations. First, we consider real-time edge computing applications, such as 3D scene reconstruction, AR/VR and others, with an end-to-end service delay at an order of 10s-100s ms [49]. A scheduling slot duration of 20 ms will be reasonable to capture the bursty traffic demands and queue delay states. Another motivation to choose 20 ms slot duration is the underlying networks. 5G new radio defines a frame to be 10 ms in duration and the minimal round-trip time at the application layer can be around 20 ms. Further, the slot is used as the time unit in our evaluation, and the delay is measured by the number of time slots and the actual value of the time slot duration can be changed. The processing capabilities of edge servers are modeled with two states characterizing the high and low with {4, 1} tasks per slot to reflect the edge servers with different available resources and heterogeneous, fluctuating capabilities. The processing states s_n^t, $\forall n \in \mathcal{N}$ of different edge servers are independent of each other and evolve according to a Markov chain. Similarly, the network delay between two edge servers, c_{nj}^t, $\forall n, j \in \mathcal{N}$, is modeled as a Markov chain with three states, {1, 0.5, 0.2} time slots. The network delay between the edge server and the cloud $c_{nn_c}^t$, $\forall n \in \mathcal{N}$ is assumed to be a large value, 15 slots due to the transmission over the Internet. In addition, we assume the maximal size of the queue buffer at an edge server is 30 tasks. We consider that task queue overflow impacts QoS more significantly than that of service delay, thus, the weight factors in the utility function, w_d and w_o are set to be 1 and 10, respectively.

The parameters of the proposed DRL schemes are set according to the needs and through our simulation experiments to achieve good learning accuracy and reasonable convergency time. For the DDQN-based task assignment scheme (Algorthim 1), the neural networks of Q-tar and Q-eval in each subagent have a single hidden layer with 10 neurons each. We employ the ReLU (Rectified Linear Unit) function as the activation function of the hidden layer because ReLU reduces the vanishing gradient problem [32] and is simple to implement. The softmax function is used as the activation function of the output layer to output the possibility matrix for action selection because it can calculate a vector of real numbers into a vector of probability values that ranges between 0 and 1 with the sum of the probabilities being equal to 1 [32, 37]. The number of iterations for updating parameters of Q-tar is set to be 20, and the memory replay size and the batch size are set to be 40 because they yield a good balance between the convergence time and learning accuracy through our simulation experiments. The training and learning process is triggered when the system collects enough samples and it will pull out all the samples to train.

For the proposed PG-based scheme (Algorthim 2), the Q-network has a single hidden layer with 35 neurons. A larger number of neurons are used in the hidden layer for the PG-based scheme because the action policy is directly determined without decomposition and more neurons are needed to handle the large network states. Similar to the DDQN-based scheme, we employ ReLU as the activation function of the hidden layer and Softmax for the output layer to output the possibility matrices for the action selection. The number of iterations for updating parameters of Q-network is set to be 20, and the memory replay size and the batch size are set to be 40 because they yield good convergence time and learning accuracy in our simulation experiments.

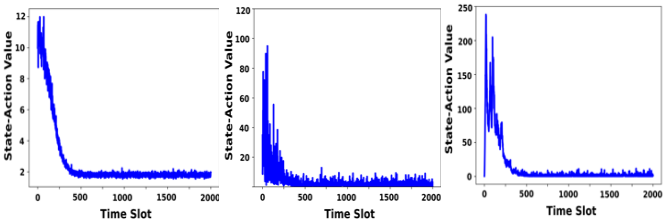

Fig. 6. Convergence of the proposed DRL-based task assignment schemes in the learning process; (a) DDQN-based scheme; (b) PG-based scheme; (c) AC-based scheme.

For the proposed hyrbid AC-based scheme (Algorithm 3), the actor is built with a neural network that includes one hidden layer with 35 neurons. The tanh function is employed as the activation function of the hidden layer because it produces a zero-centered output, thereby easily supporting the TD-based learning process [35, 36]. The softmax is used as the activation function for the output layer. The critic is constructed by a 50-neuron network with a hidden layer using tanh as the activation function but no activation function for the output layer.

5.2 Convergence Performacne

We first investigate the convergence property of the proposed DRL-based algorithms under dynamic stochastic MEC network environments. Figs. 6(a), 6(b) and 6(c) illustrate the convergence of the state value function for DDQN, PG, and AC-based task assignment algorithms, respectively. Three edge servers and one cloud data center are used in the simulations and the number of tasks arriving at each of the edge servers follows an independent Poisson arrival process with an average arrival rate of 4 tasks per slot. We can observe that each of the three algorithms spends a short time-period to learn and then converges to a stable state in a reasonable time, specifically few hundreds of decision slots.

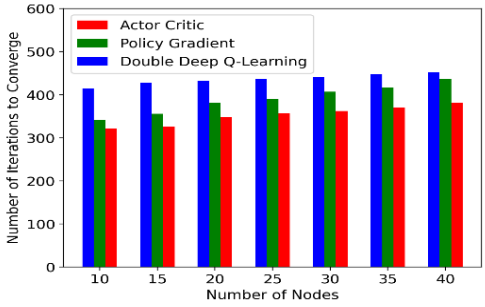

Fig. 7. Convergence of the proposed DRL-based task assignment schemes versus the network size.

Fig. 7 illustrates the convergence of the proposed three DRL-based task assignment algorithms versus the network size. DRL learns the parameterized value function or policy with DNNs [32, 35, 36], instead of all action values in all states separately as in traditional Q learning. When the network size increases, the value function or policy will be more complex, it takes more time to capture the function parameters. The algorithms thus take more time to converge with the increased network size.

Authorized licensed use limited to: NIST Virtual Library (NVL). Downloaded on August 25,2023 at 18:48:25 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3270242

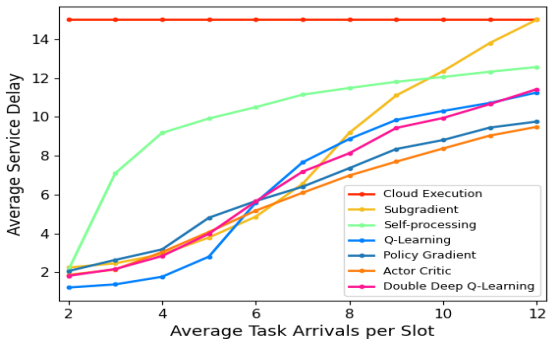

Fig. 8. The average task service delay versus the average task arrivals per slot for different algorithms.

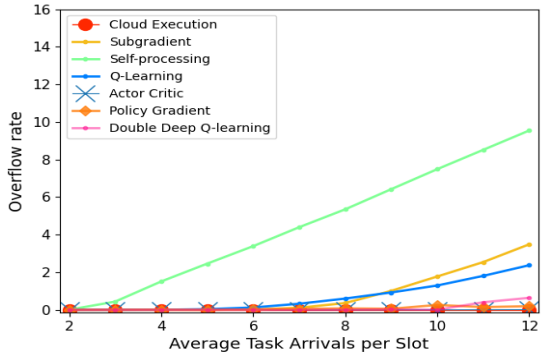

Fig. 9. The average queue overflow rate versus the average task arrivals per slot for different algorithms.

5.3 Performance Comparision of Different Algorithms

In this section, we compare the performance of the proposed DRL-based task assignment algorithms with baselines. We first investigate the effect of edge servers' task arrival rates on the performance. Figs 8 and 9 show the average task service delay and the average task queue overflow rate (the average number of overflowed tasks per slot), respectively, for different algorithms. The edge servers and the cloud data center cooperate to jointly process the tasks. The task arrivals at the edge servers follow independent Poisson arrival process. One edge server, node 1, changes its average task arrival rate (the average number of task arrivals per slot) in the experiment while the average task arrival rates of the other nodes are fixed. The service delay includes the network delay and queuing delay, and the queueing delay is the time a task waits in the buffer for being processed and the time it is processed, i.e., the sojourn time of a task in the system, as described in Section 3. The unit for the delay is time slot duration.

The proposed learning-based schemes outperform the subgradient-based algorithm and other baselines in terms of service delay, especially when the workload is high. This is because the learning-based schemes can capture the dynamic MEC network state transitions and determine the optimal task assignment matrix by taking into consideration the effects of time-varying stochastic task arrivals and network conditions on the expected long-term performance. On the other hand, the subgradient optimization algorithm makes task assignment decisions based on the current average task arrivals and network conditions without considering the underlying dynamics and randomness as well as their impacts to the long-term QoS. This thus may cause a lot of tasks to be shipped to the cloud data center for processing so that it leads to a large network delay and a large task service delay.

The traditional Q-learning algorithm can achieve the same or slightly better task service delay performance when the task arrival rate is low. The reason is that the queue state space is relatively small under the low workload and the optimal solution can be achieved by the traditional value-based tabular Q-learning method. However, the average task service delay of the traditional Q-learning scheme quickly increases as the task arrival rate become high. This is because the traditional Q-learning may overestimate the control policy in the large state and action space [32] and makes inaccurate task assignment decisions so that a node may choose to process the tasks locally even it does not have sufficient resources or may cause shipping the tasks to the cloud for processing when it is capable to handle them. The proposed DDQN-based algorithm addresses the large state space problem of the traditional Q-learning scheme by using two value-based Q-learning neural networks and it performs relatively well when the workload is high. However, it may still cause overestimation of the control policy when the task arrival rate is high and the state-action space is huge.

The massive state-action space problem is handled much better in the policy-based and hybrid actor-critic approaches. The PG-based algorithm can prevent the overestimation by periodically amending the task assignment action probability matrix that outputs the optimal decision and improves the system performance. However, with the PG-based algorithm, the perdition process of the action probability may result in premature stagnation at local optima [35], which degrades the performance. The proposed AC-based task assignment scheme offers the best performance in terms of the task service delay and queue overflow as it combines the policy-based approach in the decision-making to avoid the overestimation and the value-based approach to determine the optimal task assignment matrix with the precise utility related to the performance at each time slot.

For the cloud execution scheme, a large value of network delay is always incurred to ship the tasks to the cloud data center over the Internet. The edge server self-processing scheme performs well when the workload is very low because an edge server can process the tasks by itself. However, as the task arrival rate becomes high, the edge server does not have enough resources to process all the tasks so that the service delay rapidly increases, and more tasks are overflowed from the queue. In Fig. 9, we present the task queue overflow rates for different algorithms. The overflow rates for the learning-based and subgradient-based task assignment algorithms are close to zero because the algorithms give the task queue overflow minimization a high weight and the edge servers will forward the tasks to the cloud data center when their buffers become full. We will thus focus on the task service

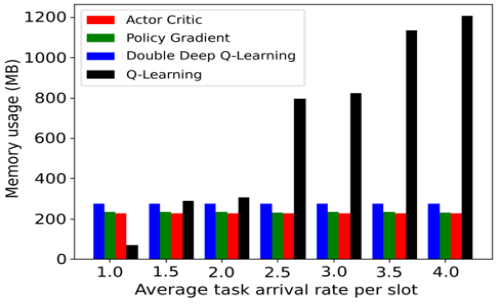

Fig. 10. Memory usage versus the average task arrivals per slot for different learning algorithms.

Authorized licensed use limited to: NIST Virtual Library (NVL). Downloaded on August 25,2023 at 18:48:25 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3270242

delay performance of the DRL-based schemes under different scenarios in the following sections as their task queue overflow is always close to zero.

5.4 Memory Usage of Differenent Algorithms

Next, we evaluate the memory usage of the proposed DRL-based algorithms and traditional Q-learning algorithm during the learning process. As illustrated in Fig. 10, the traditional tabular Q-learning consumes much higher memory resources than the DRL-based schemes and cannot scale well due to the explosion in state and action spaces, making it unviable for the scenarios such as very high task arrival rates or very large networks. On the other hand, the memory usage by the DRL-based task assignment schemes scale well as the task arrival rate and the state/action space increases.

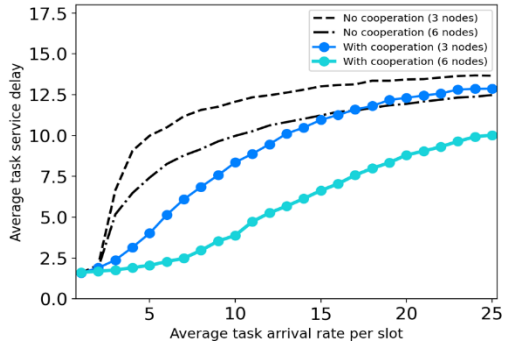

Fig. 11. The average task service delay versus the average task arrivals per slot with and without edge server cooperation.

5.5 Performance of Horizontal Cooperation

In this subsection, we evaluate the impact of the horizontal cooperation of the edge servers to the system performance. Fig. 11 shows the average task service delay of the proposed AC-based scheme with and without the edge server cooperation for task processing. We change the task arrival rate of one edge server, node 1, and fix the task arrival rate of the other edge servers. It can be observed that with the cooperation of the edge servers, the growing speed of the task service delay is decreased as the task arrival rate of edge server 1 increases, compared to the case without edge server cooperation. This is because edge server 1 can offload its tasks to other edge servers with the horizontal cooperation to jointly process the tasks when its workload is high. Without the horizontal cooperation, the edge server can only offload its tasks to the cloud so that it incurs a large network delay.

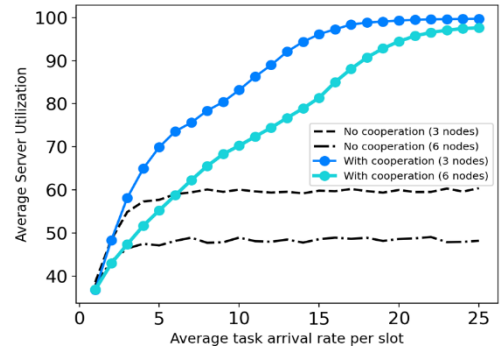

Fig. 12. The average server utilization rate versus the average task arrivals per slot with and without edge server cooperation.

In addition, with more edge servers, the MEC computation capability increases so that the average task service delay reduces. We only show the results of the AC-based scheme here due to the page limit, and the same conclusions can be obtained for the DDQN and PG-based schemes.

In Fig. 12, we investigate the average utilization rate of the edge servers. The utilization rate of an edge server is defined as the ratio of the actual task process rate to the task process capability of the edge server. We again fix the task arrival rate of all the edge servers except edge server 1. Compared to the case without the edge-to-edge cooperation, the cooperation of edge servers can increase the average utilization rate. This is because that the workload of different edge servers can be balanced through cooperation, more specifically, an edge server may process the tasks for others when its own workload is low and will forward part of its tasks to others when its own workload is high. In addition, we can observe that the utilization rate initially increases and then approaches to a fixed value as the task arrival rate grows to the total task processing capacity of the edge servers.

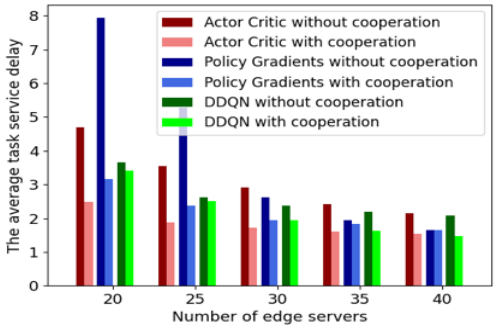

Fig. 13. The average task service delay versus the number of edge servers in the MEC network for the proposed DRL-based schemes.

5.6 Different Sizes of MEC Networks

Fig. 13 illustrates the average task service delay of the proposed DRL-based schemes under different numbers of edge servers in the MEC network. In the experiment, the task arrivals at each of the edge servers are random with a total average arrival rate of 20. We can observe again that the proposed AC-based algorithm performs the best, especially with edge server cooperation. Although the PG-based algorithms can perform efficiently in continuous or large action space, they may suffer from a large variance in action and stagnate prematurely at local optima [35]. For the scenarios without cooperation, i.e., the edge servers do not cooperate to process the tasks, the PG algorithm may stagnate prematurely around 20 and 25 servers so that it makes a premature task assignment decision that the edge server will not be able to process certain tasks in time as the resources are limited. The edge server will offload these tasks to the remote cloud data center for task processing, which incurs a large network delay, given no cooperation with other edge servers. Further, the results demonstrate that the cooperation of edge servers can greatly enhance the overall system processing capabilities and reduce the average task service delay, compared to no edge server cooperation, which is consistent with Fig. 11. The system performance can be improved as the number of edge servers increases. This is because the total MEC computation capability increases with

Authorized licensed use limited to: NIST Virtual Library (NVL). Downloaded on August 25,2023 at 18:48:25 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3270242

more densely deployed edge servers and the proposed schemes can learn the optimal control policy and assign the tasks to the edge servers with sufficient resources for processing.

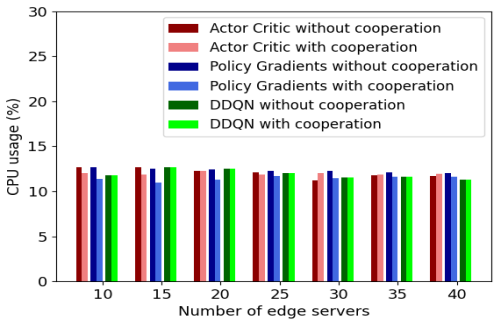

Fig. 14. CPU usage versus the number of edge servers in the MEC network for the proposed DRL-based schemes.

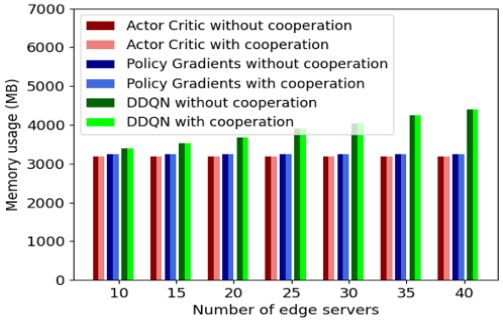

Fig. 15. Memory usage versus the number of edge servers in the MEC network for the proposed DRL-based schemes.

Figs. 14 and 15 show the CPU and memory usages of the proposed DRL-based task assignment algorithms under different MEC network sizes. We observe that the memory and CPU usages slightly increase as the network size increases. This is because the memory and CPU usages of the DRL-based algorithms mainly depend on the size, structure, and parameter updates of deep neural networks used by the algorithms. The DNN update process in the decision inference of the DRL algorithms is not significantly affected by the MEC network size or the state space [31, 35]. The memory and CPU usages of the DRL-based algorithms are also not significantly affected by the MEC network size. The proposed algorithms can scale well in a relatively large network.

6 Related Work

6.1 Task Offloading

Significant research work has investigated computational task offloading from mobile devices to MEC edge servers. In [9], the authors formulated task offloading as a QoS optimization problem by assuming the network environments were deterministic, and they transformed the original problem into a convex problem of latency minimization to decompose and solve it. Similarly, the authors in [10] formulated the offloading as a convex optimization problem for minimizing the weighted sum of mobile energy consumption under the constraint of computation latency with infinite or finite edge computation capacity and proposed an optimal offloading policy according to users' wireless channel gains and local computing energy consumption. In [11], a resource demand estimation and provisioning scheme is presented for an edge micro data center to support the requested services and maximize resource utilization. In [12], the problem of delay optimal task offloading was considered as a Markov decision process and a search algorithm was developed to find the optimal solution. In [13], the computation offloading policy for a MEC system with wireless energy harvesting-enabled mobile devices were investigated using a Lyapunov optimization technique and an approximate optimization solution was obtained. In [14], the authors formulated the task offloading problem as a one-to-many matching game to determine the best edge server to offload the tasks to while minimizing overall energy consumption. In [15], the advantages of offloading the tasks to distributed MEC edge servers have been demonstrated in terms of service delay and power efficiency, compared to traditional offloading to centralized cloud data centers.

More recently, the authors in [42] proposed a Bayesian online learning algorithm for joint optimization of the service and wireless network parameters to process data streams offloaded from mobile devices in video analytics applications, which maximized the analytics accuracy in a cross-layer way. In [43], a reinforcement learning scheme was developed to learn varying demands and then reserve edge servers to support the computation tasks from connected vehicles. The authors in [44] jointly optimized task offloading and caching to minimize a composite metric of mobile device energy saving and task response latency by using an alternating minimization algorithm. In [45], a resource allocation and task scheduling optimization scheme based on service priority was proposed to minimize the total delay of the system and ensure the delay requirement of high priority services. The authors in [46] proposed a task offloading scheme for an air-ground integrated edge computing system according to information freshness, and the optimal offloading policy was obtained based on DRL without presuming the dynamic network states to be known.

6.2 Edge Server Cooperation

Vertical cooperation between the MEC edge layer and the remote cloud layer has been studied. In [16], an approximate algorithm for joint resource allocation of a MEC server and a cloud data center is designed to minimize carbon footprint for video streaming service. In [17], a resource provisioning scheme for an edge micro datacenter is presented, which conducts resource estimation and management while deciding what type of data is to be uploaded to the cloud based on fluctuating relinquish probability of a user, service type, and service price. In [18], the authors proposed a hierarchical game framework to model the interactions where the edge servers help the cloud data center operators process delay-sensitive tasks from mobile users that originally target at the cloud data centers and to determine the edge server resource allocation, service price, and pairing of edge servers and data center operators with Stackelberg game and matching theory. In [19], a reinforcement learning-based resource management algorithm is developed, which obtains the optimal policy for dynamically offloading the tasks from energy-harvesting MEC edge servers to a centralized cloud data center and provisioning the edge servers to minimize the long-term system cost. The authors in [47] analyzed the delay in a cloud-fog-edge computing system and proposed a computational resource allocation method to maximize a social welfare metric, constrained to specific QoS requirements. In [48], a task offloading scheme in integrated edge-fog-cloud

Authorized licensed use limited to: NIST Virtual Library (NVL). Downloaded on August 25,2023 at 18:48:25 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3270242

computing environments was proposed and the performance was analyzed, where a device offloaded its generated tasks to a layer for computation, a task was only accepted if the queue size was below the pre-defined threshold, otherwise, it would be offloaded to the next layer. These previous works mainly consider a hierarchical network architecture and focus on the vertical interaction among users, MEC edge servers and cloud data centers, but they either abstract the MEC layer as a single edge server or assume that the edge servers are independent of each other and there is no cooperation among them.

Recently, exploring the cooperation of geographically distributed edge servers for jointly processing the tasks has received some research attention. In [39], the benefits of edge server cooperation were demonstrated through several application scenarios, however the algorithms for such cooperation were not presented. The authors in [20, 41] addressed the task assignment problem when an edge server can forward its tasks to the neighboring edge servers for processing. However, they formulated the task assignment as a classic convex optimization problem based on a much-simplified network model where the task arrival rate, the queueing delay, and the network delay are all deterministic and known beforehand. Such a deterministic model not only contrasts many practical MEC scenarios (where networks are dynamic and stochastic, and the network state and task arrival statistics are unknown in advance) but also fails to capture the broad range of network parameters and ignores the effects of various dynamics so that the task assignment depends only on one-shot optimization given a static MEC network state, irrespective of the underlying non-stationary process.

In our previous work [30], we proposed a machine learning based stochastic optimization approach for task assignment, which can achieve optimal performance with no assumptions on knowing the dynamic network state and task arrival statistics. We also showed that the machine learning-based task assignment approach outperforms the model-based deterministic task assignment optimization. However, the algorithm used in [30] is based on conventional tubular Q-learning, which is subject to scalability limitation due to the state space explosion and high memory consumption. In [33], we proposed a double deep Q network-based learning algorithm for task assignment in dynamic MEC networks, which addressed the state space explosion problem. This paper significantly extends our previous work with new DRL-based schemes and performance comparison of value-based, policy-based, and hybrid DRL algorithms.

7 Conclusions and Future Work

In many MEC scenarios, the user task arrival statistics, task processing rates at edge servers, and network delay between edge servers are time-varying and unknown beforehand. Therefore, it is more reasonable and compelling to cast task assignment as a stochastic and dynamic optimization problem, instead of model-based deterministic optimization. In this paper, we proposed and investigated a stochastic cooperative framework, which enables cooperation among the various entities of a MEC system, including the horizontal cooperation among geographically distributed MEC edge servers and the vertical cooperation between edge servers and cloud data centers, to jointly process user tasks to balance the varying workload on heterogeneous edge servers and improve the QoS in dynamic MEC networks. The task assignment optimization problem is formulated as a Markov decision process by taking into consideration the interaction of the involved entities. Particularly, we proposed three online deep reinforcement learning algorithms, value-based, policy-based, and a hybrid approach, which all perform sequential task assignment decisions in a series of time slots for the edge servers to help each other process tasks according to an assignment matrix that optimizes a long-term expected QoS-aware utility function in terms of task service delay and queue overflow. A function decomposition technique was introduced to simplify the problem in the learning process. The proposed online DRL-based algorithms can learn the optimal task assignment matrix with no assumption on prior knowledge of task arrival statistics and network state transitions. Their convergence was validated. We compared the performance of the proposed DRL-based task assignment schemes with each other and with several existing baseline algorithms. The evaluation results show the proposed DRL-based schemes significantly reduce the task service delay and task queue overflow rate, especially in high workload, compared to the baselines. The AC-based hybrid scheme performs the best. In addition, the DRL-based schemes require much less memory than the conventional Q-learning based algorithm. They can scale well in the dense deployment and handle a relatively large MEC network with 40 edge servers. We also demonstrated that the service delay of user tasks can be significantly reduced by allowing edge servers to cooperate and offload the tasks to each other.

Reinforcement learning models take time to converge and the time cost of retraining a reinforcement learning model to adapt to the changing system states and new configurations is high if the real-world system is highly dynamic. There is recent advancement in machine learning techniques that trains an ensemble of DRL models, each for a different system environment, and employs a high-level meta policy to choose or combine the results of ensemble members based on the system environment states [50, 51] so that the algorithm can adapt to the changing system environments with few-shot retraining. As part of our future work, we will exploit ensemble or meta reinforcement learning to design fast adaptive task scheduling algorithms for cooperative MEC in highly dynamic system environments.

References

[1] B. Chandramouli, J. Claessens, S. Nath, I. Santos, and W. Zhou, "Race: Real-time applications over cloud-edge," in *Proc. of ACM SIGMOD'12*, pp. 625–628, 2012.

[2] A. Chandra, J. Weissman, and B. Heintz, "Decentralized edge clouds," IEEE Internet Computing, vol. 17, no. 5, pp. 70–73, 2013.

[3] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, 2009.

[4] U. Drolia, R. Martins, J. Tan, A. Chheda, M. Sanghavi, R. Gandhi, and P. Narasimhan, "The case for mobile edge-clouds," in *Proc. of IEEE UIC/ATC'13,* pp. 209–215, 2013.

[5] Y. Jararweh, A. Doulat, O. AlQudah, E. Ahmed, M. Al-Ayyoub, and E. Benkhelifa, "The future of mobile cloud computing: Integrating cloudlets and mobile edge computing," in *Proc. of IEEE ICT'16*, 2016.

[6] H. Liu, F. Eldarrat, H. Alqahtani, A. Reznik, X. de Foy, Y. Zhang, "Mobile Edge Cloud System: Architectures, Challenges, and Approaches," *IEEE Systems Journal*, vol. 12, no. 3, pp. 2495-2508, Sept. 2018.

[7] ETSI Whitepaper, "Harmonizing standards for edge computing - A synergized architecture leveraging ETSI ISG MEC and 3GPP specifications," https://www.etsi.org/images/files/ETSIWhitePapers/, 2020.

Authorized licensed use limited to: NIST Virtual Library (NVL). Downloaded on August 25,2023 at 18:48:25 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3270242

[8] 3GPP TS 23.558, "Architecture for Enabling Edge Applications," https://www.3gpp.org/DynaReport/23558.htm, 2021.
[9] C. Wang, C. Liang, F. R. Yu, Q. Chen, and L. Tang, "Computation offloading and resource allocation in wireless cellular networks with mobile edge computing," *IEEE Trans. Wireless Commun*., vol. 16, no.8, pp. 4924–4938, Aug. 2017.
[10] C. You, K. Huang, H. Chae, and B. H. Kim, "Energy-efficient resource allocation for mobile-edge computation offoading," *IEEE Trans. Wireless Commun.*, vol. 16, no. 3, pp. 1397–1411, Mar. 2017.
[11] M. Aazam and E.-N. Huh, "Dynamic resource provisioning through fog micro datacenter," in *Proc. of the IEEE PerCom Workshops*, St. Louis, MO, Mar. 2015, pp. 105–110.
[12] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, 'Delay-optimal computation task scheduling for mobile-edge computing systems," in *Proc. IEEE ISIT*, Barcelona, Spain, Jul. 2016.
[13] 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.
[14] B. Gu, Y. Chen, H. Liao, Z. Zhou, and D. Zhang, "A distributed and context-aware task assignment mechanism for collaborative mobile edge computing," *Sensors*, vol. 18, no. 8, pp. 2423–2439, 2018.
[15] S. Sarkar, S. Chatterjee, and S. Misra, "Assessment of the suitability of fog computing in the context of internet of things," *IEEE Transactions on Cloud Computing*, 2016.
[16] C. Do, N. Tran, C. Pham, M. Alam, J. H. Son, and C. S. Hong, "A proximal algorithm for joint resource allocation and minimizing carbon footprint in geo-distributed fog computing," in *the Proc. of the IEEE ICOIN*, pp. 324–329, Siem Reap, Cambodia, Jan. 2015.
[17] M. Aazam and E.-N. Huh, "Dynamic resource provisioning through fog micro datacenter," in *Proc. Of IEEE PerCom Workshops*, pp. 105–110, St. Louis, MO, Mar. 2015,
[18] H. Zhang, Y. Xiao, S. Bu, D. Niyato, F. R. Yu, and Z. Han, "Computing resource allocation in three-tier IoT fog networks: A joint optimization approach combining stackelberg game and matching," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1204–1215, 2017.
[19] J. Xu, L. Chen, and S. Ren, "Online learning for offloading and autoscaling in energy harvesting mobile edge computing," *IEEE Trans. Cogn. Commun. Netw.*, vol. 3, no. 3, pp. 361–373, Jul. 2017.
[20] Y. Xiao and M. Krunz, "QoE and power efficiency tradeoff for fog computing networks with fog node cooperation," in *Proc. of IEEE INFOCOM'17*, Atlanta, GA, May 2017.
[21] A. Gudipati, D. Perry, E. Li, and S. Katti, "SoftRAN: Software Defined Radio Access Networks," ACM HotSDN, Aug. 2013.
[22] K.R. Smith, H. Liu, L. Hsieh, X. de Foy, R. Gazda, "Wireless Adaptive Video Streaming with Edge Cloud," Wiley/Hindawi Journal of Wireless Communications and Mobile Computing, vol. 2018, pp. 1-13, Dec. 2018.
[23] X. Chen, Z. Han, H. Zhang, G. Xue, Y. Xiao, and M. Bennis, "Wireless resource scheduling in virtualized radio access networks using stochastic learning*," IEEE Transactions on Mobile Computing*, vol. 17, no. 4, pp. 961-974, 2018.
[24] D. Adelman and A. J. Mersereau, "Relaxations of weakly coupled stochastic dynamic programs," *Oper. Res.*, vol. 56, no. 3, pp. 712–727, Jan. 2008.
[25] D. P. Bertsekas, *Dynamic programming and optimal control*. Athena Scientific, Belmont, MA, 1995.
[26] S. M. Ross, *Introduction to stochastic dynamic programming*. Academic press, 2014.
[27] M. L. Puterman and M. C. Shin, "Modified policy iteration algorithms for discounted Markov decision problems," *Management Science*, vol. 24, no. 11, pp. 1127–1137, 1978.
[28] R. Howard, *Dynamic Programming and Markov Processes*. The MIT Press, 1960.
[29] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, The MIT Press, 2005.
[30] L. Hsieh, H. Liu, Y. Guo, R. Gazda, "Task Management for Cooperative Mobile Edge Computing," In Proceedings of the 5th ACM/IEEE Symposium on Edge Computing (SEC) HotWoT workshop, 2020.
[31] V. Francois-Lavet, Vincent et al., "An Introduction to Deep Reinforcement Learning," Foundations and Trends in Machine Learning. Vol. 11, no. 3-4, pp. 219–354, 2018.
[32] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-Learning," In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI'16), Pages 2094–2100, February 2016.
[33] L. Hsieh, H. Liu, Y. Guo, R. Gazda, "Quality of Service Optimization in Mobile Edge Computing Networks via Deep Reinforcement Learning," In Proceedings of 15th International Conference on Wireless Algorithms, Systems, and Applications (WASA), 2020.
[34] J. N. Tsitsiklis and B. van Roy, "Feature-based methods for large scale dynamic programming," *Mach. Learn.*, vol. 22, no. 1-3, pp. 59 - 94, Jan. 1996.
[35] I. Grondman, L. Busoniu, G.A. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 42(6), 1291-1307, 2012.
[36] Williams, Ronald J. "Simple statistical gradient-following algorithms for connectionist reinforcement learning." Machine learning 8(3), pp. 229-256, 1992.
[37] V.R. Konda and J.N. Tsitsiklis, "Actor-critic algorithms," Advances in neural information processing systems, 2000.
[38] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, M. Bennis, "Optimized Computation Offloading Performance in Virtual Edge Computing Systems Via Deep Reinforcement Learning," IEEE Internet of Things Journal, vol. 6 , no. 3, pp. 4005 – 4018, June 2019.
[39] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili, "Collaborative mobile edge computing in 5g networks: New paradigms, scenarios, and challenges," IEEE Communications Magazine, vol. 55, no. 4, pp. 54–61, 2017.
[40] Open Edge Compute Initiative, https://www.openedgecomputing.org/.
[41] Y. Xiao and M. Krunz, "Distributed Optimization for Energy-Efficient Fog Computing in the Tactile Internet," in *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 11, pp. 2390-2400, Nov. 2018.
[42] A. Galanopoulos, J. A. Ayala-Romero, D. J. Leith and G. Iosifidis, "AutoML for Video Analytics with Edge Computing," in *Proceedings of 2021 IEEE INFOCOM*, July 2021.
[43] J. Zhang, S. Chen, X. Wang and Y. Zhu, "DeepReserve: Dynamic Edge Server Reservation for Connected Vehicles with Deep Reinforcement Learning," in *Proceedings of 2021 IEEE INFOCOM*, July 2021.
[44] C. Tang, C. Zhu, X. Wei, H. Wu, Q. Li and J. J. P. C. Rodrigues, "Task Offloading and Caching for Mobile Edge Computing," in *Proceedings of 2021 International Wireless Communications and Mobile Computing* (IWCMC), pp. 698-702, June 2021.
[45] J. X. Liao and X. W. Wu, "Resource Allocation and Task Scheduling Scheme in Priority-Based Hierarchical Edge Computing System," in *Proceedings of 2020 19th International Symposium on Distributed Computing and Applications*, Dec. 2020.
[46] X. Chen, C. Wu, T. Chen, Z. Liu, H. Zhang, M. Bennis, H. Liu, and Y. Ji, X. Chen et al., "Information Freshness-Aware Task Offloading in Air-Ground Integrated Edge Computing Systems," *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 1, pp. 243-258, Jan. 2022.
[47] R. Fantacci and B. Picano, "Performance Analysis of a Delay Constrained Data Offloading Scheme in an Integrated Cloud-Fog-Edge Computing System," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 10, pp. 12004-12014, Oct. 2020.
[48] S. D. Okegbile, B. T. Maharaj and A. S. Alfa, "A multi-user tasks offloading scheme for integrated edge-fog-cloud

Authorized licensed use limited to: NIST Virtual Library (NVL). Downloaded on August 25,2023 at 18:48:25 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3270242

computing environments," in *IEEE Transactions on Vehicular Technology*, April 2022.

[49] W. Zhang, S. Li, L. Liu, Z. Jia, Y. Zhang, and D. Raychaudhuri. "Hetero-edge: Orchestration of real-time vision applications on heterogeneous edge clouds," In *Proceedings of 2019 IEEE INFOCOM*, 2019.

[50] T. M. Hospedales, A. Antoniou, P .Micaelli, and A. J. Storkey, "Meta-learning in neural networks: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[51] A. Modi, N. Jiang, A. Tewari, and S. Singh, "Sample Complexity of Reinforcement Learning using Linearly Combined Model Ensembles," in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, 2020.

ACKNOWLEDGMENTS

Certain commercial equipment, instruments, or materials are identified in this paper in order to specify the experimental procedure adequately. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that the materials or equipment identified are necessarily the best available for the purpose.

Li-Tse Hsieh received his B.S. degree in Computer Science and Information Engineering from Fu Jen Catholic University, New Taipei City, Taiwan in 2014, M.S. in Computer Science from The Catholic University of America in 2016, and Ph.D. in Computer Science from The Catholic University of America in 2021. Currently, he is a software engineer at UiPath Inc., Bellevue, WA. His research interests include wireless communications and networking, distributed systems, edge-cloud network architecture, resource management for Internet of Things, mobile edge computing, deep reinforcement learning, software-defined networks, and 5G/6G mobile networks.

Hang Liu joined the Catholic University of America in 2013, where he currently is a Professor with the Department of Electrical Engineering and Computer Science. Prior to joining the Catholic University, he had more than 10 years of research experience in networking industry and worked in senior research and management positions at several companies. He has published more than 100 papers in leading journals and conferences and holds over 50 US patents. He received two best paper awards and one best student paper award. He has also made many contributions to the IEEE 802 wireless standards and 3GPP standards, and was the editor of the IEEE 802.11aa standard and the rapporteur of a 3GPP work item. He received his Ph.D. degree in Electrical Engineering from the University of Pennsylvania. His research interests include wireless communications and networking, millimeter wave communications, dynamic spectrum management, mobile computing, Internet of Things, future Internet architecture and protocols, mobile content distribution, video streaming, and network security.

Yang Guo is a senior computer scientist at the Computer Security Division, National Institute of Standards and Technology (NIST), Gaithersburg, MD. His research interests span broadly over the distributed systems and networking, with a focus on Software Defined Networking, Cybersecurity, and AI/ML. He obtained Ph.D. from University of Massachusetts at Amherst in 2000, and B.S. and M.S. from Shanghai Jiao Tong University. Before joining the NIST, he was a researcher at Bell Labs from 2010 to 2015 and was a Principal Scientist at Technicolor Corporate Research from 2005 to 2010. At NIST, he leads the research on Software Defined Networking (SDN) and the High-Performance Computing (HPC) Security Working Group. He has published over 80 peer-reviewed papers in renowned technical journals and conferences and holds 18 US patents. He received Bell Labs' teamwork award, ICC Best Paper award, and was on Technicolor's Fellowship Network as a technical leader.

Robert (Bob) Gazda is a Senior Director in InterDigital's Wireless Networking Lab. He is an accomplished engineering professional and technologist with over 20 years of industry experience in wireless telecommunications, networking, and embedded systems. Currently at InterDigital, Bob is leading research and innovation focused on 5G and 6G Distributed and Converged Computing and Communications Architectures. Bob holds a Master, Software Engineering (MSE) degree from Carnegie Mellon University, a MS in Computer Engineering degree from Villanova University, and a B.S. in Electrical Engineering from Drexel University. His research interests include wireless communications and networking, mobility management, edge & distributed computing, and real-time systems.

Authorized licensed use limited to: NIST Virtual Library (NVL). Downloaded on August 25,2023 at 18:48:25 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.