Fast Adaptive Task Offloading in Edge Computing based on Meta Reinforcement Learning

Jin Wang, Jia Hu, Geyong Min, Albert Y. Zomaya, *Fellow, IEEE*, and Nektarios Georgalas

Abstract—Multi-access edge computing (MEC) aims to extend cloud service to the network edge to reduce network traffic and service latency. A fundamental problem in MEC is how to efficiently offload heterogeneous tasks of mobile applications from user equipment (UE) to MEC hosts. Recently, many deep reinforcement learning (DRL) based methods have been proposed to learn offloading policies through interacting with the MEC environment that consists of UE, wireless channels, and MEC hosts. However, these methods have weak adaptability to new environments because they have low sample efficiency and need full retraining to learn updated policies for new environments. To overcome this weakness, we propose a task offloading method based on meta reinforcement learning, which can adapt fast to new environments with a small number of gradient updates and samples. We model mobile applications as Directed Acyclic Graphs (DAGs) and the offloading policy by a custom sequence-to-sequence (seq2seq) neural network. To efficiently train the seq2seq network, we propose a method that synergizes the first order approximation and clipped surrogate objective. The experimental results demonstrate that this new offloading method can reduce the latency by up to 25% compared to three baselines while being able to adapt fast to new environments.

Index Terms—Multi-access edge computing, task offloading, meta reinforcement learning, deep learning

✦

1 INTRODUCTION

Recent years have witnessed the rapid advance of new computing and communication technologies, driving the increasing emergence of innovative mobile applications and services, such as augmented reality, virtual reality, face recognition, and mobile healthcare. These mobile applications introduce a significant surge in demands for computing and storage resources that are often provided by cloud servers. This situation generates huge network traffic between cloud and users, thus placing a heavy burden on the backhaul links and causing high service latency. Multi-access Edge Computing (MEC) [1] was recently introduced as a key technology to address this problem. The underlying principle of MEC is to extend cloud computing capabilities to MEC host at the network edge close to users, which can significantly alleviate network congestion and reduce service latency.

One of the key functionalities of MEC is task offloading (aka, computation offloading), which enables to offload computation-intensive tasks of mobile applications from user equipment (UE) to MEC host at the network edge. In real-world scenarios, many mobile applications (e.g., face recognition [2], gesture recognition [2], and augmented

reality [3]) are composed of dependent tasks, which can be modelled as a Directed Acyclic Graph (DAG). Thus, offloading dependent tasks in a DAG with the minimum latency is a crucial problem in MEC. Since this problem is NP-hard, many existing solutions are based on heuristic or approximation algorithms [4]–[6]. However, these solutions rely heavily on expert knowledge or accurate mathematical models for the MEC system. Whenever the environment of the MEC system changes, the expert knowledge or mathematical models may need to be updated accordingly. Therefore, it is difficult for one specific heuristic/approximation algorithm to fully adapt to the dynamic MEC scenarios arisen from the increasing complexity of applications and architectures of MEC.

Deep Reinforcement Learning (DRL), which combines reinforcement learning (RL) with Deep Neural Network (DNN), provides a promising solution to the above challenge, because DRL can learn to solve complex problems such as games [7], robotics [8], and traffic scheduling [9] by trial and error without accurate models for the environment. More recently, researchers studied the application of DRL to various MEC task offloading problems [10]–[13]. They considered the MEC system including UE, wireless channels, and MEC host as one stationary RL environment and learn an offloading policy through interacting with the environment. However, these methods have weak adaptability for unexpected perturbations or unseen situations (i.e., new environments) like changes of applications, task numbers, or data rates. Because they have low sample efficiency and need full retraining to learn an updated policy for the new environment, they are time-consuming.

Meta learning [14] is a promising method to address the aforementioned issues by leveraging previous experiences

- *Jin Wang, Jia Hu, and Geyong Min are with the Department of Computer Science, University of Exeter, United Kingdom.*
  E-mail: {jw855, j.hu, g.min}@exeter.ac.uk
- *Albert Y. Zomaya is with the School of Information Technologies, The University of Sydney, Australia.*
  E-mail: albert.zomaya@sydney.edu.au
- *Nektarios Georgalas is with Applied Research Department, British Telecom, United Kingdom.*
  E-mail: nektarios.georgalas@bt.com
- *Corresponding authors: Jia Hu and Geyong Min.*

across a range of learning tasks to significantly accelerate learning of new tasks. In the context of RL problems, meta reinforcement learning (MRL) aims to learn policies for new tasks within a small number of interactions with the environment by building on previous experiences. In general, MRL conducts two "loops" of learning, an "outer loop" which uses its experiences over many task contexts to gradually adjust parameters of the meta policy that governs the operation of an "inner loop". Based on the meta policy, the "inner loop" can adapt fast to new tasks through a small number of gradient updates [15].

There are significant benefits of adapting MRL to solving the computation offloading problem. Firstly, specific policies for new mobile users can be fast learned based on their local data and the meta policy. Secondly, MRL training in the MEC system can leverage resources from both the MEC host and UE. More specifically, training for the meta policy (outer loop) is run on the MEC host and training for the specific offloading policy (inner loop) is processed on UE. Normally, the "inner loop" training only needs several training steps and a small amount of sampling data, thus the UE with limited computation resources and data is able to complete the training process. Finally, MRL can significantly improve the training efficiency in learning new tasks and make the offloading algorithm more adaptive to the dynamic MEC environment.

In this paper, we propose an MRL-based method that synergizes the first-order MRL algorithm with a sequence-to-sequence (seq2seq) neural network. The proposed method learns a meta offloading policy for all UE and fast obtains the effective policy for each UE based on the meta policy and local data. To evaluate the performance of the MRLCO under dynamic scenarios, we consider the following scenarios: 1) Heterogeneous users with personal preferences of mobile applications which are represented as DAGs with different heights, widths, and task numbers. 2) Varying transmission rates according to the distance between the UE and the MEC host.

The major contributions of this paper can be summarized as follows:

- This paper is the first of its kind to propose an MRL-based method (MRLCO) to address the computation offloading problem, achieving fast adaptation to dynamic offloading scenarios. MRLCO has high sample efficiency towards new learning tasks, thus it enables UE to run the training process by using its own data even with limited computation resources.
- We propose a new idea to model the dynamic computation offloading process as multiple MDPs, where the learning of offloading policies is decomposed into two parts: effectively learning a meta policy among different MDPs, and fast learning a specific policy for each MDP based on the meta policy.
- We convert the offloading decision process as a sequence prediction process and design a custom seq2seq neural network to represent the offloading policy. An embedding method is also proposed to embed the vertices of a DAG considering both its task profiles and dependencies. In addition, we propose a new training method which combines the first-order

approximation and clipped surrogate objective to stabilize the training of the seq2seq neural network.
- We conduct simulation experiments using generated synthetic DAGs according to real-world applications, covering a wide range of topologies, task numbers, and transmission rates. The results show that MRLCO achieves the lowest latency within a small number of training steps compared to three baseline algorithms including a fine-tuning DRL method, a greedy algorithm, and a heterogeneous earliest finish time (HEFT) based heuristic algorithm.

The rest of the paper is organised as follows. A brief introduction to MEC, RL, and MRL is given in Section 2. The problem formulation for task offloading is presented in Section 3. The details of the MRLCO are described in Section 4. Evaluation results are presented and discussed in Section 5. The related work is reviewed in Section 6. We discuss the MRLCO and its future work in Section 7. Finally, Section 8 concludes the paper.

2 BACKGROUND

This section briefly introduces the background related to MEC, RL, and MRL.

2.1 Multi-access Edge Computing

Over recent years, MEC has been acknowledged as one of the emerging network paradigms, which can release the pressure introduced by an unprecedented increase in traffic volume and computation demands nowadays through enabling cloud services to the network edge. Typically, MEC hosts coupled with computation and storage resources are deployed in the network edge, supporting intensive computation and data processing. As such, MEC can alleviate the burden of backhaul links and cut down the service latency. MEC is beneficial to a wide variety of emerging applications that require high volume data and low latency, e.g., autonomous driving, augmented reality, and digital healthcare.

In practice, many mobile applications are composed of multiple tasks with inner dependencies among them, which can be offloaded to MEC hosts for processing. Specifically, the objective of task offloading is to find the optimal policy to partition an application into two groups of computation tasks with one executed on the UE and the other offloaded to an MEC host so that the total running cost is minimal.

2.2 Reinforcement Learning

RL considers learning from environment so as to maximize the accumulated reward. Formally, a learning task, $\mathcal{T}$, is modelled as an MDP, which is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{P}_0, \mathcal{R}, \gamma)$. Here, $\mathcal{S}$ is the state space, $\mathcal{A}$ denotes the action space, $\mathcal{R}$ is a reward function, $\mathcal{P}$ is the state-transition probabilities matrix, $\mathcal{P}_0$ is the initial state distribution, and $\gamma \in [0, 1]$ is the discount factor. A policy $\pi(a|s)$, where $a \in \mathcal{A}$ and $s \in \mathcal{S}$, is a mapping from state s to the probability of selecting action a. We define the trajectories sampled from the environment according to the policy π as $\tau_\pi = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots)$, where $s_0 \sim \mathcal{P}_0$, $a_t \sim \pi(\cdot|s_t)$ and r_t is a reward at time step t.

The state value function of a state s_t under a parameterized policy $\pi(a|s;\theta)$, denoted as $v_\pi(s_t)$, is the expected return when starting in s_t and following $\pi(a|s;\theta)$ thereafter. Here, θ is the vector of policy parameters and $v_\pi(s_t)$ can be calculated by

$$v_\pi(s_t) = \mathbb{E}_{\tau \sim P_\mathcal{T}(\tau|\theta)} \left[\sum_{k=t} \gamma^{k-t} r_k \right] \tag{1}$$

, where $P_\mathcal{T}(\tau|\theta)$ is the probability distribution of sampled trajectories based on $\pi(a|s;\theta)$. The goal for RL is to find an optimal parameterized policy $\pi(a|s;\theta^*)$ to maximize the expected total rewards $J = \sum_{s_0 \sim \mathcal{P}_0} v_\pi(s_0)$.

2.3 Meta Reinforcement Learning

MRL enhances the conventional RL methods with meta learning, which aims to learn a learning algorithm that can quickly find the policy for a learning task $\mathcal{T}_i$ drawn from a distribution of tasks $\rho(\mathcal{T})$. Each learning task $\mathcal{T}_i$ corresponds to a different MDP, and these learning tasks typically share the same state and action spaces but may differ in reward functions or their dynamics (i.e., $\mathcal{P}$ and $\mathcal{P}_0$).

Recent years have brought a wealth of methods focused on different aspects of MRL. One typical example is gradient-based MRL, which aims to learn the initial parameters θ of a policy neural network, so that performing a single or few steps of policy gradient over θ with a given new task can lead to an effective policy for that task. We follow the formulation of model-agnostic meta-learning (MAML) [16], giving the target of gradient-based MRL as

$$J(\theta) = \mathbb{E}_{\mathcal{T}_i \sim \rho(\mathcal{T})} \left[J_{\mathcal{T}_i}(\theta') \right], \text{with } \theta' := U(\theta, \mathcal{T}_i), \tag{2}$$

where $J_{\mathcal{T}_i}$ denotes the objective function of task $\mathcal{T}_i$. For example, when using vanilla policy gradient (VPG), $J_{\mathcal{T}_i}(\theta) = \mathbb{E}_{\tau \sim P_{\mathcal{T}_i}(\tau|\theta)} \sum_{t=0} (\gamma^t r_t - b(s_t))$, where $b(s_t)$ denotes an arbitrary baseline which does not vary with a_t. U denotes the update function which depends on $J_{\mathcal{T}_i}$ and the optimization method. For instance, if we conduct k-step gradient ascent for $\mathcal{T}_i$, then $U(\theta, \mathcal{T}_i) = \theta + \alpha \sum_{t=1}^{k} g_t$, where g_t denotes the gradient of $J_{\mathcal{T}_i}$ at time step t and α is the learning rate. Therefore, the optimal parameters of policy network and update rules are

$$\theta^* = \arg\max_\theta \mathbb{E}_{\mathcal{T}_i \sim \rho(\mathcal{T})} \left[J_{\mathcal{T}_i}(U(\theta, \mathcal{T}_i)) \right],$$
$$\theta \leftarrow \theta + \beta \mathbb{E}_{\mathcal{T}_i \sim \rho(\mathcal{T})} \left[\nabla_\theta J_{\mathcal{T}_i}(U(\theta, \mathcal{T}_i)) \right], \tag{3}$$

where β is the learning rate of "outer loop" training. The gradient-based MRL has good generalization ability. However, the second-order derivative in MAML may bring huge computation cost during training, which is inefficient. In addition, when combining with a complex neural network architecture, e.g., a seq2seq neural network, the implementation of second-order MAML becomes intractable. To address these challenges, some algorithms [16], [17] use the first-order approximation to MAML target. In this work, we implement MRLCO based on the first-order MRL due to its low computation cost, good performance, and easy implementation when combing with a seq2seq neural network.

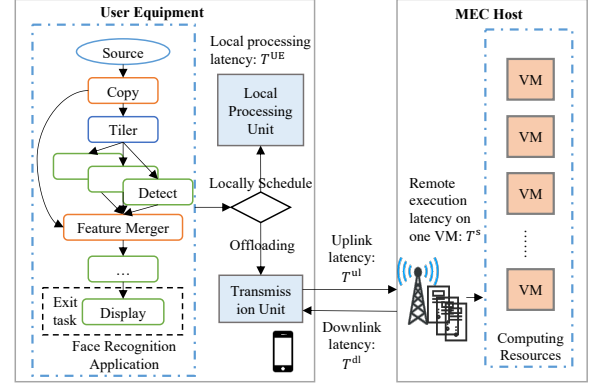Fig. 1. An example of computation offloading in MEC.

3 PROBLEM FORMULATION

Fig. 1 gives an example of computation offloading in MEC. This example considers a real-world application—face recognition, which consists of dependent tasks such as tiler, detection, or feature mergence [2]. The UE makes offloading decisions for those tasks according to the system status and task profiles, thus some tasks are run locally on the UE while others are offloaded to the MEC host via wireless channels. In general, each MEC host runs multiple virtual machines (VMs) processing the tasks. In this work, we consider that each UE is associated with a dedicated VM providing private computing, communications and storage resources to the UE, the same as in works [18], [19]. The computation capacity (i.e., the number of CPU cores times the clock speed of each core) of an MEC host is denoted as f_s. We consider an equal resource allocation for VMs, i.e., all VMs evenly share the computing resource of the MEC host. Therefore, assuming there are k users in the MEC systems, the computation capacity for each VM is $f_{vm} = f_s/k$. Formally, we model mobile applications as DAGs, $G = (T, E)$, where the vertex set T represents the tasks and the directed edge set E represents the dependencies among tasks, respectively. Each directed edge is denoted by $\overrightarrow{e} = (t_i, t_j)$, corresponding to the dependency between task t_i and t_j, where t_i is an immediate parent task of t_j, and t_j is an immediate child task of t_i. With the constraint of dependency, a child task cannot be executed until all of its parent tasks are completed. In $G = (T, E)$, we call a task without any child task as an *exit task*.

In computation offloading, a computation task can either be offloaded to the MEC host or executed locally on the UE. If task t_i is offloaded, there are three steps to execute t_i. First, the UE sends t_i to an MEC host through a wireless channel. Second, the MEC host runs the received task. Finally, the running result of t_i is returned to the UE. The latency at each step is related to the task profile and the MEC system state. Here, the task profile of t_i includes required CPU cycles for running the task, C_i, data sizes of the task sent, data_i^s, and the result received, data_i^r. Besides, the MEC system state contains the transmission rate of wireless uplink channel, R_{ul}, and rate of downlink channel, R_{dl}. Therefore, the latency for sending data, T_i^{ul}, executing on the MEC host, T_i^s, and

receiving result, T_i^{dl}, of task t_i can be calculated as:

$$T_i^{\text{ul}} = \text{data}_i^s/R_{\text{ul}}, \quad T_i^s = C_i/f_{\text{vm}}, \quad T_i^{\text{dl}} = \text{data}_i^r/R_{\text{dl}}. \quad (4)$$

If task t_i runs locally on the UE, there is only running latency on the UE, which can be obtained by $T_i^{\text{UE}} = C_i/f_{\text{UE}}$ where f_{UE} denotes the computation capacity of the UE. The end-to-end latency of a task offloading process includes local processing, uplink, downlink, and remote processing latency, as shown in Fig. 1.

The scheduling plan for a DAG, $G = (T, E)$, is denoted as $A_{1:n} = \{a_1, a_2, \ldots, a_n\}$, where $|T| = n$ and a_i represents the offloading decision of t_i. Tasks are scheduled in a sequence based on the scheduling plan, where all parent tasks are scheduled before their child tasks. We denote FT_i^{ul}, FT_i^s, FT_i^{dl}, and FT_i^{UE} as the finish time of task t_i on the uplink wireless channel, the MEC host, the downlink wireless channel, and the UE, respectively. We also denote the available time of these resources when scheduling task t_i as $\mathcal{M}_i^{\text{ul}}$, $\mathcal{M}_i^s$, $\mathcal{M}_i^{\text{dl}}$, and $\mathcal{M}_i^{\text{UE}}$. The resource available time depends on the finish time of the task scheduled immediately before t_i on that resource. If the task scheduled immediately before t_i does not utilize the resource, we set the finish time on the resource as 0.

If task t_i is offloaded to the MEC host, t_i can only start to send its data when its parent tasks are all completed and the uplink channel is available. Therefore, the finish time on the uplink channel, FT_i^{ul}, can be defined by

$$FT_i^{\text{ul}} = \max\left\{\mathcal{M}_i^{\text{ul}}, \max_{j \in \text{parent}(t_i)}\left\{FT_j^{\text{UE}}, FT_j^{\text{dl}}\right\}\right\} + T_i^{\text{ul}},$$
$$\mathcal{M}_i^{\text{ul}} = \max\left\{\mathcal{M}_{i-1}^{\text{ul}}, FT_{i-1}^{\text{ul}}\right\}. \quad (5)$$

Similarly, the finish time of t_i on the MEC host, FT_i^s, and that on the downlink channel, FT_i^{dl}, are given by

$$FT_i^s = \max\left\{\mathcal{M}_i^s, \max\left\{FT_i^{\text{ul}}, \max_{j \in \text{parent}(t_i)} FT_j^s\right\}\right\} + T_i^s,$$
$$FT_i^{\text{dl}} = \max\left\{\mathcal{M}_i^{\text{dl}}, FT_i^s\right\} + T_i^{\text{dl}},$$
$$\mathcal{M}_i^s = \max\left\{\mathcal{M}_{i-1}^s, FT_{i-1}^s\right\},$$
$$\mathcal{M}_i^{\text{dl}} = \max\left\{\mathcal{M}_{i-1}^{\text{dl}}, FT_{i-1}^{\text{dl}}\right\}. \quad (6)$$

If t_i is scheduled on the UE, the start time of t_i depends on the finish time of its parent tasks and the available time of the UE. Formally, the finish time of t_i on the UE, FT_i^{UE}, is defined as

$$FT_i^{\text{UE}} = \max\left\{\mathcal{M}_i^{\text{UE}}, \max_{j \in \text{parent}(t_i)}\left\{FT_j^{\text{UE}}, FT_j^{\text{dl}}\right\}\right\} + T_i^{\text{UE}},$$
$$\mathcal{M}_i^{\text{UE}} = \max\left\{\mathcal{M}_{i-1}^{\text{UE}}, FT_{i-1}^{\text{UE}}\right\}. \quad (7)$$

Overall, the objective is to find an effective offloading plan for the DAG to obtain the minimal total latency. Formally, the total latency of a DAG given a scheduling plan $A_{1:n}$, $T_{A_{1:n}}^c$, is given by

$$T_{A_{1:n}}^c = \max\left[\max_{t_k \in \mathcal{K}}\left(FT_k^{\text{UE}}, FT_k^{\text{dl}}\right)\right], \quad (8)$$

where $\mathcal{K}$ is the set of exit tasks. The problem in Eq. (8) is NP-hard, so finding the optimal offloading plan can be extremely challenging due to the highly dynamic DAG topologies and MEC system states. In the next section, we present the details of MRLCO for handling this problem.

TABLE 1
Summary of Main Notations

Notation	Description
$\mathbb{E}$	Mean value
data_i^s, data_i^r	Size of data sending to or receiving from a task t_i
R_{ul}, R_{dl}	Transmission rate of uplink and downlink
f_{UE}, f_s, f_{vm}	Computation capacity of UE, MEC host and VM
T_i^{ul}, T_i^s, T_i^{dl}, T_i^{UE}	Latency for task t_i on uplink channel, MEC host, downlink channel, and UE.
FT_i^{ul}, FT_i^s, FT_i^{dl}, FT_i^{UE}	Finish time for task t_i on uplink channel, MEC host, downlink channel, and UE
$\mathcal{M}_i^{\text{ul}}$, $\mathcal{M}_i^s$, $\mathcal{M}_i^{\text{dl}}$, $\mathcal{M}_i^{\text{UE}}$	Resource available time for task t_i on uplink channel, MEC host, downlink channel, and UE
$A_{1:n}$	Computational offloading plan for n tasks
$\mathcal{T}_i, \rho(\mathcal{T})$	A learning task and distribution of learning tasks
s_i, a_i, r_i	State, action, and reward of an MDP at time step i
$\pi(a\|s;\theta), v(s;\theta)$	Parametrized policy and value function for computation offloading.
τ_π	Trajectories sampled from the environment based on the policy π.
f_{enc}, f_{dec}	Functions of encoder and decoder
e_i, d_i	Output of encoder and decoder at time step i
c_i	Context vector at decoding step i
$\hat{A}_t$	Advantage function at time step t
$U(\theta, \mathcal{T}_i)$	Update function (e.g., Adam) for the learning task

4 MRLCO: AN MRL-BASED COMPUTATION OFFLOADING SOLUTION

In this section, we first give an overview of the architecture of the MRLCO and explain how it works with the MEC system. Next, we present the detailed MDP modelling for the computation offloading problem. Finally, we describe the implementation of the MRLCO algorithm.

4.1 The MRLCO Empowered MEC System Architecture

The MRLCO aims to leverage the computation resources from both the UE and the MEC host for the training process. There are two loops of training — "inner loop" training for the task-specific policy and "outer loop" training for the meta policy. The "inner loop" training is conducted on the UE while the "outer loop" training on the MEC host.

Fig. 2 shows an architecture that integrates the MRLCO into an emerging MEC system [1] composed of the user level, edge level, and remote level. Here, the user level includes heterogeneous UE, the edge level contains MEC hosts that provide edge computing services, and the remote level consists of cloud servers. Specifically, mobile users communicate with an MEC host through the local *Transmission unit*. The MEC host incorporates an MEC platform and a virtualization infrastructure that provides the computing, storage, and network resources. The MEC platform provides *Traffic management* (i.e., traffic rules control and domain name handling) and offers edge services. The five key modules of MRLCO (*parser*, *local trainer*, *offloading scheduler*, *global training service*, and *remote execution service*) can be deployed at the user and edge levels of the MEC system separately, as described below:

- At the user level, the *parser* aims to convert mobile applications into DAGs. The *local trainer* is responsible for the "inner loop" training, which receives
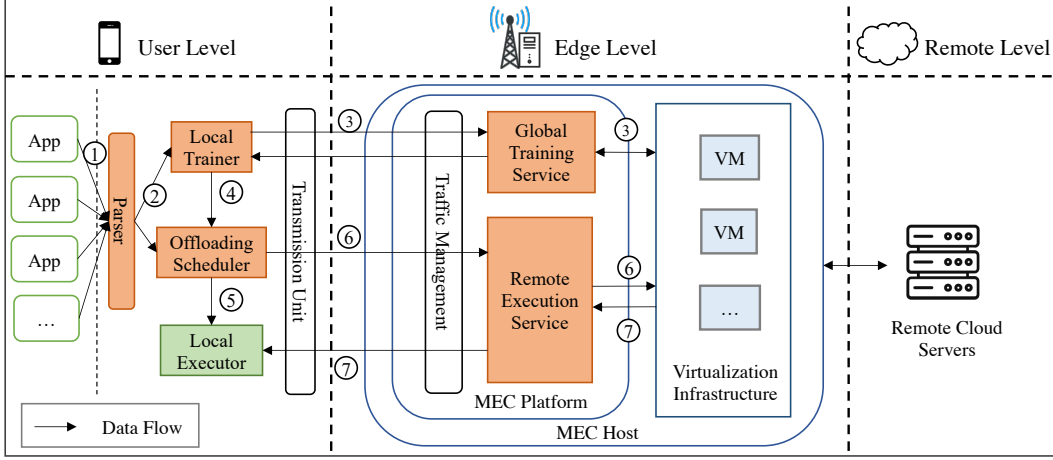
Fig. 2. The system architecture of the MRLCO empowered MEC system. The data flows in this architecture include: ① mobile applications, ② parsed DAGs, ③ parameters of the policy network, ④ the trained policy network, ⑤ tasks scheduled to local executor, ⑥ tasks offloaded to the MEC host, ⑦ results from the offloaded tasks.

the parsed DAGs from the *parser* as training data and uploads/downloads parameters of the policy network to/from the MEC host through local transmission unit. Once the training process is finished, the trained policy network will be deployed to the *offloading scheduler* that is used to make offloading decisions through policy network inference. After making decisions for all tasks of a DAG, the locally scheduled tasks will run on the local executor and the offloaded tasks will be sent to the MEC host for execution.

- At the edge level, the *global training service* and *remote execution service* modules are deployed to the MEC platform. The *global training service* is used to manage the "outer loop" training, which sends/receives parameters of the policy network to/from the UE and deploys the global training process on the virtualization infrastructure in the MEC host. The *remote execution service* is responsible for managing the tasks offloaded from the UE, assigning these tasks to associated VMs, and sending the results back to the UE.

Next, we describe the detailed training process of the MRLCO in the MEC system, as shown in Fig. 3. The training process for MRLCO includes four steps. First, the UE downloads the parameters of the meta policy from the MEC host. Next, an "inner loop" training is run on every UE based on the meta policy and the local data, in order to obtain the task-specific policy. The UE then uploads the parameters of the task-specific policy to the MEC host. Finally, the MEC host conducts an "outer loop" training based on the gathered parameters of task-specific policies, generates the new meta policy, and starts a new round of training. Once obtaining the stable meta policy, we can leverage it to fast learn a task-specific policy for new UE through "inner loop" training. Notice that the "inner loop" training only needs few training steps and a small amount of data, thus can be sufficiently supported by the UE. We will present the algorithmic details of the "outer loop" and "inner loop" training in Section 4.3.

4.2 Modelling the Computation Offloading Process as Multiple MDPs

To adapt MRL to solve the computation offloading problem, we firstly model the process of computation offloading under various MEC environments as multiple MDPs, where learning an effective offloading policy for one MDP is considered as a learning task. Formally, we consider a distribution over all learning tasks in MEC as $\rho(\mathcal{T})$, where each task $\mathcal{T}_i \sim \rho(\mathcal{T})$ is formulated as a different MDP, $\mathcal{T}_i = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{P}_0, \mathcal{R}, \gamma)$. (Please refer to Section 2.2 for the meaning of these notations.) In order to obtain the adaptive offloading policy for all learning tasks, we decompose the learning process into two parts: effectively learning a meta policy among all MDPs and fast learning a specific offloading policy for one MDP based on the meta policy. The definitions of the state, action, and reward for the MDP are listed as follows:

- **State:** When scheduling a task t_i, the latency of running the task depends on the task profile (i.e., required CPU cycles, data sizes), DAG topologies, the wireless transmission rate, and the state of MEC resources. According to Eqs. (5), (6), and (7), the state of MEC resources is related to the offloading decisions of task scheduled before t_i. Therefore, we define the state as a combination of the encoded DAG and the partial offloading plan:

$$\mathcal{S} := \{s_i | s_i = (G(T, E), A_{1:i})\} \quad \text{where} \quad i \in [1, |T|], \quad (9)$$

where $G(T, E)$ is comprised of a sequence of task embeddings and $A_{1:i}$ is the partial offloading plan for the first i tasks. To convert a DAG into a sequence of task embeddings, we first sort and index tasks according to the ascending order of the rank value of each task, which is defined as

$$rank(t_i) = \begin{cases} T_i^o & \text{if } t_i \in \mathcal{K}, \\ T_i^o + \max_{t_j \in child(t_i)} (rank(t_j)) & \text{if } t_i \notin \mathcal{K}, \end{cases} \quad (10)$$

where $T_i^o = T_i^{ul} + T_i^s + T_i^{dl}$ denotes the latency for task i from starting offloading to finishing execution,
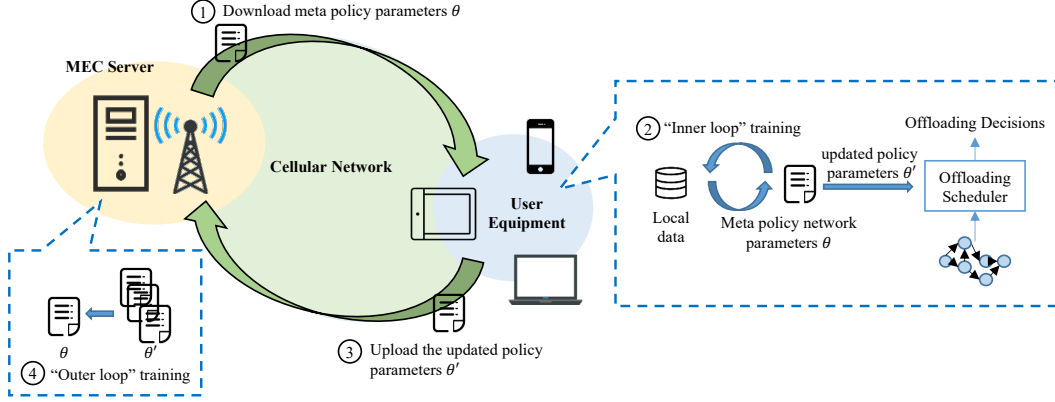
Fig. 3. The training process of the MRLCO empowered MEC system includes four steps: 1) the UE downloads the parameters of meta policy, θ, from the MEC host; 2) "inner loop" training is conducted on the UE based on θ and the local data, obtaining the parameters of task-specific policy, θ'; 3) the UE uploads θ' to the MEC host; 4) the MEC host conducts "outer loop" training based on the gathered updated parameters θ'.

$child(t_i)$ represents the set of immediate child tasks of t_i. Each task is converted into an embedding that consists of three elements: 1) a vector that embeds the current task index and the normalized task profile, 2) a vector that contains the indices of the immediate parent tasks, 3) a vector that contains the indices of the immediate child tasks. The size of vectors that embed parent/child task indices is limited to p. We pad the vector with -1, in case the number of child/parent tasks is less than p.

- **Action:** The scheduling for each task is a binary choice, thus the action space is defined as $\mathcal{A} := \{0, 1\}$, where 0 stands for execution on the UE and 1 represents offloading.

- **Reward:** The objective is to minimize $T^c_{A_{1:n}}$ given by Eq. (8). In order to achieve this goal, we define the reward function as the estimated negative increment of the latency after making an offloading decision for a task. Formally, when taking action for the task t_i, the increment is defined as $\Delta T^c_i = T^c_{A_{1:i}} - T^c_{A_{1:i-1}}$.

Based on the above MDP definition, we denote the policy when scheduling t_i as $\pi(a_i | G(T, E), A_{1:i-1})$. For a DAG with n tasks, let $\pi(A_{1:n} | G(T, E))$ denote the probability of having the offloading plan $A_{1:n}$ given the graph $G(T, E)$. Therefore, $\pi(A_{1:n} | G(T, E))$ can be obtained by applying chain rules of probability on each $\pi(a_i | G(T, E), A_{1:i-1})$ as

$$\pi(A_{1:n} | G(T, E)) = \prod_{i=1}^{n} \pi(a_i | G(T, E), A_{1:i-1}). \tag{11}$$

A seq2seq neural network [20] is a natural choice to represent the policy defined in Eq. (11). Fig. 4 shows our design of a custom seq2seq neural network, which can be divided into two parts: encoder and decoder. In our work, both encoder and decoder are implemented by recurrent neural networks (RNN). The input of the encoder is the sequence of task embeddings, $[t_1, t_2, ..., t_n]$, while the output of the decoder is the offloading decisions of each tasks, $[a_1, a_2, ..., a_n]$. To improve the performance, we include the attention mechanism [20] into our custom seq2seq neural network. Attention mechanism allows the decoder to attend to different parts of the source sequence (i.e., the input sequence of the encoder) at each step of the output generation,
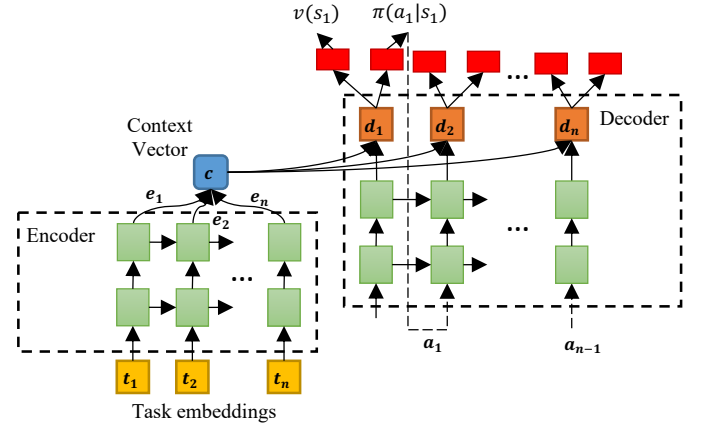

Fig. 4. Architecture of the seq2seq neural network in MRLCO. The architecture consists of an encoder and a decoder, where the input of the encoder is the sequence of task embeddings and the output of the decoder is used to generate both policy and value function.

thus it can alleviate the issue of information loss caused by the original seq2seq neural network that encodes the input sequence into a vector with fixed dimensions.

Formally, we define the functions of the encoder and decoder as f_{enc} and f_{dec}, respectively. In our work, we use the Long Short-Term Memory (LSTM) as f_{enc} and f_{dec}. At each step of encoding, the output of the encoder, e_i, is obtained by

$$e_i = f_{enc}(t_i, e_{i-1}). \tag{12}$$

After encoding all the input task embeddings, we have the output vector as $\mathbf{e} = [e_1, e_2, ..., e_n]$. At each decoding step, we define the output of the decoder, d_j, as

$$d_j = f_{dec}(d_{j-1}, a_{j-1}, c_j), \tag{13}$$

where c_j is the context vector at decoding step j and is computed as a weighted sum of the outputs of the encoder:

$$c_j = \sum_{i=0}^{n} \alpha_{ji} e_i. \tag{14}$$

The weight α_{ji} of each output of encoder, e_i, is computed by

$$\alpha_{ji} = \frac{\exp\left(\text{score}(d_{j-1}, e_i)\right)}{\sum_{k=1}^{n} \exp\left(\text{score}(d_{j-1}, e_k)\right)}, \tag{15}$$

where the score function, score(d_{j-1}, e_i), is used to measure how well the input at position i and the output at position j match. We define the score function as a trainable feedforward neural network according to the work [20]. We use the seq2seq neural network to approximate both policy $\pi(a_j|s_j)$ and value function $v_\pi(s_j)$ by passing the output of decoder $\mathbf{d} = [d_1, d_2, ..., d_n]$ to two separate fully connected layers. Notice that the policy and value function share most of the parameters (i.e., the encoder and decoder) which are used to extract common features of DAGs (e.g., the graph structure and task profiles). Therefore, training the policy can accelerate the training of value function and vice versa. During training for the seq2seq neural network, the action a_j is generated through sampling from the policy $\pi(a_j|s_j)$. Once the training is finished, the offloading decisions for a DAG can be made by inference through the seq2seq neural network, where the action a_j is generated by $a_j = \arg\max_{a_j} \pi(a_j|s_j)$. Therefore, the time complexity for our algorithm is the same as the inference of the seq2seq neural network with attention, which is $O(n^2)$ [21]. Normally, the task number, n, of a mobile application is less than 100 [4], [5], [22], thus the time complexity of the MRLCO is feasible.

4.3 Implementation of MRLCO

MRLCO shares a similar algorithm structure with gradient-based MRL algorithms, which consists of two loops for training. Instead of using VPG as the policy gradient method for the "inner loop" training [16], we define our objective function based on Proximal Policy Optimization (PPO) [23]. Compared to VPG, PPO achieves better exploring ability and training stability. For one learning task $\mathcal{T}_i$, PPO generates trajectories using the sample policy $\pi_{\theta_i^o}$ and updates the target policy π_{θ_i} for several epochs, where θ_i equals θ_i^o at the initial epoch. In order to avoid a large update of the target policy, PPO uses a clipped surrogate objective as

$$J_{\mathcal{T}_i}^{C}(\theta_i) = \mathbb{E}_{\tau \sim P_{\mathcal{T}_i}(\tau, \theta_i^o)} \left[\sum_{t=1}^{n} \min\left(\Pr_t \hat{A}_t, \mathrm{clip}_{1-\epsilon}^{1+\epsilon}(\Pr_t)\hat{A}_t \right) \right]. \quad (16)$$

Here, θ_i^o is the vector of parameters of the sample policy network. $\Pr_t$ is the probability ratio between the sample policy and target policy, which is defined as

$$\Pr_t = \frac{\pi_{\theta_i}(a_t|G(T, E), A_{1:t})}{\pi_{\theta_i^o}(a_t|G(T, E), A_{1:t})}. \quad (17)$$

The clip function $\mathrm{clip}_{1-\epsilon}^{1+\epsilon}(\Pr_t)$ aims to limit the value of $\Pr_t$, in order to remove the incentive for moving $\Pr_t$ outside of the interval $[1-\epsilon, 1+\epsilon]$. $\hat{A}_t$ is the advantage function at time step t. Specially, we use general advantage estimator (GAE) [24] as our advantage function, which is defined by

$$\hat{A}_t = \sum_{k=0}^{n-t+1} (\gamma\lambda)^k (r_{t+k} + \gamma v_\pi(s_{t+k+1}) - v_\pi(s_{t+k})), \quad (18)$$

where $\lambda \in [0, 1]$ is used to control the trade-off between bias and variance. The value function loss is defined as

$$J_{\mathcal{T}_i}^{\mathrm{VF}}(\theta_i) = \mathbb{E}_{\tau \sim P_{\mathcal{T}_i}(\tau, \theta_i^o)} \left[\sum_{t=1}^{n} (v_\pi(s_t) - \hat{v}_\pi(s_t))^2 \right], \quad (19)$$

where $\hat{v}_\pi(s_t) = \sum_{k=0}^{n-t+1} \gamma^k r_{t+k}$.

Algorithm 1 : Meta Reinforcement Learning based Computation Offloading

---

Require: Task distribution $\rho(\mathcal{T})$,
1: Randomly initialize the parameters of meta policy, θ
2: **for** iterations $k \in \{1, ..., K\}$ **do**
3:     Sample n tasks $\{\mathcal{T}_0, \mathcal{T}_1, ..., \mathcal{T}_n\}$ from $\rho(\mathcal{T})$
4:     **for** each task $\mathcal{T}_i \in \{\mathcal{T}_0, \mathcal{T}_1, ..., \mathcal{T}_n\}$ **do**
5:         Initialize $\theta_i^o \leftarrow \theta$ and $\theta_i \leftarrow \theta$
6:         Sample trajectories set $D = (\tau_1, \tau_2, ...)$ from $\mathcal{T}_i$ using sample policy $\pi_{\theta_i^o}$
7:         Compute the policy network parameters $\theta_i' \leftarrow \theta_i + \alpha\nabla_{\theta_i} J_{\mathcal{T}_i}^{\mathrm{PPO}}(\theta_i)$ via m steps of *Adam* with D
8:     **end for**
9:     Update $\theta \leftarrow \theta + \beta g^{\mathrm{MRLCO}}$ via *Adam*
10: **end for**

Overall, we combine Eq. (16) and Eq. (19), defining the objective function for each "inner loop" task learning as:

$$J_{\mathcal{T}_i}^{\mathrm{PPO}}(\theta_i) = J_{\mathcal{T}_i}^{C}(\theta_i) - c_1 J_{\mathcal{T}_i}^{\mathrm{VF}}(\theta_i), \quad (20)$$

where c_1 is the coefficient of value function loss.

According to the target of gradient-based MRL defined in Eq. (2) and our objective function given by Eq. (20), the "outer loop" training target of MRLCO is expressed as

$$J^{\mathrm{MRLCO}}(\theta) = \mathbb{E}_{\mathcal{T}_i \sim \rho(\mathcal{T}), \tau \sim P_{\mathcal{T}_i}(\tau, \theta_i')}[J_{\mathcal{T}_i}^{\mathrm{PPO}}(\theta_i')],$$
$$\text{where} \quad \theta_i' = U_{\tau \sim P_{\mathcal{T}_i}(\tau, \theta_i)}(\theta_i, \mathcal{T}_i), \quad \theta_i = \theta. \quad (21)$$

Next, we can conduct gradient ascent to maximize the $J^{\mathrm{MRLCO}}(\theta)$. However, optimizing this objective function involves gradients of gradients, which introduces large computation cost and implementation difficulties when combining a complex neural network such as the seq2seq neural network. To address this challenge, we use the first-order approximation to replace the second-order derivatives as suggested in [17], which is defined as

$$g^{\mathrm{MRLCO}} := \frac{1}{n} \sum_{i=1}^{n} \left[(\theta_i' - \theta)/\alpha/m \right], \quad (22)$$

where n is the number of sampled learning tasks in the "outer loop", α is the learning rate of the "inner loop" training, and m is the conducted gradient steps for the "inner loop" training.

We present the overall design of the algorithm in Algorithm 1. The parameters of the meta policy neural network are denoted as θ. We firstly sample a batch of learning tasks $\mathcal{T}$ with batch size n and conduct "inner loop" training for each sampled learning task. After finishing the "inner loop" training, we update the meta-policy parameters θ by using gradient ascent $\theta \leftarrow \theta + \beta g^{\mathrm{MRLCO}}$ via *Adam* [25]. Here, β is the learning rate of "outer loop" training.

5 PERFORMANCE EVALUATION

This section presents the experimental results of the proposed method. First, we introduce the algorithm hyperparameters of MRLCO and the simulation environment. Next, we evaluate the performance of MRLCO by comparing it with a fine-tuning DRL method and a heuristic algorithm.

TABLE 2
The Neural Network and Training Hyperparameters

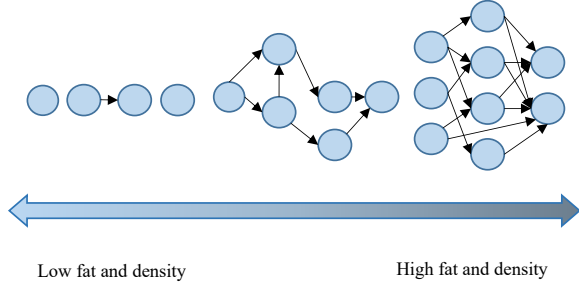| Hyperparameter	Value	Hyperparameter	Value
Encoder Layers	2	Encoder Layer Type	LSTM
Encoder Hidden Units	256	Encoder Layer Norm.	On
Decoder Layers	2	Decoder Layer Type	LSTM
Decoder Hidden Units	256	Decoder Layer Norm.	On
Learning Rate β	5×10^{-4}	Learning Rate α	5×10^{-4}
Optimization Method	Adam	Activation function	Tanh
Discount Factor γ	0.99	Loss Coefficient c_1	0.5
Adv. Discount Factor λ	0.95	Clipping Constant ϵ	0.2
Gradient Step m	3		

Fig. 5. Examples of generated DAGs.

5.1 Algorithm Hyperparameters

The MRLCO is implemented via Tensorflow. The encoder and decoder of the seq2seq neural network are both set as two-layer dynamic Long Short-Term Memory (LSTM) with 256 hidden units at each layer. Moreover, the layer normalization [26] is added in both the encoder and decoder. For the training hyperparameters setting in MRLCO, the learning rate of "inner loop" and "outer loop" are both set as 5×10^{-4}. The coefficient c_1 is set as 0.5 and the clipping constant ϵ is set as 0.2. The discount factors γ and λ are set as 0.99 and 0.95, respectively. The number of gradient steps for "inner loop" training, m, is set as 3. Overall, we summarize the hyperparameter setting in Table 2.

5.2 Simulation Environment

We consider a cellular network, where the data transmission rate varies with the UE's position. The CPU clock speed of UE, f_{UE}, is set to be 1 GHz. There are 4 cores in each VM of the MEC host with the CPU clock speed of 2.5 GHz per core. The offloaded tasks can run in parallel on all cores, thus the CPU clock speed of a VM, f_{vm}, is $4 \times 2.5 = 10$ GHz.

Many real-world applications can be modelled by DAGs, with various topologies and task profiles. To simulate the heterogeneous DAGs, we implement a synthetic DAG generator according to [27]. There are four parameters controlling topologies and task profiles of the generated DAGs: n, *fat*, *density*, and *ccr*, where n represents the task number, *fat* controls the width and height of the DAG, *density* decides the number of edges between two levels of the DAG, and *ccr* denotes the ratio between the communication and computation cost of tasks. Fig. 5 shows the generated DAGs from low fat and density to high fat and density examples.

We design three experiments to evaluate the performance of MRLCO under dynamic scenarios. The first two experiments simulate the scenarios where UE has different application preferences represented by various topologies and task numbers. While the third experiment simulates the scenarios where UE has varying dynamic transmission rates. For all experiments, the data size of each task ranges from 5 KB to 50 KB; the CPU cycles required by each task ranges from 10^7 to 10^8 cycles [5]. The length of child/parent task indices vector p is set as 12. We randomly select *ccr* from 0.3 to 0.5 for each generated DAG, since most of mobile applications are computation-intensive. The generated datasets in each experiment are separated into "training datasets" and "testing datasets". We consider learning an effective offloading policy for each dataset as a learning task. The MRLCO firstly learns a meta policy based on "training datasets" by using Algorithm 1. The learned meta policy is then used as the initial policy to fast learn an effective offloading policy for the "testing datasets".

We compare MRLCO with three baseline algorithms:

- *Fine-tuning DRL*: It first pretrains one policy for all "training datasets" using the DRL-based offloading algorithm proposed in [12]. Next, it uses the parameters of the trained policy network as an initial value of the task-specific policy network, which is then updated on the "testing datasets".
- *HEFT-based*: This algorithm is adapted from [4], which firstly prioritizes tasks based on HEFT and then schedules each task with earliest estimated finish time.
- *Greedy*: Each task is greedily assigned to the UE or the MEC host based on its estimated finish time.

5.3 Results Analysis

In the first experiment, we generate DAG sets with different topologies to simulate the scenario where users have different preferences of mobile applications. Each dataset contains 100 DAGs of similar topologies with the same *fat* and *density*, which are two key parameters influencing the DAG topology. We set the task number for each generated DAG as $n = 20$ and set *fat* $\in \{0.4, 0.5, 0.6, 0.7, 0.8\}$, *density* $\in \{0.4, 0.5, 0.6, 0.7, 0.8\}$. 25 DAG sets are generated with different combinations of fat and density. Each DAG set represents the application preference of one mobile user and consider finding the effective offloading policy for a DAG set as a learning task. We randomly select 22 DAG sets as the training datasets and the other 3 as unseen testing datasets. We train the MRLCO and the fine-tuning DRL method on the training datasets and evaluate MRLCO and baseline algorithms on the testing datasets.

During training of MRLCO, we set the meta batch size as 10, thus 10 learning tasks are sampled from $\rho(\mathcal{T})$ in the "outer loop" training stage. At each "inner loop", we sample 20 trajectories for a DAG and conduct m policy gradient updates ($m = 3$) for the PPO target. After training, we evaluate the MRLCO and fine-tuning DRL method by running up to 20 policy gradient updates, each samples 20 trajectories for a DAG on the testing datasets. Fig. 6 shows the performance of the MRLCO and baseline algorithms with different DAG sets. Overall, the Greedy algorithm has the highest latency, while the MRLCO obtains the lowest latency. Fig. 6a demonstrates that the MRLCO is better than
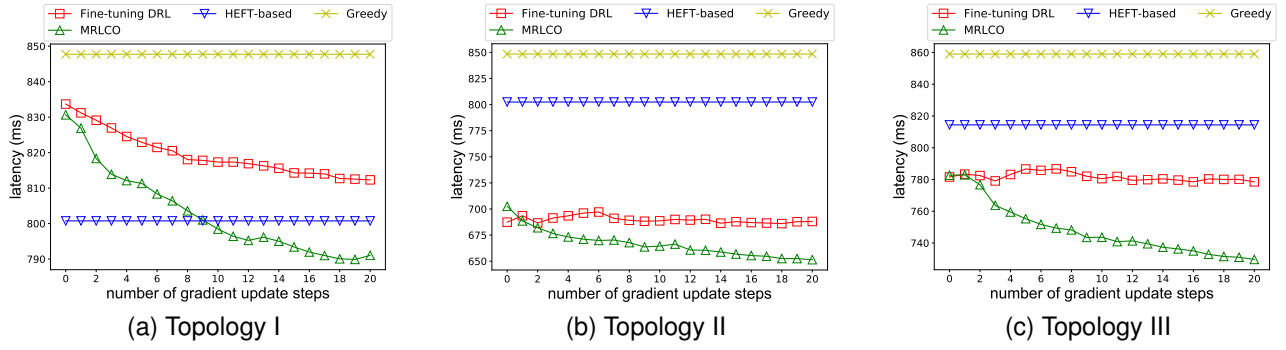
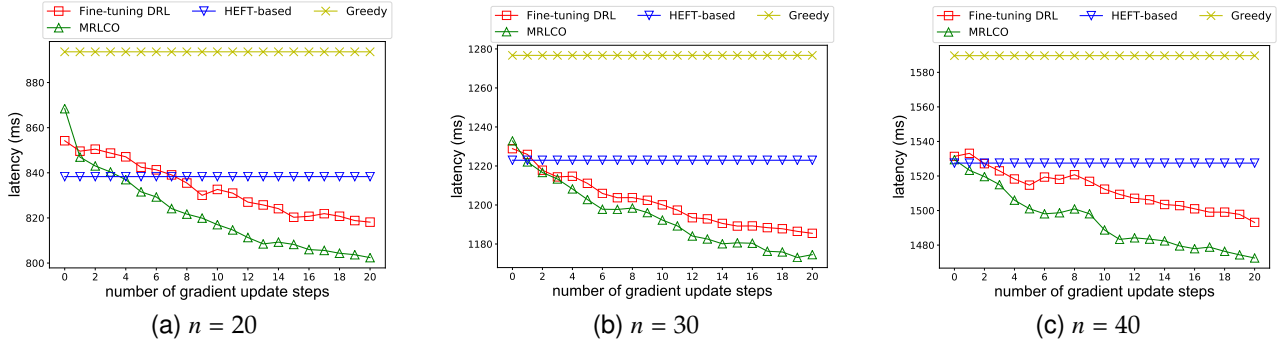Fig. 6. Evaluation results with different DAG topologies.

(a) Topology I (b) Topology II (c) Topology III

Fig. 7. Evaluation results with different task numbers.

(a) $n = 20$ (b) $n = 30$ (c) $n = 40$

Fig. 8. Evaluation results with different transmission rates.
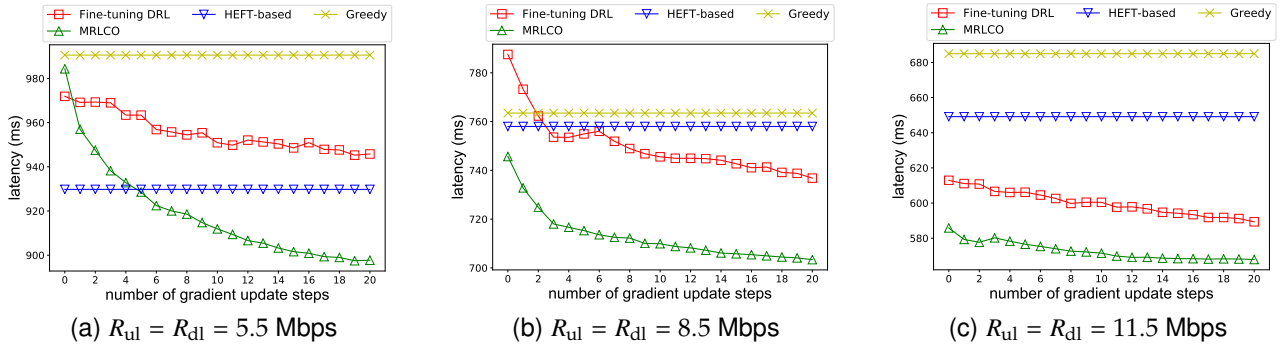
(a) $R_{ul} = R_{dl} = 5.5$ Mbps (b) $R_{ul} = R_{dl} = 8.5$ Mbps (c) $R_{ul} = R_{dl} = 11.5$ Mbps

the HEFT-based algorithm after 9 steps of gradient update, while the fine-tuning DRL method consistently performs worse than the HEFT-based algorithm. This indicates that the MRLCO can adapt to new tasks much more quickly than the fine-tuning DRL method. In Fig. 6b and Fig. 6c, the MRLCO and the fine-tuning DRL method with 0 step of gradient updates already beat the two heuristic-based algorithms: HEFT-based and the Greedy algorithms, because both the MRLCO and fine-turning DRL learn the updated policy based on the pre-trained models instead of learning from scratch. These heuristic-based algorithms use fixed policies to obtain the offloading plan, which cannot adapt well to different DAG topologies.

The second experiment aims to show the influence of the task number n on the performance of different algorithms. We randomly generate 6 training datasets with $n \in \{10, 15, 25, 35, 45, 50\}$ and 3 testing datasets with $n \in \{20, 30, 40\}$. In each dataset, we generate DAGs by randomly selecting *fat* from $\{0.4, 0.5, 0.6, 0.7, 0.8\}$, *density* from $\{0.4, 0.5, 0.6, 0.7, 0.8\}$, and *ccr* from 0.3 to 0.5, thus the distributions of DAG topologies of all datasets are similar. In this experiment, we set the meta batch size as 5 and the rest of the settings the same as the first experiment. Fig. 7 shows that both the MRLCO and the fine-tuning DRL method outperform the HEFT-based algorithms after a few gradient updates, and are consistently better than the Greedy from step 0 of gradient updates. Moreover, MRLCO adapts to new learning tasks faster than the fine-tuning DRL method. For example, Fig. 7b shows that, after one step gradient update, the latency of MRLCO decreases sharply and is less than both fine-tuning and HEFT-based algorithms. After 20 gradient updates, MRLCO obtains the lowest latency

TABLE 3
The comparison of MRLCO and baseline algorithms in average latency (ms) on different testing datasets,
N/A denotes cases unable to find optimum.

Testing dataset		Heuristic Algorithms		Fine-tuning DRL		MRLCO	
	Optimal	HEFT-based	Greedy	update steps(20)	update steps(100)	update steps(20)	update steps(100)
Toploogy I	679.31	800.75	847.73	812.32	789.92	791.03	722.63
Toploogy II	555.46	802.46	848.43	688.05	636.49	651.42	601.93
Toploogy III	605.05	814.39	859.03	778.52	712.79	729.63	641.92
$n = 20$	689.21	838.31	893.62	818.14	802.50	802.41	743.42
$n = 30$	N/A	1222.93	1276.70	1185.47	1152.07	1174.55	1098.43
$n = 40$	N/A	1527.47	1589.66	1493.11	1432.41	1472.53	1397.63
$R_{ul} = R_{dl} = 5.5$ Mbps	770.10	929.79	990.58	945.82	901.36	897.73	831.58
$R_{ul} = R_{dl} = 8.5$ Mbps	628.21	757.99	763.49	736.81	701.75	703.38	674.93
$R_{ul} = R_{dl} = 11.5$ Mbps	524.14	649.15	684.97	589.33	570.19	567.88	548.26

compared to the baseline algorithms.

We conduct the third experiment to evaluate the performance of MRLCO with different transmission rates. Learning the offloading policy for each transmission rate is considered as an individual learning task. We randomly generate the DAG dataset by setting $n = 20$ and other parameters the same as the second experiment. In addition, we implement *Optimal* algorithm via exhaustively searching the solution space to find the optimal offloading plan. We conduct meta training process based on randomly selected transmission rates from 4 Mbps to 22 Mbps with a step size of 3 Mbps. We then evaluate the trained meta policy among transmission rates from {5.5 Mbps, 8.5 Mbps, 11.5 Mbps}, which are unseen in the training procedure. Fig. 8 shows that the MRLCO again adapts to new learning tasks much faster than the fine-tuning DRL method in all test sets and achieves the lowest latency after 20 gradient updates. In some cases (Fig. 8b and Fig. 8c), MRLCO even achieves the lowest latency at the initial point.

Table 3 summarizes the average latency of all algorithms on different testing datasets. Overall, the MRLCO outperforms all heuristic baseline algorithms after 20 gradient update steps. The MRLTO and fine-tuning DRL method will get better results with more update steps. Table 3 also shows the performance of the fine-tuning and the MRLCO algorithms after 100 update steps. Compared to the fine-tuning algorithm, the MRLCO achieves better result after both 20 and 100 update steps. However, there are still gaps between the results of MRLCO and the Optimal values. One possible solution could be to integrate the seq2seq neural network with another sample efficient off-policy MRL method [28], which is a direction for future work.

6 RELATED WORK

The task offloading problem in MEC has attracted significant research interests [5], [10], [11], [13], [29]–[40]. In general, there are two task models used in the related work: task model for binary offloading and that for partial offloading [29]. In the task model for binary offloading, there are no inner dependencies among computation tasks for an application. Dinh et al. [5] aimed to find an offloading plan for a set of tasks among different access points and MEC hosts, in order to achieve the minimal joint target of latency and energy. Chen et al. [30] focused on computation

offloading for independent tasks in a software-defined ultra-dense network. They formulated the task offloading problem as a mixed integer non-linear program and solved it by using decomposition and heuristic methods. Hong et al. [31] proposed an approximate dynamic programming algorithm for computation offloading to achieve the optimal quality of experience. In the task model for partial offloading, applications were composed of tasks with inner dependencies, which is able to achieve a fine granularity of computation offloading, leading to better offloading performance. Wang et al. [32] modelled both the applications and the computing system as graphs and proposed an approximation algorithm for finding the task offloading plan to obtain the lowest cost. Neto et al. [33] implemented a user-level online offloading framework for Android applications, aiming at minimizing the remote execution overhead. Zanni et al. [34] proposed an innovative task selection algorithm for Android applications, achieving method-level granularity of offloading.

In order to adapt the offloading strategies for dynamic scenarios, over recent years, DRL has been widely applied to solve task offloading problems in MEC systems. Dinh et al. [10] focused on the multi-user multi-edge-node computation offloading problem by using deep Q-learning. Chen et al. [11] considered an ultra-dense network, where multiple base stations can be selected for offloading. They also adopted deep Q-learning to obtain the offloading strategy. Huang et al. [38] proposed a DRL-based offloading framework which jointly considers both the offloading decisions and resource allocations. Zhan et al. [36] proposed an efficient task offloading method combining PPO and convolutional neural networks. Tan et al. [37] proposed a deep Q-learning based offloading method considering constraints of limited resources, vehicle's mobility and delay. Huang et al. [13] proposed a DRL-based online offloading framework to maximize the weighted sum of the computation rates of all the UE. Ning et al. [39] proposed a deep Q-learning based method for jointly optimising task offloading and resource allocation in MEC. The existing studies mostly assume the offloading problem as one learning task and apply conventional DRL algorithms to solve the task. However, many DRL algorithms suffer from poor sample efficiency — when facing new scenarios, those DRL-based offloading methods need long training time to find the effective policy, which impedes their practical deployment. To address this issue,

our offloading method adopts an MRL approach which can efficiently solve new learning tasks with the requirement of only few gradient update steps and a small amount of data. As a result, our method can quickly adapt to the changes of environments with the requirement of only few training steps rather than fully retraining the offloading policy from scratch. With the low demands of computation and data, our method can efficiently run on the resource-constrained UE using its own data.

7 DISCUSSION

MRLCO has many advantages over the existing RL-based task offloading methods, such as learning to fast adapt in a dynamic environment and high sample efficiency. Beyond the scope of task offloading in MEC systems, the proposed MRLCO framework has the potential to be applied to solve more decision-making problems in MEC systems. For instance, content caching in MEC aims to cache popular contents at MEC hosts to achieve high Quality-of-Service (QoS) for mobile users and reduce the network traffic. While MEC hosts can have different caching policies to suit the dynamic content preferences and network conditions of users in different areas. The proposed MRL framework can be adapted to solve this problem through executing the "outer loop" training at cloud servers to learn a meta caching policy and the "inner loop" training at MEC hosts to learn a specific caching policy for each MEC host.

Even though MRLCO has many benefits to MEC systems, there are several challenges for further exploration. In this paper, we consider stable wireless channels, reliable mobile devices, and sufficient computation resources. Thus, the MRLCO will not break down when increasing the number of users. However, when operating at large-scale, some UE as stragglers may drop out due to broken network connections or insufficient power. Considering the synchronous process of "outer loop" training that updates the meta policy after gathering parameters from all UE, the stragglers might affect the training performance of MRLCO. One way to solve this issue is to apply an adaptive client selection algorithm which can automatically filter out stragglers and select reliable clients to join the training process based on their running states.

8 CONCLUSION

This paper proposes an MRL-based approach, namely MRLCO, to solve the computation offloading problem in MEC. Distinguished from the existing works, the MRLCO can quickly adapt to new MEC environments within a small number of gradient updates and samples. In our proposed method, the target mobile applications are modelled as DAGs, the computation offloading process is converted to a sequence prediction process, and a seq2seq neural network is proposed to effectively represent the policy. Moreover, we adopt the first-order approximation for the MRL objective to reduce the training cost and add a surrogate clipping to the objective so as to stabilize the training. We conduct simulation experiments with different DAG topologies, task numbers, and transmission rates. The results demonstrate that, within a small number of training steps, MRLCO

achieves the lowest latency compared to three baseline algorithms including a fine-tuning DRL method, a greedy algorithm, and an HEFT-based algorithm.

REFERENCES

[1] D. Sabell, V. Sukhomlinov, L. Trang, S. Kekki, P. Paglierani, R. Rossbach, X. Li, Y. Fang, D. Druta, F. Giust *et al.*, "Developing software for multi-access edge computing," *ETSI White Paper*, vol. 20, 2019.

[2] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: enabling interactive perception applications on mobile devices," in *Proceedings of MobiSys*. ACM, 2011, pp. 43–56.

[3] A. Al-Shuwaili and O. Simeone, "Energy-efficient resource allocation for mobile edge computing-based augmented reality applications," *IEEE Wireless Communications Letters*, vol. 6, no. 3, pp. 398–401, 2017.

[4] X. Lin, Y. Wang, Q. Xie, and M. Pedram, "Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment," *IEEE Transactions on Services Computing*, vol. 8, no. 2, pp. 175–186, 2015.

[5] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Transactions on Communications*, vol. 65, no. 8, pp. 3571–3584, 2017.

[6] H. Wu, W. Knottenbelt, and K. Wolter, "An efficient application partitioning algorithm in mobile environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 7, pp. 1464–1480, 2019.

[7] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[8] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[9] S. Chinchali, P. Hu, T. Chu, M. Sharma, M. Bansal, R. Misra, M. Pavone, and S. Katti, "Cellular network traffic scheduling with deep reinforcement learning," in *Proceedings of AAAI*, 2018.

[10] T. Q. Dinh, Q. D. La, T. Q. Quek, and H. Shin, "Learning for computation offloading in mobile edge computing," *IEEE Transactions on Communications*, vol. 66, no. 12, pp. 6353–6367, 2018.

[11] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and 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, 2019.

[12] J. Wang, J. Hu, G. Min, W. Zhan, Q. Ni, and N. Georgalas, "Computation offloading in multi-access edge computing using a deep sequential model based on reinforcement learning," *IEEE Communications Magazine*, vol. 57, no. 5, pp. 64–69, 2019.

[13] L. Huang, S. Bi, and Y. J. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Transactions on Mobile Computing*, 2019.

[14] J. Vanschoren, "Meta-learning: A survey," *arXiv preprint arXiv:1810.03548*, 2018.

[15] M. Botvinick, S. Ritter, J. X. Wang, Z. Kurth-Nelson, C. Blundell, and D. Hassabis, "Reinforcement learning, fast and slow," *Trends in cognitive sciences*, 2019.

[16] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of ICML*, 2017, pp. 1126–1135.

[17] A. Nichol, J. Achiam, and J. Schulman, "On first-order meta-learning algorithms," *arXiv preprint arXiv:1803.02999*, 2018.

[18] X. Sun and N. Ansari, "Avaptive avatar handoff in the cloudlet network," *IEEE Transactions on Cloud Computing*, 2017.

[19] C. Borcea, X. Ding, N. Gehani, R. Curtmola, M. A. Khan, and H. Debnath, "Avatar: Mobile distributed computing in the cloud," in *2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*. IEEE, 2015, pp. 151–156.

[20] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proceedings of ICLR*, 2015.

[21] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Proceedings of NIPS*, 2015, pp. 2692–2700.

[22] S. E. Mahmoodi, R. Uma, and K. Subbalakshmi, "Optimal joint scheduling and cloud offloading for mobile applications," *IEEE Transactions on Cloud Computing*, vol. 7, no. 2, pp. 301–313, 2019.

[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[24] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *Proceedings of ICLR*, 2016.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of ICLR*, 2015.

[26] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," in *Proceedings in NIPS*, 2016.

[27] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 682–694, 2013.

[28] K. Rakelly, A. Zhou, D. Quillen, C. Finn, and S. Levine, "Efficient off-policy meta-reinforcement learning via probabilistic context variables," *arXiv preprint arXiv:1903.08254*, 2019.

[29] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[30] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.

[31] S.-T. Hong and H. Kim, "Qoe-aware computation offloading to capture energy-latency-pricing tradeoff in mobile clouds," *IEEE Transactions on Mobile Computing*, vol. 18, no. 9, pp. 2174–2189, 2019.

[32] S. Wang, M. Zafer, and K. K. Leung, "Online placement of multi-component applications in edge computing environments," *IEEE Access*, vol. 5, pp. 2514–2533, 2017.

[33] J. L. D. Neto, S.-Y. Yu, D. F. Macedo, J. M. S. Nogueira, R. Langar, and S. Secci, "Uloof: a user level online offloading framework for mobile edge computing," *IEEE Transactions on Mobile Computing*, vol. 17, no. 11, pp. 2660–2674, 2018.

[34] A. Zanni, S.-Y. Yu, P. Bellavista, R. Langar, and S. Secci, "Automated selection of offloadable tasks for mobile computation offloading in edge computing," in *2017 13th international conference on network and service management (CNSM)*. IEEE, 2017, pp. 1–5.

[35] A. Zanni, S.-Y. Yu, S. Secci, R. Langar, P. Bellavista, and D. F. Macedo, "Automated offloading of android applications for computation/energy-usage optimizations," in *INFOCOM WK-SHPS*. IEEE, 2017.

[36] W. Zhan, C. Luo, J. Wang, C. Wang, G. Min, H. Duan, and Q. Zhu, "Deep reinforcement learning-based offloading scheduling for vehicular edge computing," *IEEE Internet of Things Journal*, 2020.

[37] L. T. Tan and R. Q. Hu, "Mobility-aware edge caching and computing in vehicle networks: A deep reinforcement learning," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 11, pp. 10 190–10 203, 2018.

[38] L. Huang, X. Feng, C. Zhang, L. Qian, and Y. Wu, "Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing," *Digital Communications and Networks*, vol. 5, no. 1, pp. 10–17, 2019.

[39] Z. Ning, P. Dong, X. Wang, J. J. Rodrigues, and F. Xia, "Deep reinforcement learning for vehicular edge computing: An intelligent offloading system," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 6, pp. 1–24, 2019.

[40] W. Zhan, C. Luo, G. Min, C. Wang, Q. Zhu, and H. Duan, "Mobility-aware multi-user offloading optimization for mobile edge computing," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 3, pp. 3341–3356, 2020.

Jia Hu is a Senior Lecturer in Computer Science at the University of Exeter. He received his Ph.D. degree in Computer Science from the University of Bradford, UK, in 2010, and M.Eng. and B.Eng. degrees in Electronic Engineering from Huazhong University of Science and Technology, China, in 2006 and 2004, respectively. His research interests include mobile and ubiquitous computing, resource optimization, applied machine learning, and network security.

Geyong Min is a Professor of High Performance Computing and Networking in the Department of Computer Science within the College of Engineering, Mathematics and Physical Sciences at the University of Exeter, United Kingdom. He received the PhD degree in Computing Science from the University of Glasgow, United Kingdom, in 2003, and the B.Sc. degree in Computer Science from Huazhong University of Science and Technology, China, in 1995. His research interests include Computer Networks, Wireless Communications, Parallel and Distributed Computing, Ubiquitous Computing, Multimedia Systems, Modelling and Performance Engineering.

Albert Y. Zomaya is currently the Chair Professor of High Performance Computing & Networking in the School of Computer Science, University of Sydney. He is also the Director of the Centre for Distributed and High Performance Computing. He published more than 600 scientific papers and articles and is author, co-author or editor of more than 20 books. He is the Founding Editor in Chief of the IEEE Transactions on Sustainable Computing and previously he served as Editor in Chief for the IEEE Transactions on Computers (2011-2014). He delivered more than 190 keynote addresses, invited seminars, and media briefings and has been actively involved, in a variety of capacities, in the organization of more than 700 conferences. He is the recipient of the IEEE Technical Committee on Parallel Processing Outstanding Service Award (2011), the IEEE Technical Committee on Scalable Computing Medal for Excellence in Scalable Computing (2011), the IEEE Computer Society Technical Achievement Award (2014), the ACM MSWIM Reginald A. Fessenden Award (2017), and the New South Wales Premiers Prize of Excellence in Engineering and Information and Communications Technology (2019). He is a Chartered Engineer, a Fellow of AAAS, IEEE, IET (UK), an Elected Member of Academia Europaea, and an IEEE Computer Societies Golden Core member.

Nektarios Georgalas is a Principal Researcher in the Applied Research department of British Telecom. In his current role, he leads two collaborative research programmes with key BT partners delivering innovations in the areas of Cloud, Data Centres, Network Virtualisation, Smart Cities, IoT and Mobility. During his career with BT, since 1998, he has managed numerous collaborative and internal research projects in areas such as network management, market-driven data management systems, policy-based management, distributed information systems, SOA/Web Services, Model Driven Design and Development of telecoms OSS, Cloud and NFV. Nektarios is inventor and co-inventor of 11 patents. He has also authored more than 60 papers in international journals and conferences. He has served as General Co-chair, Programme Co-Chair, Programme Committee and Keynote Speaker or Invited Panelist in top international IEEE academic and TMForum conferences.

Jin Wang is a Ph.D. candidate in Computer Science at the University of Exeter. He received both B.Eng. and M.Eng. degree in Computer Science from the University of Electronic Science and Technology of China (UESTC), Chengdu, China in 2014 and 2017, respectively. His research interests include deep reinforcement learning, applied machine learning, cloud and edge computing, and computer system optimization.