Deep Reinforcement Learning-based Task Offloading over In-Network Computing and Multi-Access Edge Computing

Zhao Ming*, Qize Guo*, Hao Yu*, and Tarik Taleb*
*Oulu University, Oulu, Finland
Email: {zhao.ming, qize.guo, hao.yu, tarik.taleb}@oulu.fi

***Abstract*—With the blooming of information technology and network applications/services, emerging multi-access edge computing (MEC) and in-network computing (INC) are regarded as key computing paradigms to support time-sensitive tasks and requests by task offloading. Existing studies concerning task offloading seldom considered the combinations of MEC, INC, and cloud computing. In this paper, we explore INC-enhanced task offloading in MEC networks and design a three-layer task offloading network architecture, which consists of not only user equipment, edge servers, and cloud, but also network elements like routers/switches. We focus on reducing the system latency and energy consumption (EC) and formulate the optimization problem as minimizing the weighted sum of these two indicators. To solve this problem, we propose a deep reinforcement learning-based framework and creatively map the actions of the agent to the offloading policies and resource allocation strategies for determining these two indicators simultaneously. Simulation results show that the proposed INC-enhanced task offloading framework achieves fast convergence speed with double deep Q-network and outperforms other baselines in reducing the system latency and EC.**

***Index Terms*—Task Offloading, In-networking Computing, Deep Reinforcement Learning**

I. Introduction

With the rapid development of information technology and the emergence of various network services and applications such as Internet of Things (IoT), Internet of Vehicles, and Augmented Reality, a huge number of devices and facilities are connected to the network to serve vertical industries, posing great challenges to the running and maintenance of the network. On the one hand, the time-sensitive requests may need to be processed even within 0.1 ms [1]–[3], which is hard to be met by the remote cloud; on the other hand, processing tasks at user equipments (UEs) will introduce huge energy consumption (EC) and high latency due to the limited calculation resources. Under this circumstance, an emerging computing paradigm, Multi-access Edge Computing (MEC), was proposed to address this challenge [4]–[8]. As a key technology of MEC, task offloading focuses on offloading the tasks from UEs to the edge of the network or the remote cloud, and can significantly reduce the latency of processing tasks and the EC of UEs [9]–[11].

In-network computing (INC), which tries to utilize the network elements like programmable switches/routers for computation, is witnessed to achieve much lower latency and has received a great deal of interest recently [12], [13]. Specifically, to meet the strict service requirements of tasks, these programmable network elements can not only be utilized for packet manipulation and forwarding but also for processing requests from UEs that are close to, by exploiting the vacant resources for calculation [14]. Thus, INC can further improve the calculation capacity of the network, reduce the latency for processing tasks, and reduce the network traffic.

Lots of researchers have investigated INC for processing tasks and requests, for instance, in [13], the researchers proposed an INC-empowered task offloading framework to offload lightweight critical tasks to the INC devices. In [14], Leonardo *et al.* designed a multi-source deep learning-based task offloading model and proposed a particle swarm optimization algorithm to solve the problem. The authors in [15] proposed to solve the formulated in-network task offloading problem by the integer linear programming algorithm. Besides, in [16], the authors proposed an energy-efficient in-network computing paradigm that integrated network functions into a computing platform that supports the co-optimization of resource utilization and network communication overhead and mimicked real application data requests for evaluation. However, these studies either didn't consider the combination of INC, MEC, and cloud computing for task offloading, or utilized heuristic methods to solve the optimization problem, rather than adopting more intelligent AI-based solutions. Thus, the design of a systematical and intelligent INC-enhanced task offloading framework, with MEC and cloud computing considered, is still unexplored well.

In this paper, we investigate INC-enhanced task offloading in MEC networks with cloud computing considered. Specifically, we design a three-layer task offloading network architecture that consists of UEs, edge servers (ESs), INC devices, and the remote cloud, and formulate the problem as minimizing the weighted sum of system latency and EC. To solve this problem, we propose a double Deep Q-learning (DQN) based framework and creatively map the actions of the agent to the task offloading policies and resource allocation strategies for determining these two indicators simultaneously. To evaluate our proposed framework, we perform numerical experiments and simulation results have demonstrated that our proposed scheme outperforms existing schemes in reducing the system latency and EC. The main contributions of this paper are as follows:

- We propose a comprehensive INC-enhanced task offloading framework in MEC networks, which combines INC, MEC, and cloud computing for task offloading, and design a general three-layer network architecture under this scenario;
- We focus on reducing the system latency and EC and formulate the optimization problem as minimizing the weighted sum of the two indicators. To solve this problem, we propose a DRL-based task offloading algorithm and creatively map the actions of the agent to offloading policies and resource allocation strategies, to determine these two indicators simultaneously;
- Simulation results have demonstrated that the proposed scheme has fast convergence speed and outperforms existing schemes in reducing the system latency and EC.

The rest of this paper is organized as follows. Sect. II introduces the system model and formulates the optimization problem. The DRL-based task offloading framework for solving the problem is provided in Sect. III, Sect. IV presents the simulation results. Finally, Sect. V concludes this paper.

II. System Model and Problem Formulation

In this part, we introduce the network architecture and system model and formulate the optimization problem.

A. Network Architecture

The considered network architecture of INC-enhanced task offloading in MEC networks is shown in Fig. 1, which consists of three layers, i.e., device layer, edge layer, and cloud layer. In the device layer, we simultaneously consider indoor and outdoor scenarios. The indoor UEs include smartphones, tablets, personal computers, IoT devices, *etc.*, and are connected to routers via wireless links. Besides, in the outdoor scenario, different kinds of UEs are geographically distributed and connected to Base Stations (BSs). We assume a UE can only be served by one router or one BS in this paper. In the edge layer, the routers and BSs are connected to the core network and the remote cloud through network switches via backhaul links, which are generally built with optical links. Moreover, the switches are also connected to their neighbors by optical links. Note that these network elements with limited calculation/storage resources not only can support packet manipulation but also can be programmed for processing tasks. Moreover, we consider the BSs to be equipped with ESs with limited calculation and storage resources, and the cloud has large capacities for calculation and storage.

For the task offloading scenario, we assume that tasks are generated at UEs and can be processed locally (at the UEs), or offloaded to the routers/ESs, switches, or the cloud. We consider the INC-enhanced task offloading scenario periodically. Specifically, in each period, the UEs continuously generate tasks, and each router/ES collects task information (e.g., task size, require CPU cycles, *etc.*) and device information (e.g., remaining resources of UEs, routers, ESs, *etc.*) and send to the cloud to determine the offloading policies and resource

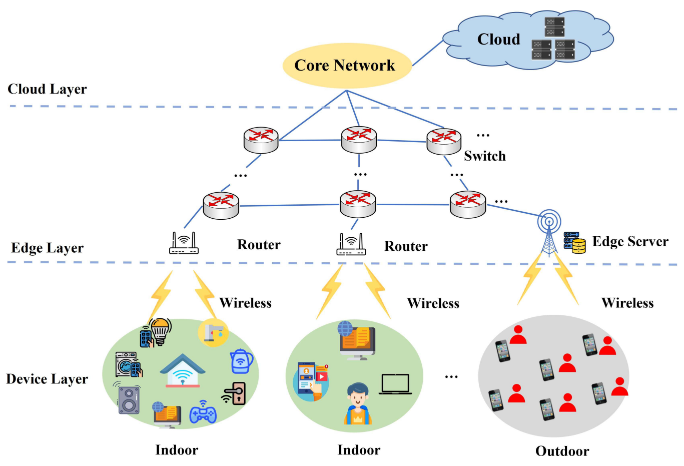


Fig. 1. The considered network architecture of INC-enhanced task offloading in MEC networks.

allocation strategies of tasks. After that, these tasks are offloaded to the corresponding devices for processing, and the network status information like system latency will be utilized to measure the performance of the decision policies. Moreover, we neglect the time for collecting task and device information and for decision-making in this paper.

B. System Model

We consider a general INC-enhanced task offloading scenario with N UEs denoted as $\mathcal{N}$ and M switches denoted as $\mathcal{M} = \{1, 2, \ldots, M\}$. The total storage and calculation resources of UE $n \in \mathcal{N}$ are denoted as D_n and C_n, respectively, where D_n indicates the number of bits n can cache, and C_n indicates the CPU cores of n. The total storage and calculation resources (in CPU cores) of $m \in \mathcal{M}$ can be denoted as D_m and C_m. UE n generates a task T_n in each period, the size and calculation amount (in CPU cycles) of T_n can be denoted as D_{T_n}, and $C_{T_n}, \forall n \in \mathcal{N}$, respectively. Moreover, we denote L_n as the router or ES that n links to, where L_n should be a router when UE n is an indoor device or an ES when n is an outdoor device. Similarly, the total storage and calculation resources (in CPU cores) of L_n can be denoted as D_{L_n} and C_{L_n}. L_n can only connect to one of the switches, we denote $S_n \in \mathcal{M}$ as the switch L_n connects to. The total storage and calculation resources of the cloud are denoted as D and C.

C. INC Enhanced Task Offloading Model

For task T_n, it can be processed at UE n (defined as *locally*), offloaded to the linked router/ES (L_n), offloaded to one of the switches $m \in \mathcal{M}$, or offloaded to the cloud. Moreover, allocating how many CPU cores to process task T_n should also be considered when offloading T_n. We define $\pi_n \in \{-1, 0, 1, \ldots, M, M+1\}$ as the offloading policy and γ_n as the resource allocation strategy of T_n, respectively. Here, -1, 0, and $M+1$ indicate T_n is processed locally, offloaded to L_n, or offloaded to the cloud, respectively; $1, 2, \ldots, M$ indicate T_n is offloaded to switch 1, 2, …, or M, respectively.

Besides, γ_n represents the number of CPU cores that should be allocated to process T_n.

T_n should firstly be transmitted from UE n to where it is offloaded, we denote the transmission latency as t_n^{trans} and the EC as e_n^{trans}. When T_n is processed locally ($\pi_n = -1$), intuitively we have $t_n^{trans} = e_n^{trans} = 0$. Otherwise, if T_n is offloaded to its linked router/ES ($\pi_n = 0$), it should be transmitted from n to L_n by wireless links. We denote the wireless channel gain from n to L_n as h_n, the transmit power of n as P_n, the bandwidth of the wireless communication as W, thus, we can calculate the upload speed from n to L_n (denoted as R_n) as

$$R_n = Wlog_2(1 + \frac{P_n|h_n|^2}{\sigma^2}). \tag{1}$$

Thus, t_n^{trans} can be calculated as $t_n^{trans} = \frac{D_{T_n}}{R_n}$, and e_n^{trans} can be calculated as $e_n^{trans} = P_n t_n^{trans}$. Otherwise if $\pi_n = 1$, 2, ..., or M, it should first be uploaded to L_n by wireless links, then sent to S_n, and finally sent to the destination via optical links. In this case, we denote the shortest path from S_n to π_n as a set Ω_n, which consists of $|\Omega_n|$ switches. Thus, the transmission latency t_n^{trans} should be calculated as

$$t_n^{trans} = \frac{D_{T_n}}{R_n} + |\Omega_n|\frac{D_{T_n}}{R}, \tag{2}$$

where R denotes the transmission speed of optical links. We denote the transmission power between switches as P^S, thus, e_n^{trans} can be calculated as

$$e_n^{trans} = P_n\frac{D_{T_n}}{R_n} + P^S|\Omega_n|\frac{D_{T_n}}{R}. \tag{3}$$

Otherwise if $\pi_n = M + 1$, T_n should be offloaded to the remote cloud for processing. Without loss of generality, we consider the latency and EC of the system for transmitting T_n from L_n to the remote cloud as fixed numbers in this paper, denoted as t^C and e^C, thus, we have the t_n^{trans} and e_n^{trans} as the sum of the latency/EC of uploading T_n to L_n and transmitting T_n from L_n to the remote cloud, written by

$$\begin{cases} t_n^{trans} = \frac{D_{T_n}}{R_n} + t^C, \\ e_n^{trans} = P_n\frac{D_{T_n}}{R_n} + e^C. \end{cases}, \forall n \in \mathcal{N}. \tag{4}$$

Based on the analysis above, we have

$$t_n^{trans} = \begin{cases} 0, \ \pi_n = -1 \\ \frac{D_{T_n}}{R_n}, \pi_n = 0 \\ \frac{D_{T_n}}{R_n} + |\Omega_n|\frac{D_{T_n}}{R}, \ \pi_n \in \mathcal{M} \\ \frac{D_{T_n}}{R_n} + t^C, \pi_n = M+1 \end{cases}, \forall n \in \mathcal{N}. \tag{5}$$

The EC for transmission T_n to where it should be processed can be calculated by

$$e_n^{trans} = \begin{cases} 0, \ \pi_n = -1 \\ P_n\frac{D_{T_n}}{R_n}, \pi_n = 0 \\ P_n\frac{D_{T_n}}{R_n} + P^S|\Omega_n|\frac{D_{T_n}}{R}, \pi_n \in \mathcal{M} \\ P_n\frac{D_{T_n}}{R_n} + e^C, \pi_n = M+1 \end{cases}, \forall n \in \mathcal{N}. \tag{6}$$

We denote the calculation latency of T_n as t_n^{cal}, which can be calculated by $t_n^{cal} = \frac{C_{T_n}}{\gamma_n F}$, where F denotes the capacity of a CPU core, i.e., the CPU cycles that a CPU core can calculate per second. Moreover, when a task is processed in routers/ESs, switches, or the cloud which are connected to the power supply, we denote the power of one CPU core as P^C, the calculation EC of T_n when processing in UEs can be obtained by

$$e_n^{cal} = \begin{cases} \kappa(\gamma_n F)^2 * C_{T_n}, \pi_n = -1; \\ P^C\gamma_n t_n^{cal}, \ \ otherwise; \end{cases}, \forall n \in \mathcal{N}, \tag{7}$$

where $\kappa = 10^{-26}$ indicates the EC coefficient of chips. Thus, the latency and EC for offloading task T_n can be given by

$$\begin{cases} t_n = t_n^{trans} + t_n^{cal}, \\ e_n = e_n^{trans} + e_n^{cal}, \end{cases}, \forall n \in \mathcal{N}. \tag{8}$$

D. Problem Formulation

We focus on minimizing the weighted sum of system latency and EC. Denote an indicator variable $\Theta_{n,X} \in \{0, 1\}$ to represent if π_n equals to X or not, where "1" means $\pi_n = X$ and "0" means not. Thus, we can formulate the optimization problem with constraints as

$$\min_{\{\pi_n,\gamma_n\}} \sum_{n\in\mathcal{N}} \alpha t_n + \beta e_n \tag{9a}$$

$$s.t. \ \Theta_{n,-1} \ D_{T_n} \le D_n, \Theta_{n,-1} \ \gamma_n \le C_n, \tag{9b}$$

$$\Theta_{n,0} \ D_{T_n} \le D_{L_n}, \Theta_{n,0} \ \gamma_n \le C_{L_n}, \tag{9c}$$

$$\Theta_{n,S_n} \ D_{T_n} \le D_{S_n}, \Theta_{n,S_n} \ \gamma_n \le C_{S_n}, \tag{9d}$$

$$\sum_{n\in\mathcal{N}} \Theta_{n,m} \ D_{T_n} \le D_m,$$

$$\sum_{n\in\mathcal{N}} \Theta_{n,m} \ \gamma_n \le C_m, \forall m \in \mathcal{M}\backslash S_n, \tag{9e}$$

$$\sum_{n\in\mathcal{N}} \Theta_{n,M+1} \ D_{T_n} \le D, \sum_{n\in\mathcal{N}} \Theta_{n,M+1} \ \gamma_n \le C, \tag{9f}$$

$$\forall n \in \mathcal{N}. \tag{9g}$$

Here, constraints (9b), (9c), and (9d) ensure that the requested resources of task T_n can be satisfied when T_n is offloaded to n, L_n, and S_n, respectively; constraints (9e) and (9f) ensure the requested resources of tasks that offloaded to the switches and the cloud swarm can be satisfied, respectively.

III. DRL-based INC-enhanced task offloading framework

In this section, we introduce a DRL-based framework for INC-enhanced task offloading to determine the offloading strategies in each period. The system state, system action, and system reward are given as follows.

A. DRL Framework Design

1) System State: The size and requested computation resources of task T_n, as well as the available resources of all the system facilities (UEs, routers, switches, ESs, and the cloud), will simultaneously affect the decision-making strategies of where to process the task and how

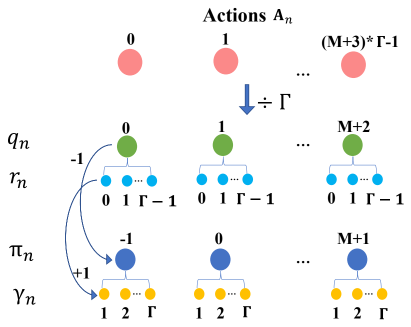


Fig. 2. The action mapping framework.

many computation resources should be allocated. We define $\hat{D}_\iota$ and $\hat{C}_\iota$ as the available storage and calculation resources for a facility ι, and $\hat{D}$ and $\hat{C}$ for the cloud. Thus, the state of task T_n can be represent as $\mathbf{S}_n = (D_{T_n}, C_{T_n}, \hat{D}_n, \hat{C}_n, \hat{D}_{L_n}, \hat{C}_{L_n}, \underset{\forall m \in \mathcal{M}}{\cup}(\hat{D}_m, \hat{C}_m), \hat{D}, \hat{C})$.

2) System Action: In this part, we first define the system action and then introduce the mapping from system action to task offloading policy and resource allocation strategy for determining these two indicators simultaneously.

i) Definition: For each task T_n, it can be processed locally, offloaded to the connected router/ESs, offloaded to the switches, or offloaded to the remote cloud, we denote the action of T_n as $\mathbf{A}_n$, which is defined as follows.

ii) Action Mapping: Moreover, we simultaneously decide the CPU resources allocated for processing the task, as shown in Fig. 2. To this end, we consider that the allocated CPU core(s) for task T_n as a range $[1, 2, \ldots, \Gamma]$, which means T_n should be allocated at least 1 CPU core and at most Γ CPU core(s). We set the action of task T_n as $\mathbf{A}_n \in \{0, 1, \ldots, (M+3) \times \Gamma - 1\}$, and let $mod(\mathbf{A}_n, \Gamma) = q_n, r_n$, where $mod(\cdot)$ denotes $\mathbf{A}_n$ divided by Γ, $q_n \in \{0, 1, \ldots, M, M+1, M+2\}$ and $r_n \in \{0, 1, \ldots, \Gamma - 1\}$ represent the quotient and remainder, respectively. At last, let $q'_n = q_n - 1$ and $r'_n = r_n + 1$, we have $q'_n \in \{-1, 0, \ldots, M, M+1\}$ and $r_n \in \{1, 2, \ldots, \Gamma\}$, and q'_n and r'_n can be map to π_n and γ_n, the details of action mapping is provided in Algorithm 1 (see Line 6-10).

3) System Reward: To improve the decision-making performance of the agent, after executing each action, the agent obtains feedback from the system to train the model parameters. According to the optimization goal of this paper, we set the reward of T_n as $\mathbf{R}_n = \theta\, exp(-(\alpha t_n + \beta e_n)/\Upsilon), \forall n \in \mathcal{N}$, where θ and Υ are variables to control the range of the reward. Moreover, when the agents adopt actions that don't meet the resource constraints, we set negative feedback and set $\mathbf{R}_n$ as a penalty variable $\eta < 0$.

B. Double DQN Model

We use the double DQN model in this paper for training the agent to avoid possible overestimation of the conventional DQN, the whole process is shown in Fig. 3. Specifically, we consider the tasks to form a queue to be offloaded by the agent, each task observes the current state (the requested

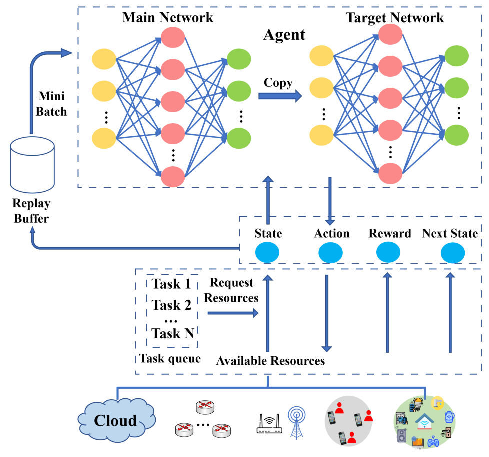


Fig. 3. The DRL-based INC-enhanced task offloading framework in MEC networks.

storage and calculation resources of the task, as well as the available resource of all facilities), then the agent decides the action for the task. After the action is executed based on the mapping policy, the agent can get feedback and also the available resources of facilities will change due to task offloading, which will be the next state of the system.

Specifically, the double DQN model leverages two neural networks (target network and main network) for action selection and evaluation, and the Q-function can be given as

$$Q(\mathbf{S}_n, \mathbf{A}_n) = \mathbf{R}_n + \gamma \cdot \sum_{\mathbf{S}_{n+1}} Pr\{\mathbf{S}_{n+1} | \mathbf{S}_n, \mathbf{A}_n\} \cdot \max_{\mathbf{A}_{n+1}} Q(\mathbf{S}_{n+1}, \mathbf{A}_{n+1}). \tag{10}$$

We adopt deep neural network (DNN) to approximate $Q(\mathbf{S}_n, \mathbf{A}_n)$ and update the parameters of DNN by the stored experience in the replay buffer. Let $Q_i(\mathbf{S}_n, \mathbf{A}_n; \theta_n)$ denote the DQN model with parameters in episode i, we have

$$Q_{i+1}(\mathbf{S}_n, \mathbf{A}_n; \theta_n) = Q_i(\mathbf{S}_n, \mathbf{A}_n; \theta_n) + \alpha^i \cdot (\mathbf{R}_n + \gamma \cdot \max_{\mathbf{A}_{n+1}} Q_i(\mathbf{S}_{n+1}, \mathbf{A}_{n+1}; \hat{\theta}_n) - Q_i(\mathbf{S}_n, \mathbf{A}_n; \theta_n)), \tag{11}$$

where α^i denotes the learning rate, θ_n and $\hat{\theta}_n$ denote the parameters of the main network and the target network, respectively. The main network's loss function used for updating the parameters by gradient descent can be expressed as

$$L(\theta_n) = \sum_{n \in \mathcal{B}_n} (y_i - Q_i(\mathbf{S}_n, \mathbf{A}_n; \theta_n))^2, \tag{12}$$

where $y_i = \mathbf{R}_n + \gamma \cdot \max_{\mathbf{A}_{n+1}} Q_{i+1}(\mathbf{S}_{n+1}, \mathbf{A}_{n+1}; \hat{\theta}_n)$. Here $\mathbf{R}_n$ denotes the reward when offloading T_n in episode i, $\mathcal{B}_n$ denotes a mini-batch when offloading T_n. Algorithm 1 illustrates the proposed double DQN-based INC-enhanced task offloading algorithm. Let l and τ denote the numbers of layers and transitions in each layer, and N_n denote the number of transitions randomly sampled, the complexity of Algorithm 1 for training the parameters can be expressed as $\mathcal{O}(N_n l \tau)$.

Algorithm 1: Double DQN Based INC Enhanced Task Offloading Algorithm.

```
   Input: $\mathbf{S}_n$, $\mathcal{N}$, $\mathcal{M}$, $\gamma$, $\Gamma$.
1  Initialize: The Q-function $Q(\mathbf{S}_n, \mathbf{A}_n; \theta_n)$ of the target
     network with random $\theta_n$, the decay rate of $\epsilon$ as $\xi$, the
     episode index $i = 1$.
2  for $i \leq$ Epsisode Number do
3      Get the system state $\mathbf{S}_n$ from the environment.
4      for $n \in \mathcal{N}$ do
5          With probability $\epsilon$, select an action $\mathbf{A}_n$
             randomly, otherwise select
             $\mathbf{A}_n = \arg\max_{\mathbf{A}_n} Q_i(\mathbf{S}_n, \mathbf{A}_n; \theta_n)$.
6          //Action Mapping
7          Calculate $\mathbf{A}_n \div \Gamma$, get the quotient and
             remainder $q_n$ and $r_n$.
8          Set $\pi_n \leftarrow q_n - 1$.
9          Set $\gamma_n \leftarrow r_n + 1$.
10         Offload $T_n$ to the corresponding device that $\pi_n$
             represents, and allocate $\gamma_n$ CPU cores.
11         Get the reward $\mathbf{R}_n$ and the next-state $\mathbf{S}_{n+1}$.
12         Store the data $(\mathbf{S}_n, \mathbf{A}_n, \mathbf{R}_n, \mathbf{S}_{n+1})$ in the replay
             buffer.
13         Randomly sample from the replay buffer.
14         Calculate the loss function $L(\theta_n)$ by (12),
             update $\theta_n$ to minimize $L(\theta_n)$ by gradient
             descent.
15         $i \leftarrow i + 1$.
16         Update $\epsilon \leftarrow e^{-i/\xi}$.
17 Update the parameters in the target network
     periodically, i.e., $\hat{\theta}_n \leftarrow \theta_n$ after several episodes.
```

IV. Simulation Results

In this section, we evaluate our proposed scheme and compare it with other baselines.

A. Setup and Baseline

We evaluate the proposed INC-enhanced task offloading scheme with several areas (an area defined as a cell), each of which consists of a router/ES (depending on the indoor or outdoor scenario) and 5 UEs. We consider the system consists of 20 switches and one cloud swarm. The total storage resources of the UEs, the routers/switches, the ESs, and the cloud swarm are set as 100 Mbits, 2 Gbits, 5 Gbits, and 1 Tbits, respectively; the total computation resources (in CPU cores) are set as 1, 32, 128, and 6400 cores, respectively. The computation capability of a CPU core is set as 2×10^9 CPU cycles. The data sizes of the tasks are randomly set as $[80, 100]$ Mbits. The computation intensities of tasks are set as 300 Cycles/bit. The uplink bandwidth of UE is set as 5 MHz, and the transmission speed of optical fiber is set as 1 Gbits/s. The latency and EC for transmitting a task from L_n to the cloud are set as 10 s and 1 J. The transmit power of wireless communication and optical fibers are set as 1 W and 0.1 W, respectively. The channel gain is set as $127 + 30 \times log(d)$, where d represents the distance in meters. The Gaussian noise power σ^2 is set as 2×10^{-13} W [11], [17], [18]. The weights α and β for measuring the importance of system latency and EC are set as 0.7 and 0.3, respectively.

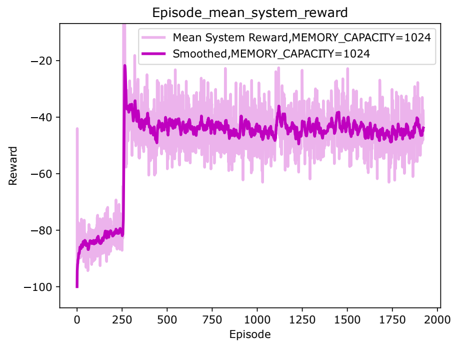


Fig. 4. The system reward.

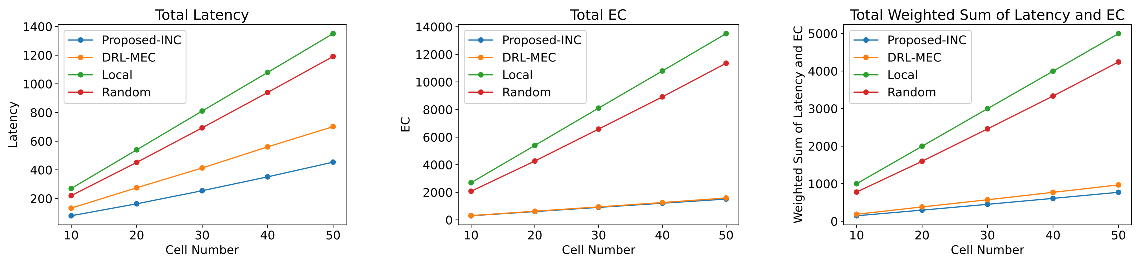
We compare our proposed framework to conventional DRL-based MEC task offloading (defined as *DRL-MEC*) [19] to evaluate that INC can enhance edge offloading; moreover, we also compare the proposed scheme to random offloading (randomly choose action and if failed, process the task locally) and local processing in INC-enhanced networks to see how much the proposed scheme can improve.

B. Evaluation Results

We first present the system reward of the agent to see the convergence performance of the proposed INC-enhanced task offloading algorithm, as shown in Fig. 4. The system reward gradually increases at the beginning episodes of training, followed by a shape rise at the 256-th episode (which is the batch size set in the experiment). Since after the stored experience of the agent is more than the batch size, the agent starts learning. Afterwards, the system reward declines slowly and eventually stabilized and converged after 500 episodes.

With different numbers of cells, we compare the proposed scheme with the DRL-MEC, random offloading, and local processing schemes as shown in Fig. 5. From Fig. 5(a), we observe that the system latency increases with the number of cells, as the number of UEs increases with cells. The proposed scheme outperforms other schemes in all cell numbers and gets more obvious advantages as the number of cells increases, since when we have more UEs, the proposed scheme can offload more tasks to the INC devices, contributing to much lower system latency. Overall, the proposed scheme can reduce the system latency by up to 40.63%, 63.86%, and 70.37% compared to DRL-MEC task offloading, random offloading, and local processing schemes, respectively.

Moreover, in Fig. 5(b), we compare the system EC of the four schemes. Similar to the system latency, the system EC of all schemes increases with the cell number, while the proposed scheme has a slight advantage over the DRL-MEC scheme since these two schemes can both effectively



(a) System latency. (b) System EC. (c) Weighted sum of system latency and EC.

Fig. 5. The system latency, EC, and the weighted sum of the considered system versus different cell numbers.

offload the tasks to the facilities connected to the power supply, which have the same power for every CPU core. On the other hand, the EC of random offloading and local processing is much higher and increases with the number of cells. Overall, the proposed INC-enhanced task offloading and DRL-MEC offloading schemes can reduce the system EC by up to 86.71% and 88.86% compared to random offloading and local processing, respectively.

At last, we summarize the weighted sum of system latency and EC achieved by these schemes, as shown in Fig. 5(c). Overall, the proposed INC-enhanced task offloading scheme can reduce the weighted sum of system latency and EC by up to 22.42%, 81.83%, and 85.36% compared to the DRL-MEC, random offloading, and local processing schemes, respectively.

V. Conclusion

In this paper, we have considered INC-enhanced task offloading in MEC networks to reduce system latency and EC. To this end, we have designed a three-layer end-edge-cloud network architecture with INC elements considered and then proposed a DRL-based framework to solve the formulated optimization problem. Particularly, we map the action of the agent to offloading policy and resource allocation strategy to simultaneously determine these two indicators. Simulation results have demonstrated that our proposed INC-enhanced task offloading scheme achieves fast convergence and outperforms other schemes in reducing system latency and EC.

Acknowledgments

This research work is partially supported by the Business Finland 6Bridge 6Core project under Grant No. 8410/31/2022, the Academy of Finland 6G Flagship program under Grant No. 346208, and the Academy of Finland IDEA-MILL project under Grant No. 352428.

References

[1] M. Shokrnezhad and T. Taleb, "Near-optimal cloud-network integrated resource allocation for latency-sensitive b5g," in *Proc. IEEE Global Communications Conference (GLOBECOM)*, 2022, pp. 4498–4503.

[2] H. Mazandarani, M. Shokrnezhad, T. Taleb, and R. Li, "Self-sustaining multiple access with continual deep reinforcement learning for dynamic metaverse applications," in *Proc. IEEE International Conference on Metaverse Computing, Networking and Applications (MetaCom)*, 2023.

[3] O. A. Wahab, A. Mourad, H. Otrok, and T. Taleb, "Federated machine learning: Survey, multi-level classification, desirable criteria and future directions in communication and networking systems," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 2, pp. 1342–1397, 2021.

[4] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1657–1681, 2017.

[5] Z. Ming, X. Li, C. Sun, Q. Fan, X. Wang, and V. C. M. Leung, "Sleeping cell detection for resiliency enhancements in 5g/b5g mobile edge-cloud computing networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 18, no. 3, pp. 1–30, 2022.

[6] H. Yu, Z. Ming, C. Wang, and T. Taleb, "Network slice mobility for 6g networks by exploiting user and network prediction," in *Proc. IEEE International Conference on Communications (ICC)*, 2023.

[7] R. A. Addad, D. L. C. Dutra, T. Taleb, and H. Flinck, "Toward using reinforcement learning for trigger selection in network slice mobility," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 7, pp. 2241–2253, 2021.

[8] I. Maity and T. Taleb, "Resq: Reinforcement learning-based queue allocation in software-defined queuing framework," *Journal of Networking and Network Applications*, vol. 2, no. 4, pp. 143–152, 2022.

[9] L. Hu, Y. Tian, J. Yang, T. Taleb, L. Xiang, and Y. Hao, "Ready player one: Uav-clustering-based multi-task offloading for vehicular vr/ar gaming," *IEEE Network*, vol. 33, no. 3, pp. 42–48, 2019.

[10] Y. Chen, Y. Sun, C. Wang, and T. Taleb, "Dynamic task allocation and service migration in edge-cloud iot system based on deep reinforcement learning," *IEEE Internet of Things Journal*, vol. 9, no. 18, pp. 16 742–16 757, 2022.

[11] Z. Ming, X. Li, C. Sun, Q. Fan, X. Wang, and V. C. M. Leung, "Dependency-aware hybrid task offloading in mobile edge computing networks," in *Proc. IEEE International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2021, pp. 225–232.

[12] S. Kianpisheh and T. Taleb, "A survey on in-network computing: Programmable data plane and technology specific applications," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 1, pp. 701–761, 2023.

[13] T. Mai, H. Yao, S. Guo, and Y. Liu, "In-network computing powered mobile edge: Toward high performance industrial iot," *IEEE Network*, vol. 35, no. 1, pp. 289–295, 2021.

[14] L. Gobatto, M. Saquetti, C. Diniz, B. Zatt, W. Cordeiro, and J. R. Azambuja, "Improving content-aware video streaming in congested networks with in-network computing," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, 2022, pp. 1813–1817.

[15] L. Hu, Y. Chai, Q. Li, W. Li, and Y. Zhang, "Multi-source dnn task offloading strategy based on in-network computing," in *Proc. International Conference on Advanced Communication Technology (ICACT)*, 2023, pp. 226–231.

[16] N. Hu, Z. Tian, X. Du, and M. Guizani, "An energy-efficient in-network computing paradigm for 6g," *IEEE Transactions on Green Communications and Networking*, vol. 5, no. 4, pp. 1722–1733, 2021.

[17] C. Sun, X. Wu, X. Li, Q. Fan, J. Wen, and V. C. M. Leung, "Cooperative computation offloading for multi-access edge computing in 6g mobile networks via soft actor critic," *IEEE Transactions on Network Science and Engineering*, 2021.

[18] J. Li, Z. Yang, K. Chen, Z. Ming, X. Li, Q. Fan, J. Hao, and L. Cheng, "Dependency-aware task offloading based on deep reinforcement learning in mobile edge computing networks," *Wireless Networks*, pp. 1–13, 2023.

[19] D. C. Nguyen, P. N. Pathirana, M. Ding, and A. Seneviratne, "Privacy-preserved task offloading in mobile blockchain with deep reinforcement learning," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2536–2549, 2020.