Machine Learning for Caching Placement in Edge Computing Networks

Liang Zhang, *Member, IEEE*, and Bijan Jabbari, *Fellow, IEEE*
Department of Electrical and Computer Engineering, George Mason University, Fairfax, VA 22030, USA

***Abstract*—Internet of Things devices (*IoTD*s) that deployed for various applications have limited size, power, storage, and computing capabilities, and many of them are used for low-latency applications. Mobile edge computing (*MEC*) can be leveraged to reduce the latency of IoTDs, and judicious caching placement can help further to improve the quality of service. In this paper, we study multi-content placement (*MCP*) in the MEC networks for IoTDs by considering the background data caching in the edge nodes and the data collection of IoTDs. The MCP problem is formulated by considering the caching, the IoTD assignment, and the communication and computing resources allocation to minimize the average latency of all IoTDs. As the MCP problem is NP-hard, we propose a deep reinforcement machine learning algorithm to obtain the caching placement and IoTD assignment jointly to solve the MCP problem. We use an optimal joint resource-scheduling algorithm to assign the resources. Our results demonstrate that considerable latency improvement can be achieved through the proposed deep reinforcement machine learning algorithm as compared to baseline algorithms.**

***Index Terms*—Machine learning (*ML*), Internet of Things (*IoT*), mobile edge computing, caching placement, latency.**

I. Introduction

Billions of Internet of Things (*IoT*s) equipment is deployed for various applications such as smart homes, smart transportation, autonomous driving, and augmented reality, and many of them have been applied to low-latency applications [1]. Due to the limited size and weight, Internet of Things devices (*IoTD*s) are equipped with limited resources and capabilities such as battery, storage, and computing capacity. Meanwhile, numerous IoT applications have low latency requirements, which impose challenges in executing tasks of IoTDs [2]. Mobile edge computing (*MEC*) is an attractive solution that can overcome the hurdle by providing computation offloading and communication services to IoTDs through the deployment of computing facilities at the edge of the wireless networks, and then the latency and energy consumption of IoTDs can be reduced by MEC [2]–[4].

To execute the task of an IoTD in an edge node, the IoTD needs to transmit the collected data to the edge node and the edge node also needs to cache the background data (the related database) for computing; the computing resources of the edge node are assigned to this IoTD to provide computing services; and the computing results are returned to the IoTD [5]. To make the IoTD task offloading efficiently, the edge nodes need to cache the related database/relevant data; if the related database/relevant data is not cached, it must be downloaded from the cloud first, which will increase the total latency of the service [5]. A good example is intelligent monitoring in the smart grid: executing the computing tasks from IoTDs needs not only sensed data (the appearance, the operating status, and the abnormal conditions) but also the surrounding infrastructure data (weather conditions, protection, etc.) [5]. Due to the limited caching capacity, the related database/relevant data of all IoTDs cannot be cached at edge nodes. Therefore, what and where to cache the related database/relevant data need to be carefully designed.

Some studies related to edge computing and cache placement have been reported in the literature. Zhou *et al.* [5] studied the joint computing offloading and caching in the edge computing networks for smart-grid with the target to minimize the total network cost. Bi *et al.* [3] minimized the energy consumption of mobile users in a network with one server by considering the joint caching placement, offloading, and resource allocation. Chen *et al.* [6] investigated the problem of cache placement and bandwidth assignment in a caching-enabled MEC network with the objective to minimize the energy consumption of the edge nodes and users. Chang *et al.* [7] leveraged the cooperation of fog access points to optimize the transmission delay by the best caching strategy. Gu *et al.* [8] designed a caching replacement strategy by using the dynamic game model. Yang *et al.* [9] minimized the average utility consumption via the cooperation of the caching placement of adjacent edge nodes.

Edge caching enables edge offloading by executing the computing tasks on edge servers instead of transmitting data to the cloud and receiving the computing service from the cloud, thus reducing the latency and energy consumption [5]; In addition, a well-planned content placement can further improve the latency and the quality of service. Although there are some existing works related to MEC and caching, only a few works have investigated the latency problem in MEC networks with multi-content placement, especially with considering the background data caching in the edge nodes and the data collection of IoTDs.

Liang Zhang and Bijan Jabbari are with the Communications and Networks Laboratory, Department of Electrical and Computing Engineering, George Mason University, Fairfax, VA 22030 USA (email: lzhang36@gmu.edu; bjabbari@gmu.edu). This work was supported in part by National Science Foundation under Grant NSF 2029221.

979-8-3503-7099-7/24/$31.00 ©2024 IEEE

In this paper, we formulate the multi-content placement (*MCP*) problem in edge-computing networks for IoTDs with the target of minimizing the average latency of IoTDs.

The major contributions are summarized as follows: 1) We study multi-content placement in the MEC network for IoTDs while considering the background data caching in the edge nodes and the data collection of IoTDs. We consider the uplink data transmission from IoTD to edge nodes, the caching status of the background data/related database, fetching the background data/related database from the cloud, and executing the offloading tasks of the IoTDs in edge nodes. 2) We formulate the MCP problem by considering the caching, the IoTD assignment, the computing resource, and the communication resource allocation. 3) We design an optimal joint resource scheduling algorithm to assign the communication and computing resources to IoTDs. 4) We propose a deep reinforcement machine learning algorithm to solve the MCP problem by determining the joint caching and IoTD assignment and obtaining the resource allocation results from the optimal joint resource scheduling algorithm. 5) We evaluate the performance of the proposed machine learning algorithm with comparisons to baseline algorithms.

II. System Model

The multi-content caching scenario in the MEC network is illustrated in Fig. 1, which includes IoTDs, edge nodes, and the cloud. All edge nodes are equipped with computing and communication facilities, which can cache any background data/related database within their capacities to perform computing tasks. To clarify, the content refers to the required background data to provision the tasks of IoTDs and we may interchangeably use background data/related database and content. Different types of applications require different required background data, implying that various contents are used. Note that an edge node cannot cache all contents due to the limited storage capacity, and all of them are available in the cloud.

An IoTD can transmit data to an edge node and then receive the communication and computing service in an edge node. If the edge node has cached the corresponding content for this IoTD, the computing service is executed on the edge node directly; otherwise, the edge node needs to fetch the relevant content from the cloud first and then serve the computing tasks. Note that the communication resources are shared by the IoTDs in the frequency multiplexing scheme. To make communications more efficient, different IoTDs use different frequency spectra for communications and different edge nodes also utilize different frequency spectra for communications.

Denote $\mathscr{B}$, $\mathscr{U}$ and $\mathscr{K}$ as the set of edge nodes, IoTDs, and contents, respectively. Let $A_i = \{c_i, r_i, e_i, y_i\}$ be the computing task of IoTD i. Here, c_i, r_i, e_i and y_i represent the computing requirements in terms of CPU cycles, the data collected by IoTD i, the content type requirement, and the required background data/relevant data for executing the computing task ($i \in \mathscr{B}$, $j \in \mathscr{U}$, and $e_i \in \mathscr{K}$), respectively. Note

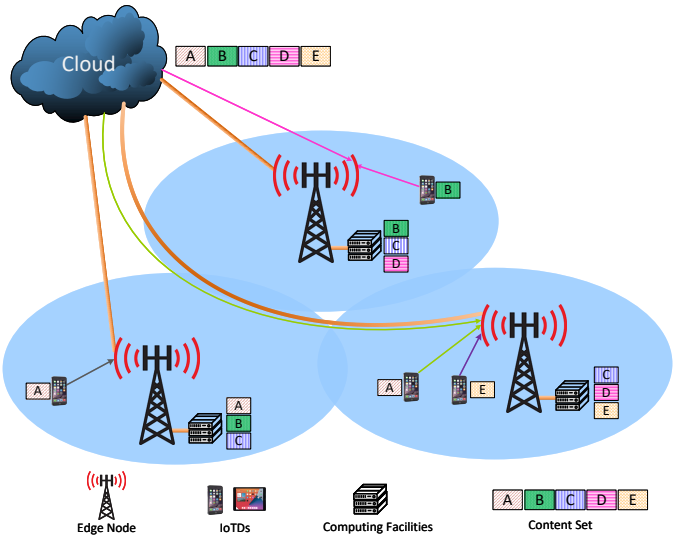

Fig. 1: Multi-content caching in the edge computing network.

TABLE I: Important Notations and Variables

Symbol	Definition
$\mathscr{B}$	the set of edge nodes.
$\mathscr{U}$	the set of IoTDs.
$\mathscr{K}$	the set of contents.
c_i	the computing requirement of IoTD i.
r_i	the data collected by IoTD i.
e_i	the content type requirement of IoTD i.
y_i	the required background data/relevant data for executing the computing task e_i of IoTD i.
P^E	the maximum transmission power of an edge node.
P^I	the maximum transmission power of an IoTD.
C_j	the computing resource capacity of edge node j.
k^{max}	the caching capacity in terms of type of contents.
$d_{i,j}$	the data rate from IoTD i towards edge node j.
ξ_j	the bandwidth capacity of edge node j.
β_0	bandwidth of one resource block.
$t_{i,j}$	the total service delay.
$t^1_{i,j}$	the total service delay if the content is cached.
$t^2_{i,j}$	the total service delay if the content is not cached.
α^k_j	the indicator of the caching status of content k in edge node j.
$\omega_{i,j}$	the indicator of service status of IoTD i by edge node j
$\beta_{i,j}$	the assigned bandwidth to IoTD i by edge node j.
$\zeta_{i,j}$	the assigned computation resource to IoTD i by edge node j.

that different IoTDs may have different content requirements. For the same content requirements from different IoTDs, we assume the requirements are the same. If one type of content is cached in an edge node, then it can be used to serve many IoTDs with the same content requirements. Important notations are summarized in Table I.

We assume the popularity of the contents of IoTDs follows the Zipf distribution, and the probability of requesting content k by IoTD i is:

$$q_i^k = \frac{\xi^{-k}}{\sum_1^K \xi^{-k}}, \tag{1}$$

where ξ is the skewness parameter of the Zipf distribution; a

larger ξ means that more IoTDs are interested in high ranked contents [8].

Let $d_{i,j}$ and $s_{i,j}$ be the data rate and the signal to interference plus noise ratio (*SINR*) of IoTD i towards the edge node j. Then, we have:

$$d_{i,j} = \beta_0 \beta_{i,j} log_2(1 + s_{i,j}), \quad \forall i \in \mathscr{U}, j \in \mathscr{B}, \tag{2}$$

where $\beta_{i,j}$ is the assigned frequency spectra to IoTD i and β_0 is the bandwidth of one resource block.

$$s_{i,j} = \frac{P^I \psi_{i,j}}{I_0 + \sigma_{i,j}^2}, \quad \forall i \in \mathscr{U}, j \in \mathscr{B}. \tag{3}$$

Here, P^I is the transmission power of an IoTD; $\psi_{i,j}$ is the path loss between IoTD i and edge node j; $\sigma_{i,j}^2 = \beta_0 \beta_{i,j} N_0$ is the thermal noise power; N_0 is a constant that represents the thermal noise power spectral density; we assume that efficient interference management techniques are utilized to reduce the interference and the power interference I_0 is fixed [10].

Denote α_j^k as the indicator of the caching status of content k in edge node j; it is a binary variable and equals to 1 if content k is cached by edge node j; otherwise, it is 0. Let $\omega_{i,j}$ be the service status indicator of IoTD i by edge node j; it is a binary variable and equals to 1 if IoTD i is served by edge node j; otherwise, it is 0. Assuming $\beta_{i,j}$ and $\zeta_{i,j}$ are the frequency spectra and computing resource assignment to IoTD i by edge node j.

We assume each IoTD can be served by no more than one edge node. For the service process, IoTD i needs to transmit the collected data r_i to edge node j, and then IoTD i is served by edge node j if the background data/related database y_i is cached ($\alpha_j^k = 1$); otherwise, the edge node needs to obtain y_i from the cloud and the additional delay $t_{i,j}^{cloud}$ is introduced, which is composed of the round-trip propagation delay, the transmission delay, and the processing delay from the cloud. Since the transmission speed from the cloud to the edge node is high and the size of the background data/relevant data is small, and then we assume $t_{i,j}^{cloud}$ is fixed and $t_{i,j}^{cloud} = t^{cloud}$. Here, the requested content from the cloud is temporarily stored at the edge node due to the limited caching capacity. In this paper, we do not consider the IoTDs served by the cloud directly.

Let $t_{i,j}$, $t_{i,j}^1$ and $t_{i,j}^2$ be the total service delay, the total service delay when the content is cached ($\alpha_j^k = 1$), and the total service delay when the required content/relevant data is not cached ($\alpha_j^k = 0$) of IoTD i provisioned by edge node j, respectively, as described below:

$$t_{i,j} = \alpha_j^k t_{i,j}^1 + (1 - \alpha_j^k) t_{i,j}^2, \quad \forall i \in \mathscr{U}, \tag{4}$$

$$t_{i,j}^1 = \frac{r_i}{d_{i,j}} + \frac{c_i}{\zeta_{i,j}}, \quad \forall i \in \mathscr{U}, \tag{5}$$

$$t_{i,j}^2 = \frac{r_i}{d_{i,j}} + \frac{c_i}{\zeta_{i,j}} + t_{i,j}^{cloud}, \quad \forall i \in \mathscr{U}, \tag{6}$$

Here, $r_i/d_{i,j}$ is the transmission delay; $c_i/\zeta_{i,j}$ is the computing delay; $\zeta_{i,j}$ is the assigned computing resource in terms of CPU cycles by edge node j.

III. Problem formulation

In this paper, we focus on minimizing the average latency of IoTDs by jointly considering the caching placement, the IoTD assignment, the computing resource, and the communication resource assignment. The MCP problem is formulated as follows:

$$\begin{aligned}
\mathscr{P}_0 : & \min_{\alpha_j^k, \omega_{i,j}, \beta_{i,j}, \zeta_{i,j}} \quad \frac{1}{|\mathscr{U}|} \sum_i \sum_j t_{i,j} \\
s.t. : & \\
& C1 : \sum_k \alpha_j^k \le \Gamma, \quad \forall j \in \mathscr{E}, \\
& C2 : \sum_j \omega_{i,j} \le 1, \quad \forall i \in \mathscr{U}, \\
& C3 : \sum_i \omega_{i,j} \beta_{i,j} \le f_j, \quad \forall j \in \mathscr{E}, \\
& C4 : \sum_i \omega_{i,j} \zeta_{i,j} \le C_j, \quad \forall j \in \mathscr{E}, \\
& C5 : \alpha_j^k \in \{0,1\}, \quad \forall j \in \mathscr{E}, k \in \mathscr{K}. \\
& C6 : \omega_{i,j} \in \{0,1\}, \quad \forall i \in \mathscr{U}, j \in \mathscr{E}.
\end{aligned} \tag{7}$$

Here, C1 and C5 are content placement constraints, which ensure that the cached contents do not exceed the caching capacity. C2 and C6 are IoTD serving constraints, implying that each IoTD can be provisioned by one edge node at most. C3 and C4 are frequency spectra and computing resource capacity constraints to ensure that the used frequency spectra and computing resource are below or equal to the maximum available resource of each edge node.

IV. Analysis

The MCP problem is NP-hard because it is a non-convex, nonlinear, and mixed discrete optimization problem [11], [12]. Then, a machine learning algorithm is proposed to solve the MCP problem. First, we focus on the resource assignment based on the given caching placement and IoTD assignment. Second, we employ a machine learning algorithm to obtain the best result of the caching placement and the IoTD assignment, and then the MCP problem is solved.

For a given α_j^k and $\omega_{i,j}$, the MCP problem can be reformulated as follows:

$$\begin{aligned}
\mathscr{P}_1 : & \min_{\beta_{i,j}, \zeta_{i,j}} \quad \frac{1}{|\mathscr{U}|} \sum_i \sum_j t_{i,j} \\
s.t. : & \\
& C1 : \sum_i \omega_{i,j} \beta_{i,j} \le f_j, \quad \forall j \in \mathscr{E}, \\
& C2 : \sum_i \omega_{i,j} \zeta_{i,j} \le C_j, \quad \forall j \in \mathscr{E}.
\end{aligned} \tag{8}$$

Since each IoTD requires both communication and computing resources to obtain the service, we assume the computing resource assigned to an IoTD is in proportion to the assigned frequency spectra; the total computing resource is exhausted

Algorithm 1: Optimal Resource Assignment

Input : $\mathscr{B}$, $\mathscr{U}$, f_j, C_j, α_j^k and $\omega_{i,j}$;
Output: $\beta_{i,j}$ and $\zeta_{i,j}$;
for j *in* $\mathscr{B}$ **do**
  determine $\mathscr{U}_j = \{\omega_{i,j} = 1\}$;
  initialize $t_{i,j}$, $\Delta t_{i,j}$, β_0 and ζ_0;
  set $f_j^1 = f_j$, $C_j^1 = C_j$, $f_j^2 = 0$, and $C_j^2 = 0$;
  calculate $t_{i,j}^{total} = \sum_i \sum_j t_{i,j}$;
  while $f_j^1 \geq \Delta b \,\&\, C_j^1 \geq \Delta\tau$ **do**
    for i *in* $\mathscr{U}_j$ **do**
      update $t_{i,j}$ by $t_{i,j}^{'}$ if β_0 and ζ_0 are assigned to this IoTD;
      calculate the latency reduction $\Delta t_{i,j} = t_{i,j} - t_{i,j}^{'}$;
    find $(i', j') = \underset{i}{\operatorname{argmax}} \Delta t_{i,j}$;
    assign β_0 and ζ_0 to IoTD i' by edge node j';
    $\beta_{i',j'} = \beta_{i',j'} + \beta_0$;
    $\zeta_{i',j'} = \zeta_{i',j'} + \zeta_0$;
    update $t_{i,j}^{total}$;
  update $\beta_{i,j}$ and $\zeta_{i,j}$;

when the total frequency spectra are used. Then, problem $\mathscr{P}_1$ is transformed into problem $\mathscr{P}_2$. Note that problem $\mathscr{P}_2$ is similar to the knapsack problem, but the user assignment is predetermined and the resource granularity is also different, e.g., the granularity of the assignment of problem $\mathscr{P}_2$ can be set as one resource unit. Then, we can obtain the optimal solution for the resource assignment, and *Algorithm* 1 is proposed to solve problem $\mathscr{P}_2$. This is because the same amount of computing resource ζ_0 and frequency spectrum β_0 of edge node j' are assigned to IoTD i' with the largest latency reduction in the total latency of all IoTDs by *Algorithm* 1 for each assignment. Here, $(i', j') = \underset{i}{\operatorname{argmax}} \Delta t_{i,j}$.

$$\mathscr{P}_2 : \min_{\beta_{i,j}, \zeta_{i,j}} \quad \frac{1}{|\mathscr{U}|} \sum_i \sum_j t_{i,j}$$

$$s.t. :$$

$$C1 : \sum_i \omega_{i,j} \beta_{i,j} \leq f_j, \quad \forall j \in \mathscr{E}. \tag{9}$$

As machine learning can achieve efficient solutions for complicated optimization problems [13], [14], a deep reinforcement learning technique is leveraged to solve the MCP problem. Deep reinforcement learning is a combination of deep learning and reinforcement learning; the neural networks are employed to learn the environment and generate actions based on the input; all neural networks are fully connected; in each neural network, there is a weight of each link between two neighbor neurons, which is updated based on the reward, the learning strategy and the environment [14], [15]. The input includes the current state, the action, and the next state. The output is the action, which is utilized to calculate the reward. To proceed with our deep reinforcement learning algorithm, we need to define the state, action, and reward as follows:

- State: $s(n) = \{XY(\mathscr{B}), XY(\mathscr{U}), c_i, e_i, r_i\}$. Here, $XY(\cdot)$ is the function to determine the locations (X-axis and Y-axis) of IoTDs/edge nodes in the cartesian coordinate system.
- Action: $a(n) = \{\alpha_{i,j}, \omega_{i,j}\}$. It represents the caching status and the IoTD assignment.
- Reward: $g(n) = -\frac{1}{|\mathscr{U}|} \sum_i \sum_j t_{i,j}$. Note that we need to set the negative value of the average latency of IoTDs for the reward because our machine learning agent can only maximize the objective value.

The deep deterministic policy gradient algorithm is proposed to solve the MCP problem (*DDPG-MCP*), as expressed in Algorithm 2. Four neural networks are utilized in the deep deterministic policy gradient algorithm:
1) actor network $a(n) = \phi(s(n)|\theta^{\phi})$,
2) target actor network $a(n) = \phi(s(n)|\theta^{\phi-})$,
3) critic network $Q(s(n), a(n)|\theta^{Q})$,
4) target critic network $Q(s(n), a(n)|\theta^{Q-})$.

Algorithm 2: DDPG-MCP

Input : $\mathscr{B}$, $\mathscr{U}$ and four neural networks;
Output: $g(n)$, α_j^k and $\omega_{i,j}$;
for *epoch* m **do**
  Initialize the actor network with the weight θ^{ϕ};
  set the target actor network with the weight $\theta^{\phi-}$;
  Initialize the critic network with the weight θ^{Q};
  set the target critic network with the weight θ^{Q-};
  set the replay buffer $\eta = 0$ and initialize η^b;
  $s(n) = 0$, $a(n) = 0$ and $g(n) = 0$;
  for *each training step* n **do**
    calculate state $s(n+1)$;
    add $\{s(n), a(n), g(n), s(n+1)\}$ to replay buffer;
    $\eta = \eta + 1$;
    if $\eta \geq \eta^b$ **then**
      update the critic network θ^Q by Eq. (10);
      update actor network θ^{ϕ} by Eq. (11);
      update target networks;
    get $a(n+1)$;
    add noise: $a(n+1) = a(n+1) + \sigma(n+1)$;
    decode $a(n+1)$ to obtain α_j^k and $\omega_{i,j}$ by Algorithm 1;
    calculate $g(n+1)$;
  find the largest reward $g(n')$;
  obtain α_j^k and $\omega_{i,j}$ by $(\alpha_j^k, \omega_{i,j}) = \underset{i,j,k}{\operatorname{argmax}} \cup g(n)$;
  return $g(n)$, α_j^k and $\omega_{i,j}$.

The critic network is updated as follows:

$$L(\theta^Q) = \frac{1}{\Gamma}\sum_{\gamma=1}^{\Gamma}[g(\gamma) - Q(s(\gamma), a(\gamma)|\theta^Q) + \lambda A_1]^2. \quad (10)$$

The actor network is updated below:

$$\bigtriangledown_{\theta^\phi}\Omega(\theta^\phi) = \frac{1}{\Gamma}\sum_{\gamma=1}^{\Gamma}(A_2 \bigtriangledown_a Q(s, a|\theta^Q)\big|_{s=s(\gamma),a=\phi(s(\gamma))}). \quad (11)$$

The target networks is updated by $\theta^{\phi-} \leftarrow \varsigma\theta^\phi + (1-\varsigma)\theta^{\phi-}$ and $\theta^{Q-} \leftarrow \varsigma\theta^Q + (1-\varsigma)\theta^{Q-}$. Here, $A_1 = Q(s(\gamma+1), a(\gamma+1)|\theta^{Q-})$; $A_2 = \bigtriangledown_{\theta^\phi}\ \phi(s|\theta^\phi)\big|_{s=s(\gamma)}$; $g(\gamma)$ stands for the reward of the sample γ and Γ represents the number of samples in a batch, $1 \le \gamma \le \Gamma$; $\bigtriangledown_\theta\Omega(\theta)$ is the gradient function and $\Omega(\theta)$ is used to obtain the policy target [13].

V. Performance Evaluation

We use Python 3.8 and Tensorflow 2.6.2 (tf.keras.optimizers.Adam) to run our simulations. The service area is set as $500\,m \times 500\,m$ and four edge nodes are placed in the fixed locations to serve IoTDs, as expressed in Fig. 2. There are 25 contents with different types, and the skewness factor is set as 0.4 in the simulations. For the IoTD distribution, a Matérn cluster process is leveraged to generate the IoTDs. For the machine learning simulation settings, four neural networks are initialized with identical parameters and all of them have the same neural network configurations; each neural network includes five layers: one input layer, three hidden layers, and one output layer; each hidden layer has 800 neurons [13], [16]. Important simulation parameters are summarized in Table II.

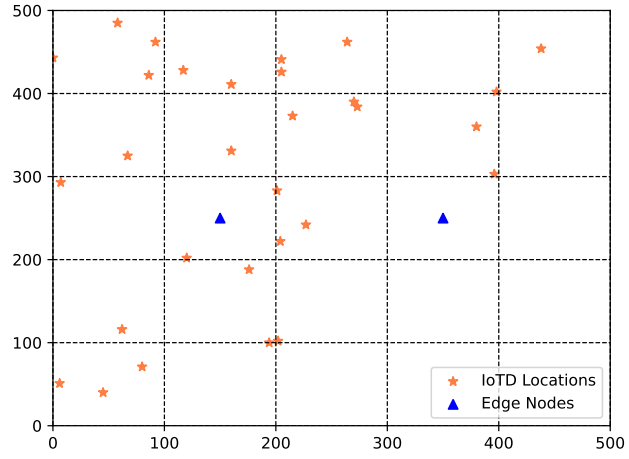

Fig. 2: Locations of edge modes and IoTDs.

To evaluate the performance of the proposed machine learning algorithm, three baseline algorithms are presented below: (1) Fixed-Best-MCP: For the content placement, only the k^{max} contents with the highest popularity are cached in edge nodes; for the IoTD assignment, the edge node with the highest SINR is chosen to serve every IoTD; and the resources are assigned by Algorithm 1. (2) Fixed-Fair-MCP: The content placement and the user assignment are the same as the Fixed-Best-MCP algorithm; the communication and computing resources are equally shared by IoTDs. (3) Random-Fair-MCP: The contents cached in edge nodes are randomly chosen from all contents; all IoTDs equally share both computing and communication resources to achieve the best fairness.

TABLE II: **Simulation Parameters**

Parameters	value
the coverage area	500 m × 500 m
$\lvert\mathcal{B}\rvert$	2
$\lvert\mathcal{U}\rvert$	$\{10, 15, \cdots, 35\}$
$\lvert\mathcal{K}\rvert$	25
r_i, the input data size	$[1, 2]$ Mb
c_i, the computing requirement	$[1, 20] \times 10^7$ CPU cycle
C_j, edge node computing capacity	2×10^{10} CPU cycle/s
$\psi_{i,j}$	$131.1 + 42.8log10(d_{i,j})$, $d_{i,j}$ in km
Rayleigh fading	8 dB
N_0	-174 dBm/Hz
P^I	20 dBm
f_j	50 RB (10 MHz)
β_0	180 kHz
t^{cloud}	100 ms
Machine Learning Parameters	
# of neurons of each hidden layer	800, 800, 800
learning rate of the actor networks	5×10^{-5}
learning rate of the actor networks	5×10^{-5}
learning rate of the critic networks	5×10^{-4}
λ, discount factor	0.99
ς, soft target update	0.001
number of training epochs	6
number of training steps	800
η^b, number of samples in a batch	64
replay buffer capacity	10^5

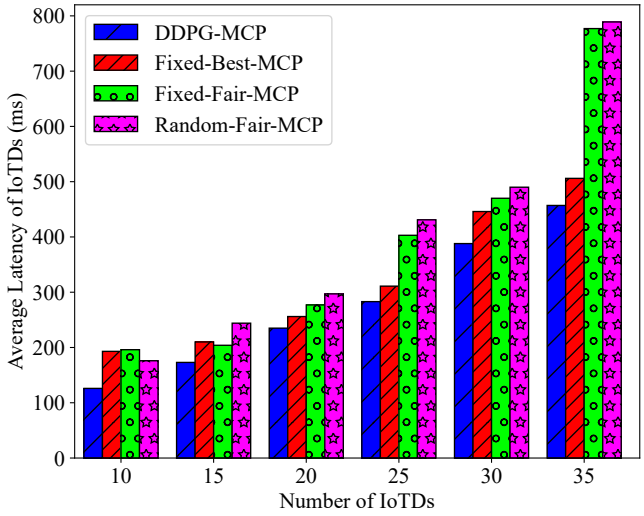

Fig. 3: Average latency of served IoTDs versus # of TDs.

Fig. 3 shows the average latency results versus the number of IoTDs, and the caching capacity k^{max} is set as 10 for each edge

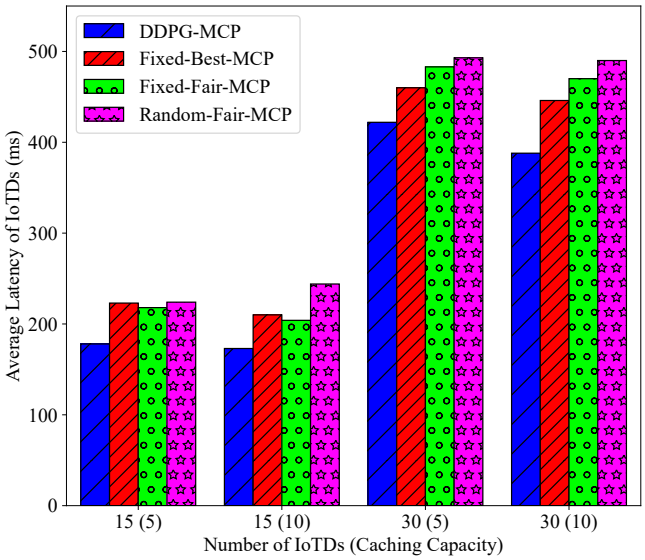

Fig. 4: Average latency of IoTDs versus caching capacity.

node. The average latency of all IoTDs increases as the number of IoTDs increases. This is because the computing resource and frequency spectra assigned to each IoTD decrease when the number of IoTD increases, which leads to high average latency. The DDPG-MCP algorithm can reduce the average latency up to 34.7%, 41.2%, and 42.1% as compared to the Fixed-Best-MCP, the Fixed-Fair-MCP, and the Random-Fair-MCP algorithms, respectively. The Random-Fair-MCP algorithm has the worst performance because the cached contents are randomly selected, which leads to many requests for downloading the background data/related database from the cloud to the edge nodes. The Fixed-Best-MCP algorithm has better performance than the Fixed-Fair-MCP algorithm because a better resource assignment mechanism (Algorithm 1) is used. The average latency of the DDPG-MCP algorithm is better than the Fixed-Best-MCP algorithm because of the better IoTD assignment and the better selection of the cached content.

Fig. 4 shows the average latency results versus different caching capacities with 15 and 30 IoTDs. The number in parenthesis stands for k^{max}, e.g., $15(5)$ represents 15 IoTDs and $k^{max} = 5$. The average latency of all algorithms increases when there are more IoTDs (30 IoTDs). The average latency of the Random-Fair-MCP algorithm does not have many differences because the cached contents in edge nodes are randomly selected, which do not well match the content requirements of IoTDs. The average latency of the DDPG-MCP, the Fixed-Best-MCP, and the Fixed-Fair-MCP algorithms decreases up to 8.1% as the caching capacity increases. This is because a higher number of IoTDs are served by edge nodes without the transmission from the cloud when edge nodes cached more contents. The DDPG-MCP algorithm has the best performance than the rest algorithms because it can obtain the best IoTD assignment and content placement based on the workload.

VI. Conclusion

In this work, we have studied caching in MEC networks by considering background data caching and the data collection of IoTDs. We have formulated the multi-content placement (*MCP*) problem in the MEC networks to minimize the average latency of IoTDs. Since the MCP problem is NP-hard, a deep reinforcement learning algorithm, referred to as DDPG-MCP, is proposed to solve the MCP problem by achieving the best joint caching placement and IoTD assignment and obtaining the resource allocation through the optimal resource scheduling algorithm. The proposed machine learning algorithm and the three baseline algorithms for the MCP problem have been evaluated via extensive simulations in Python. The simulation results have demonstrated that the DDPG-MCP algorithm is superior to the baseline algorithms by up to 42% improvement for the average latency compared to baseline algorithms.

References

[1] C. Qiu *et al.*, "Networking integrated cloud–edge–end in IoT: A blockchain–assisted collective *Q*-learning approach," *IEEE Internet Things J.*, vol. 8, no. 16, pp. 12 694–12 704, Aug. 2021.

[2] P. Wang *et al.*, "Joint task assignment, transmission, and computing resource allocation in multilayer mobile edge computing systems," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2872–2884, Apr. 2019.

[3] S. Bi, L. Huang, and Y.-J. A. Zhang, "Joint optimization of service caching placement and computation offloading in mobile edge computing systems," *IEEE Trans. Wireless Commun.*, vol. 19, no. 7, pp. 4947–4963, 2020.

[4] X. Sun and N. Ansari, "Latency aware workload offloading in the cloudlet network," *IEEE Commun. Lett.*, vol. 21, no. 7, pp. 1481–1484, Jul. 2017.

[5] H. Zhou, Z. Zhang, D. Li, and Z. Su, "Joint optimization of computing offloading and service caching in edge computing-based smart grid," *IEEE Trans. Cloud Comput.*, pp. 1–1, 2022.

[6] J. Chen, H. Xing, X. Lin, and S. Bi, "Joint cache placement and bandwidth allocation for FDMA-based mobile edge computing systems," in *Proc. of IEEE ICC*, 2020, pp. 1–7.

[7] Q. Chang, Y. Jiang, F.-C. Zheng, M. Bennis, and X. You, "Cooperative edge caching via multi agent reinforcement learning in fog radio access networks," in *Proc. of IEEE ICC*, 2022, pp. 3641–3646.

[8] H. Gu, W. Cai, L. Zhao, W. Luo, G. Zhou, Q. Chen, H. Tu, Z. Wang, and S. Li, "Dynamic game-based caching replacement in edge networks," in *IEEE 95th Vehicular Technology Conference: (VTC2022-Spring)*, 2022, pp. 1–5.

[9] S. Yang, S. Fan, G. Deng, and H. Tian, "Local content cloud based cooperative caching placement for edge caching," in *Proc. of IEEE PIMRC*, 2019, pp. 1–6.

[10] J. Liu, B. Bai, J. Zhang, and K. B. Letaief, "Cache placement in Fog-RANs: From centralized to distributed algorithms," *IEEE Trans. Wireless Commun.*, vol. 16, no. 11, pp. 7039–7051, 2017.

[11] R. M. Nauss, "Solving the generalized assignment problem: An optimizing and heuristic approach," *INFORMS Journal on Computing*, vol. 15, no. 3, pp. 249–266, 2003.

[12] L. Zhang and N. Ansari, "Latency-aware IoT service provisioning in UAV-aided mobile-edge computing networks," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 10 573–10 580, Jun. 2020.

[13] L. Wang *et al.*, "Deep reinforcement learning based dynamic trajectory control for UAV-assisted mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 21, no. 10, pp. 3536–3550, Oct. 2022.

[14] L. Zhang, B. Jabbari, and N. Ansari, "Deep reinforcement learning driven uav-assisted edge computing," *IEEE Internet Things J.*, pp. 1–1, 2022.

[15] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," arXiv preprint arXiv:1509.02971, Sept. 2015.

[16] L. Zhang and B. Jabbari, "Machine learning driven latency optimization for application-aware edge computing-based IoTs," in *Proc. of IEEE ICC*, 2022, pp. 183–188.