

Digital Twin-assisted Reinforcement Learning for Resource-aware Microservice Offloading in Edge Computing

Xiangchun Chen, Jiannong Cao, Zhixuan Liang, Yuvraj Sahni, Mingjin Zhang

The Hong Kong Polytechnic University, Hong Kong, China.

{csxcchen, csjcao, cszliang}@comp.polyu.edu.hk, yuvraj-comp.sahni@polyu.edu.hk, csmzhang@comp.polyu.edu.hk

Abstract—Collaborative edge computing (CEC) has emerged as a promising paradigm, enabling edge nodes to collaborate and execute microservices from end devices. Microservice offloading, a fundamentally important problem, decides when and where microservices are executed upon the arrival of services. However, the dynamic nature of the real-world CEC environment often leads to inefficient microservice offloading strategies, resulting in underutilized resources and network congestion. To address this challenge, we formulate an online joint microservice offloading and bandwidth allocation problem, JMOBA, to minimize the average completion time of services. In this paper, we introduce a novel microservice offloading algorithm, DTDRLMO, which leverages deep reinforcement learning (DRL) and digital twin technology. Specifically, we employ digital twin techniques to predict and adapt to changing edge node loads and network conditions of CEC in real-time. Furthermore, this approach enables the generation of an efficient offloading plan, selecting the most suitable edge node for each microservice. Simulation results on real-world and synthetic datasets demonstrate that DTDRLMO outperforms heuristic and learning-based methods in average service completion time.

Index Terms—Microservice offloading, deep reinforcement learning, digital twin, collaborative edge computing

I. INTRODUCTION

The rapid proliferation of the Internet of Things (IoT) devices, with enhanced computation and communication capacities, has given rise to innovative applications like autonomous vehicles, smart healthcare, and metaverse [1], [2]. These complex applications involve numerous dependent components and vast amounts of data, necessitating low-latency services that centralized cloud computing struggles to provide. Collaborative edge computing (CEC) has emerged as a promising solution to support these applications [3]. The trend of replacing monolithic applications with loosely-coupled, lightweight microservices is on the rise, given their elasticity and flexibility compared to traditional virtual machines [4]. Microservices offer distinct benefits over their monolithic predecessors, including friendly cross-team development and ease of deployment.

Nevertheless, edge nodes usually operate with limited resources. In critical situations, they may lack the necessary resources to process and transfer heterogeneous application microservices to edge nodes in real-time [5]. Factors such as high network load, shared radio access network, and an increasing need for network bandwidth, can impede data transmission in edge computing, further straining the already

limited resources of edge nodes. Moreover, the resource demands of edge nodes fluctuate over time, indicating a characteristic known as resource-agnostic property. Understanding this dynamic is vital for ensuring equitable resource allocation among competing edge nodes.

Microservices offloading, a key enabling technology in CEC, involves transferring resource-intensive computational microservices from end devices to resource-rich edge nodes. During microservice offloading, the optimal edge node for offloading each microservice is determined based on factors like microservice information, dynamic edge node loads, and network conditions. Data transfer time significantly impacts service completion time, due to microservices require access to data distributed across the network. However, existing works have not adequately addressed joint task offloading and network flow scheduling, resulting in flow congestion and suboptimal performance [6], [7], [8]. Some existing works are either limited to data centers or cloud edge computing or offline, rendering them unsuitable for real-time offloading decisions in rapidly changing networks [9], [10], [3].

The emerging digital twin technique, which captures system dynamics and provides future insights by integrating data analytics and machine learning methods [11], holds the potential for addressing dynamic resource availability issues. Despite its success in areas like industrial maintenance [12], aeronautics and astronautics [13], and task offloading in edge computing [14], the integration of digital twin information in microservice offloading and resource allocation remains relatively unexplored.

To fill this gap, this paper focuses on digital twin-assisted joint microservice offloading and bandwidth allocation in CEC, where the digital twin technique is utilized to predict resource availability. We formulate an online joint microservice offloading and bandwidth allocation (JMOBA) problem to minimize the average service completion time. We address three main challenges. First, how to make optimal decisions considering task offloading and network flow scheduling jointly? Second, how do we effectively deal with dynamic resource availability issues of the online CEC environment? Finally, how can digital twin techniques be employed to simulate and adapt to changing edge node loads and network conditions in real-time? To overcome these challenges, we propose a microservice offloading algorithm that integrates digital twin technology with deep reinforcement learning.

This algorithm allows us to predict and adapt to real-time changes in loads on edge node and network conditions, adjust offloading decisions based on resource availability, and use a trained DRL model to schedule tasks efficiently.

The novelty of this work lies in leveraging digital twin techniques to address dynamic resource availability issues in online joint microservice offloading and bandwidth allocation within CEC networks. By employing digital twins, our method can capture the constantly changing resource requirements and network conditions in real-time. This method enables us to make more informed decisions regarding dynamic resource availability, ultimately leading to more efficient microservice offloading and execution. The primary contributions of this paper are as follows:

- Exploring resource availability dynamics in online joint microservice offloading and bandwidth allocation in CEC. We investigate a novel optimization problem for efficient microservice offloading and bandwidth allocation: JMOBA, which is NP-hard.
- Building an optimization framework for digital twin-assisted efficient microservice offloading and bandwidth allocation in CEC. The objective is to minimize the average completion time of arrived services.
- Developing an online microservice offloading algorithm, called DTDRLMO, that utilizes digital twin and deep reinforcement learning techniques. To address dynamic resource availability issues, we design a novel state transition model that leverages digital twin techniques to simulate the state of the CEC environment.
- Evaluating the proposed algorithm DTDRLMO using both real-world and synthetic datasets through simulations. The results demonstrate that DTDRLMO outperforms state-of-the-art heuristic and learning-based algorithms.

The paper is structured as follows. Section II surveys relevant studies. Our system, network models, and problem formulation are in Section III. Section IV presents our DTDRLMO approach. The efficacy of our solution is assessed in Section V, and Section VI concludes with our findings and future research work.

II. RELATED WORK

A. Task Offloading in Edge Computing

Joint task offloading and network scheduling in CEC networks has been addressed in several recent studies. Heuristic online methods such as the approach by Meng et al. [15] and OnDisc by Han et al. [16] have been proposed, but their performance is limited due to the dynamic changes in task arrival and network conditions. Deep reinforcement learning (DRL) has been increasingly used to tackle the dynamic issues in task offloading problems in edge computing, with various DQN-based approaches and policy gradient DRL methods [17], [18], [6], [19], [7], [8]. However, these approaches do not jointly consider network flow scheduling, which can lead to network congestion and suboptimal completion time.

B. Microservice Offloading in Edge Computing

Several recent studies investigated microservice offloading in edge networks. Samanta et al. [20] proposed a dynamic scheduling scheme for MEC, optimizing end-user microservices execution while maximizing energy efficiency and QoS. However, they overlooked the overheads of container-based microservices at the edge. Gu et al. [21] addressed this by proposing a layer-aware microservice placement and request scheduling scheme, exploiting the layered structure of container images for efficiency. However, the dynamic nature of edge computing environments remains a challenge.

Only a few studies have addressed the dynamics of edge computing using deep reinforcement learning and digital twin methods. Wang et al. [22] and Chen et al. [23] utilized DRL to manage dynamic edge environments, but their reliance on prior information and human instruction is a limitation. DT-assisted methods have also been proposed [24], [14], [25]. Dai et al. [24] used an asynchronous actor-critic algorithm for stochastic computation offloading and resource allocation. Liu et al. [14] employed a combination of the decision tree algorithm and double deep-Q-learning for task offloading to cooperative mobile-edge servers. Furthering DT-assisted methods, Li et al. [25] studied DT-assisted, Service Function Chain-enabled reliable service provisioning in Mobile Edge Computing networks.

Contrary to the aforementioned studies, our work explores online joint microservice offloading and bandwidth allocation in CEC with online service arrivals. We further consider the dynamic resource availability issue by leveraging the digital twin technique to provide an accurate prediction of the state of the CEC environment for each microservice.

III. SYSTEM MODEL AND PROBLEM FORMULATION

A. System Model

Fig. 1 presents the architecture of a digital twin-based CEC system. The system comprises the end device layer, physical CEC layer, and digital twin layer.

- In the end device layer, the devices submit the application task to edge nodes for processing.
- In the physical layer, there are various edge nodes connected via a multi-hop path. The intelligence is dispersed throughout the network by sharing computational resources and data between the edge nodes for collaboratively executing application tasks.
- In the digital twin layer, each edge node and link in the CEC network is mirrored into a digital twin by monitoring and predicting the network status. It also makes offloading decisions about when and where to execute the application tasks in the CEC resource pool.

The CEC system operates in a time-slot manner. The timeline is divided into K time frames (e.g., second), with each frame composed of time slots (e.g., millisecond) of length T with $T \geq 1$. Given a sequence of time slots $\{0, T, \dots, K \cdot T\}$, we define $t = k \cdot T$ ($k = 0, 1, \dots$) as the beginning of each time frame $[t, t + T]$.

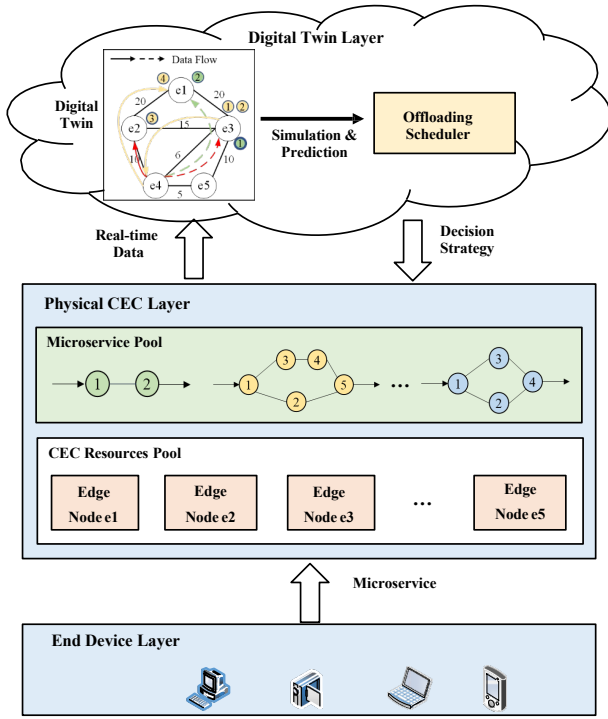


Fig. 1. The digital twin-based microservice offloading framework, consisting of the end device layer, physical CEC layer, and digital twin layer.

B. Microservice Model

Assume there are M services submitted to the CEC system. Each service is developed following the microservice principles, which consist of a set of dependent microservices. We use Directed Acyclic Graph (DAG) to model a service. The service set is represented by $CT = \{CT_i | 1 \leq i \leq M\}$. Each service i is modeled as a DAG, indicated by $CT_i = (T_i, P_i)$. T_i is the set of dependent microservices in service i , $T_i = \{j | 1 \leq j \leq N_i\}$, where N_i is the number of microservices in service i . P_i is the set of dependencies among the microservices in service i . Each microservice has dependent data with its processor microservices. Each service i is assumed to be generated at an end device and will be offloaded to edge nodes.

Different microservices have different computation and communication resource requirements. The size of the input data packet for microservice j of service i is $D_{i,j}$. Each microservice j is associated with the computation load $C_{i,j}$. The set of predecessors for microservice j is represented by $Pd_{i,j}$.

C. Digital twin-assisted Collaborative Scheduling

The network model is represented as a connected graph $G = (V, E)$. Inside, $V = \{v | 1 \leq v \leq \alpha\}$ is the set of edge nodes, where α is the total number of edge nodes. E is the set of links connecting different edge nodes, $E = \{e_{j,k} | j, k \in V\}$.

The resource consumption of an edge node n can be expressed by a tuple $\langle R_{i,j}, AR_n \rangle$, where $R_{i,j}$ is the resource amount required by microservice j in service i and $AR_{n,t}$ is the amount of available resources of edge node n . We adopt

digital twin techniques to dynamically provide accurate and personalized availability prediction for each edge node.

D. Problem Formulation

We first consider a snapshot of arrived microservices in the CEC network. Given a CEC network $G = (V, E)$, digital twins run in the remote cloud to provide a reliable prediction of microservices and edge nodes in real-time. The online joint microservice offloading and bandwidth allocation problem is to minimize the average completion time of all services submitted to the CEC resource pool in the time snapshot, while meeting their availability requirements, subject to the computing resource on each edge node in G . More specifically, we will decide which edge node to allocate a microservice, when to execute it, and the bandwidth allocated to intermediate data flow among microservices, with digital twin, to dynamically predict the resource availability of each edge node.

The completion time of a service consists of two parts: the computation cost, which indicates the completion time of a microservice on allocated edge nodes, and the flow transmission cost, which indicates the time spent to receive the dependent input data from processor microservices. The microservice generated at an end device will be offloaded to the local edge node, which is nearest to the end device after its release. So we can ignore the upload time of the task. According to [26], we can also ignore the effect of the result return time on microservice uploading because the output after microservice processing is much smaller than the input.

Computation Cost. The computation time of edge node n for microservice i is composed of four parts: the time when the service is uploaded to the edge nodes, the waiting time of the microservice at an edge node, the calculation time of the microservice at the edge nodes, and the download time when the result is returned to the user. According to [26], we can also ignore the effect of the result return time on microservice uploading because the output after microservice processing is much smaller than the input.

Let $\delta_{i,j,n,t} \in \{0,1\}$ be a binary decision variable to indicate whether the microservice j in service i is allocated to edge node n at time t . Let $W_{i,j}$ represent the waiting time of microservice j in service i . $PS_{n,t}^t$ indicates the average processing speed of an edge node n at time t . Then, the computation time $PT_{n,i,j}$ of microservice j of service i at edge node n is defined as:

$$PT_{n,i,j} = (C_{i,j}/PS_{n,t}^t + W_{i,j}) * \delta_{i,j,n,t} \quad (1)$$

Flow Transmission Cost. A microservice can not start until it receives all dependent input data. Let $B_{f,i,j}^t$ be a continuous decision variable to indicate the bandwidth allocated to the flow f for microservice j in service i at time t . The routing path of flow f is denoted as P_f . We assume it is the shortest path between the source node and the destination node. Let BW_e^t indicate the remaining bandwidth of edge link e at time t . The scale of $B_{f,i,j}^t$ is:

$$0 < B_{f,i,j}^t < \min_{e \in P_f} BW_e^t \quad (2)$$

ϵ_f is the waiting time of the flow f to be transmitted. Then, the flow transmission time $BT_{f,i,j}$ of flow f for microservice j is calculated as:

$$BT_{f,i,j} = D_{i,j}/B_{f,i,j}^t + \epsilon_f \quad (3)$$

With the formulation of the computation and flow transmission time, the finish time $FT_{i,j}$ of microservice j in service i can be calculated as:

$$FT_{i,j} = PT_{n,i,j} + BT_{f,i,j} + \max_{k \in Pd_{i,j}} (FT_k) \quad (4)$$

where $Pd_{i,j}$ represents the set of predecessors of microservice j . The finish time TT_i of service i is calculated as:

$$TT_i = \max_{j \in T_i} FT_{i,j} \quad (5)$$

Objective. Our objective is to minimize the average completion time of all services submitted at timeslot t . The joint dependent task offloading and bandwidth allocation problem can be formulated as follows:

$$\min_{\delta_{i,j,n,t}, B_{f,i,j}^t, W_{i,j}, \epsilon_f} \sum_{i=1}^M TT_i/M \quad (6)$$

$$\sum_{n=1}^{\alpha} \delta_{i,j,n,t} = 1, \quad \forall i, j \quad (7)$$

$$\sum_{i=1}^M \sum_{j=1}^{N_i} R_{i,j} * \delta_{i,j,n,t} \leq AR_{n,t} \quad (8)$$

$$\sum_{i=1}^M \sum_{j=1}^{N_i} B_{f,i,j}^t \leq \min_{e \in P_f} BW_e^t, \quad e \in P_f \quad (9)$$

where Eq. (7) ensures that a microservice can only be allocated to an edge node. Eq. (8) and Eq. (9) show the resource availability constraints. Eq. (8) indicates that the resource consumption of all microservices allocated to node n shall not exceed the available resources of node n at time t . Eq. (9) constraints that the allocated bandwidth of a link k shall not exceed the available bandwidth capacity of the link.

Theorem 1: The online joint microservice offloading and bandwidth allocation problem in $G = (V, E)$ is NP-hard.

Proof: The online joint microservice offloading and bandwidth allocation problem in $G = (V, E)$ is NP-hard. We will prove that the one-time-slot joint microservice offloading and bandwidth allocation problem is NP-hard by reducing it from the well-known NP-hard problem, the generalized assignment problem (GAP) [27].

The GAP involves packing N items into M bins, where each bin m has a capacity $C(m)$ and packing item n into bin m incurs a cost $cost(n, m)$ and has a size $size(n)$. The

objective is to minimize the total cost of packing all items into bins, subject to the bin capacities.

We consider a special case where the network topology $G = (V, E)$ is fully connected, and only microservice offloading is considered. Let CT be the set of microservices. Each edge node $v \in V$ represents a bin associated with a processing capacity PS_v , and each microservice j in service i represents an item associated with a size $C_{i,j}$, i.e., the computing resource consumption of microservice j . The cost of placing item i into bin v is $cost(i, v)$, i.e., the cost of offloading a microservice j on edge node v . We can see that this special problem is equivalent to the minimum-cost GAP. Therefore, the special problem is also NP-hard [28]. Thus, the online joint microservice offloading and bandwidth allocation problem in $G = (V, E)$ is also NP-hard.

IV. DIGITAL TWIN-ASSISTED DEEP REINFORCEMENT LEARNING

In this section, we model the problem as a Markov decision process (MDP) and propose a novel digital-twin-based deep reinforcement learning approach.

A. Markov Decision Process

The Markov Decision Process (MDP) is used to model the scheduler in our CEC system. In CEC, the scheduler's action consists of designating an offloading device and allocating bandwidth for each arriving task, thereby leading to a state transition in the MDP. The state transition can be represented as $Pr[S_{t+1} = s' | S_t = s, A_t = a]$. S_t denotes the current state, A_t is the action taken by the scheduler, and S_{t+1} represents the next state. The agent is the scheduler itself, which is responsible for determining the action that leads to the most favorable state transition. The ultimate goal of the agent is to maximize the cumulative reward over time, thereby resolves the problem of efficient task offloading in the edge network.

B. State Space

The state space is defined by the combination of the information of services CT and the network information NI . For a given microservice, the state of the CEC system at time t is contingent on the scheduling outcomes of preceding microservices and network flows.

$$S = \{s_t | s_t = (CT, NI_t)\} \quad (10)$$

$$CT = \{CT_i | CT_i = (T_i, P_i)\} \quad (11)$$

$$NI_t = \{BW_1^t, \dots, BW_p^t, AR_1^t, \dots, AR_\alpha^t\} \quad (12)$$

Specifically, CT includes the information of service DAGs, consisting of the service CT_i , a vector T_i of dependent microservices within CT_i , and the dependencies P_i among these microservices. NI_t represents the status of the CEC network environment, including the bandwidth conditions of the links and the resource conditions of edge nodes. p is the number of edge links. The remaining bandwidth of edge link

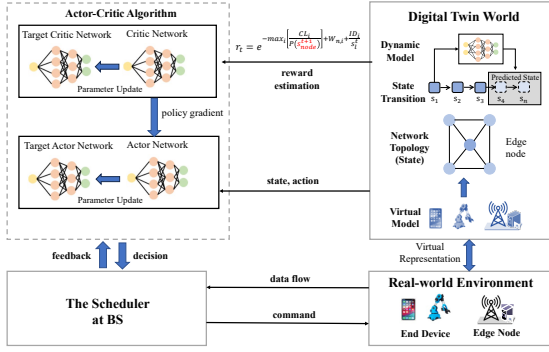


Fig. 2. The diagram of digital twin-assisted deep deterministic policy gradient algorithm.

j at time t is represented by BW_j^t . α is the number of edge nodes. The remaining computation resource (e.g., cpu and memory) of edge nodes k at time t is represented by AR_k^t .

C. Action Space

The action space comprises the set of potential decisions the agent can make in response to a given state. For our problem, the action space includes offloading decisions for the current microservice and the bandwidth allocation for the corresponding network flow.

$$a_t = \{a_{i,t} | a_{i,t} = \{q_{i,t}, \eta_{i,t}, \epsilon_{i,t}\}\} \quad (13)$$

Each action $a_{i,t}$ includes three components: the offloading decision $q_{i,t}$ for each microservice within the current service, the bandwidth $\eta_{i,t}$ allocated to each network flow, and the waiting time $\epsilon_{i,t}$ for each flow.

D. Reward Function

The reward function quantifies the immediate benefit the agent receives after executing an action. In the context of our digital twin-assisted deep deterministic policy gradient (DDPG) offloading algorithm, the reward is derived from the time consumption for completing microservices at the edge nodes. Specifically, the reward r_t at time slot t is defined as the negative weighted sum of microservice completion time. This design ensures that the agent is incentivized to minimize the microservice completion time.

$$r_t = -TT_t \quad (14)$$

In this equation, TT_t denotes the weighted sum of the completion time of microservices generated at time slot t . Each action taken by the agent will result in a new state and associated reward, contributing to the optimization of the offloading process in the digital twin edge network.

E. Digital Twin-assisted Deep Reinforcement Learning

Overall Training Framework. Fig. 2 shows our overall training framework, which includes three key learning components: the *dynamic model*, *actor network*, and *critic network*. The dynamic model is aimed at learning the state transition of

Algorithm 1 Digital Twin Modelling Algorithm

Input: Current state s , action a , reward r , next state s_+ ;

Require: Number of epochs as ep , batch size as bs , and learning rate as lr .

- 1: Initializes the transaction model t_model and the reward model r_model .
- 2: Define the transaction dataset using the input bs and ba , and target datasets bs_+ .
- 3: Define the reward dataset using the input bs_+ and target datasets r .
- 4: Train the transaction model t_model and the reward model r_model using the defined datasets, epochs, batch size, and learning rate.
- 5: The forward propagation $h = f(w_i * X + b_i)$ through the autoencoder is simply applying the activation function (e.g., ReLU, sigmoid, etc.) to the dot product of the input and the weights of the model.
- 6: This process is repeated for a specified number of epochs until the model converges, i.e., the loss doesn't decrease significantly with further training.

the environment, while the actor network and critic network are trained to learn the optimal policy. After learning the dynamic model, we use it to predict the next state and leverage this information to estimate the reward.

State Transition Model (Environment Dynamic Model).

We use a deep neural network parameterized by ϕ_s to approximate the state transition, which maps the state-action tuple to the next state:

$$\forall s_t \in \mathcal{S}, a_t \in \mathcal{A}, \quad \hat{s}_{t+1} = f_{\phi_s}(s_t, a_t) \quad (15)$$

The model can be optimized through supervised learning by regression using the mean-squared error between the predicted next state and the actual one.

$$\min_{\phi_s} \frac{1}{|\mathcal{D}|} \sum_{(s_t, s_{t+1}, a_t, r_t) \in \mathcal{D}} \|s_{t+1} - (s_t + f_s(s_t, a_t))\|^2 \quad (16)$$

Where \mathcal{D} is the replay buffer and f_s maps the state-action tuple to the difference between the current state and the next state. During each iteration of training, we alternate between model fitting and agent updates using the data collected from the environment interactions.

Estimating the reward using the state model. Inspired by the model-based reinforcement learning, we use the learned state model to predict the next state so as to estimate the remaining bandwidth and resources. Therefore, the reward function can be further designed as follows:

$$r_t = - \max_{i \in CT} \max_{j \in CT_j} \left[\frac{C_{i,j}}{P(s_{node}^{t+1})} + W_{i,j} + \frac{D_{i,j}}{s_{link}^t} \right] \quad (17)$$

Where the s_{node}^{t+1} is the next state predicted from the state transition model.

Deep Deterministic Policy Gradient. In accordance with Fig. 2, the DDPG model consists of an actor network, a critic network, and an experience pool. The actor network uses a policy-based approach, the critic network is value-based, and the experience pool retains each system state record (s_t, a_t, r_t, s_{t+1}) . The deterministic strategy μ is formulated as:

$$\mu : s_t \rightarrow a_t \quad (18)$$

By approximating the strategy with a continuous function parameterized by θ , the deterministic strategy becomes:

$$\mu_\theta(a_t|s_t) = P(a_t|s_t; \theta) \quad (19)$$

The agent's goal is to learn the optimal policy to maximize the progressive reward:

$$J(\theta) = \mathbb{E}_{s_t \sim \rho^\mu, a_t \sim \mu} [r] \quad (20)$$

Here, ρ^μ represents the state transition probability given the action distribution μ and r is the progressive expected rewards.

To address high variance in the policy gradient estimator, the actor-critic network approximates the expected returns with a function $Q_\omega(s, a)$, replacing the original return term. The gradient of the objective J is:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_t \sim \rho^\mu} [\nabla_\theta \mu_\theta(a | s_t) \nabla_a Q'(s_t, a)|_{a=\mu_\theta(s_t)}] \quad (21)$$

A replay buffer stores previous system state transition experiences, which are randomly sampled during training to enhance learning performance.

To train the critic network parameters, a batch of experiences is randomly drawn from the replay buffer. The training objective is to minimize the difference between $Q(s_t, a_t; w)$ and $Q'(s_t, a_t; w')$ using the loss function L :

$$L = 1/N \sum_i (y_i - Q(s_t, a_t; w))^2 \quad (22)$$

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'_{\theta'}(s_{i+1}); w') \quad (23)$$

where N is the number of samples drawn from the experience pool, y_i is the temporal difference target for Q-learning, and w and w' represent parameters of the critic network.

As outlined in Algorithm 2, the scheduler begins by initializing the replay buffer, critic network, and actor network with weight w and θ (lines 1-3). After obtaining the environmental state s_t , the agent chooses action a_t based on the current policy and exploration noise (lines 5-8). Following action execution and interaction with the environment in the processing pool, the agent receives reward r_t , observes the next state s_{t+1} of the environment, and stores the state $\langle s_t, a_t, r_t, s_{t+1} \rangle$ into the replay buffer (lines 9-10). Subsequently, the agent randomly samples a mini-batch experience from the replay buffer and updates the critic network by minimizing the loss (lines 11-13). After each interaction episode with the environment, the agent updates the actor and critic networks using the sampled policy gradient and target network parameters (lines 14-19).

Algorithm 2 DTDLMO Algorithm

Input: Training episode length Y , training sample length T ; DAG microservice number β , edge node number α , edge link number p ;

- 1: Randomly initialize actor network A_θ and critic network c_w with weights θ and w , respectively
 - 2: Initialize target network A_θ and c_w with weights w' and θ' , respectively
 - 3: $w' \leftarrow w, \theta' \leftarrow \theta$
 - 4: Initialize replay memory B
 - 5: **for** episode $\leftarrow 0$ **to** Y **do**
 - 6: Get environment state s_t
 - 7: Initialize a random process N for action exploration
 - 8: **for** $t \leftarrow 1$ **to** T **do**
 - 9: Execute each action and obtain the estimated reward r_t and estimated new state s_{t+1} in digital twin work.
 - 10: Select action $a_t = (q_t, \eta_t, \epsilon_t)$ with the maximum reward according to the current state, future state, current policy and exploration noise
 - 11: In collaborative edge computing, execute action a_t and obtain the reward r_t and new state s_{t+1}
 - 12: Store state transition (s_t, a_t, r_t, s_{t+1}) in pool B
 - 13: Randomly select a minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from B .
 - 14: $y_i \leftarrow r_i + \gamma Q'(s_{i+1}, \mu'_{\theta'}(s_{i+1}); w')$
 - 15: Update the critic network by minimizing the loss $L = 1/N \sum_i (y_i - Q(s_i, a_i; w))^2$
 - 16: Update the actor network as follows: $\nabla_{\theta_\mu} J \approx 1/N \sum_i Q(s_t, a_t; w)|_{s_t=s_i, a_t=\mu_\theta(s_i)} \nabla_{\theta_\mu} \mu_\theta(s_t)|_{s_t=s_i}$
 - 17: $\theta'_{\mu'} \leftarrow t\theta_\mu + (1-t)\theta'_{\mu'}$ \triangleright Update one of the target networks
 - 18: $w'_{Q'} \leftarrow tw_Q + (1-t)w'_{Q'}$ \triangleright Update the other target network
 - 19: **end for**
 - 20: **end for**
-

Difference from other existing RL approaches. Most of the existing RL approaches are model-free based approach, which does not explicitly consider the future state transition. In contrast to the existing model-based RL algorithms, our method uses the state transition model to reshape the reward function, while others only use it to generate experience data.

V. PERFORMANCE EVALUATION

A. Simulation Settings

Network Model. Our network model utilizes a stochastic topology generator [29] to construct a network topology comprising 8 edge nodes. The weight (or capacity) of the connection between any two edge nodes signifies the bandwidth of the link. Owing to the heterogeneity in devices, the computational capacity of edge nodes is chosen from a Gaussian distribution with an average of 40 Mcps (megacycles per second) and a standard deviation of 80%. Similarly, the processing power of end devices follows a normal distribution with a mean of 10 Mcps and a standard deviation of 20%. The bandwidth

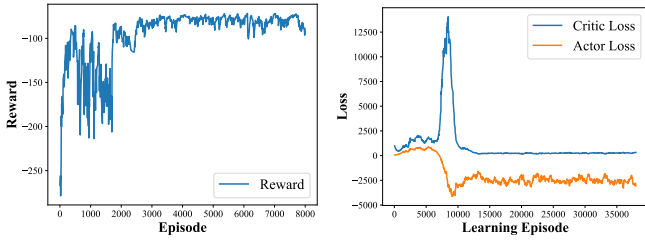


Fig. 3. The values of reward, actor loss, and critic loss in the training process of the DTDRLMO.

of each edge link connection is also selected from a normal distribution, with an average of 10 Mbps (megabits per second) and a standard deviation of 80%.

Synthetic Dataset. We use a stochastic Directed Acyclic Graph (DAG) generator [10] to create DAG microservices. Each DAG microservice has properties like microservice id, service id, data size, computational load, release time, and source device. The number of nodes and layers in the DAGs is selected from a Gaussian distribution within [1, 50]. The number of edges or dependencies between two layers in the DAG is chosen from a uniform distribution. The microservices are randomly generated at edge nodes. The data size for each microservice is randomly generated from a normal distribution with a mean of 500 Mbit and a standard deviation of 80%. The computational load for each microservice is randomly chosen from a Gaussian distribution with a mean of 500 kilo clock cycles and a standard deviation of 60%. The release time for microservices is randomly chosen using a Poisson distribution. The synthetic dataset is split into training and testing subsets.

Real-world Dataset. Alibaba cluster trace includes DAG details of real-world workload traces [30]. Some modifications are made due to incomplete information in the Alibaba cluster trace to generate DAGs. Computational loads of microservices are obtained by appropriately scaling the value of the difference between "start time" and "end time", multiplied by "plan cpu". The data size of each microservice is randomly chosen from a Gaussian distribution with a mean of 500 Mbit and a standard deviation of 80%. The release time for each microservice is randomly selected using a Poisson distribution. Microservices are randomly spawned at edge nodes. The real-world dataset is split into training and testing subsets.

Metric. In our simulations, we assess the efficacy of the suggested algorithms using the primary metric: Average Completion Time (ACT). ACT is calculated as the total completion time of all services divided by the total number of services. This measure reflects the average duration required for service completion, offering insights into the efficiency of the microservice offloading procedure.

Configuration & Baselines. DTDRLMO is implemented on our platform equipped with 32 cores and 2 GPU cards. As presented in Fig. 3, the training and convergence results of the DTDRLMO include the reward, actor loss, and critic loss. Our experimental results show that the training converges after approximately 8000 episodes. The critic loss approximates

to zero after 15000 learning episodes, where each learning episode corresponds to an iteration of the learning function that updates the actor and critic networks. The training parameters are as follows: the replay memory size is set to 10000, with a mini-batch size of 64, indicating the number of memories used for each training step. The learning rates of actor and critic networks are set at 0.001 and 0.002, respectively. The reward discount is set at 0.9, and we use a soft replacement parameter of 0.01. We implement four comparative algorithms for microservice offloading to benchmark performance. The baselines are as follows:

- **Local Execution (LE):** In this strategy, microservices are run on end devices, negating the need for network flow scheduling.
- **Greedy:** This strategy computes the estimated completion time for each device and prefers the device with the minimal time for offloading. Network flow scheduling follows the First Come First Served (FCFS) strategy.
- **DRL without DT:** Wang et al. [8] introduced a DRL-based computation offloading algorithm based on DDPG, albeit without adopting the digital twin technology.

B. Performance in the Real-world Dataset

Effect of Change in the Number of Services: Fig. 4(a) shows the effect of changing the number of services from 10 to 40 on ACT. The ACT difference between LE and DTDRLMO increases from 28.1% at 10 services to 83.4% at 40 services. This increase in the gap is observed because LE executes the microservices locally at the devices at which it is generated, which leads to a larger waiting time as the number of services is increased. When the number of services increases to 40, DTDRLMO achieves an average delay of 59.9% lower than those of DRL without DT and Greedy. This is because DTDRLMO, unlike Greedy and DRL without DT, incorporates digital twin technology, which allows it to better adapt to changes in the network environment and make more efficient offloading decisions.

Effect of Change in the Number of Microservices: Fig. 4(b) shows the effect of changing the maximum number of microservices in a service DAG from 25 to 100 within each service DAG. LE executes the task locally at the devices at which it is generated, which leads to a larger waiting time as the number of microservices is increased. So there is a significant difference in ACT between DTDRLMO and LE from about 18.1% at 25 maximum microservices to 65.6% at 100 maximum microservices. The difference in ACT between DTDRLMO and DRL without DT is 24.2% at 100 maximum microservices. This is because DRL without DT lacks the ability to adapt to dynamic changes in the network environment due to the absence of digital twin technology. DTDRLMO is around 26.3% lower than Greedy in ACT. This is because the Greedy strategy only considers immediate benefits without taking into account the overall network condition.

Effect of Change in the Average Bandwidth of Edge Links: In Fig. 4(c), the average bandwidth of edge links changes from 2 Mbps to 5 Mbps. The ACT of LE is constant

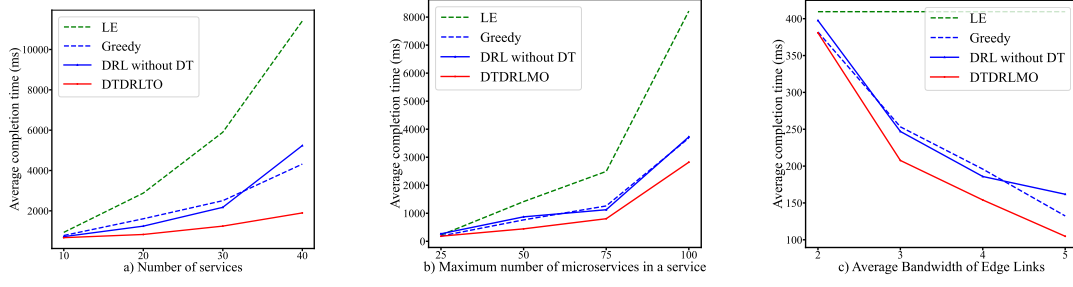


Fig. 4. Simulation Results for Real-world Dataset

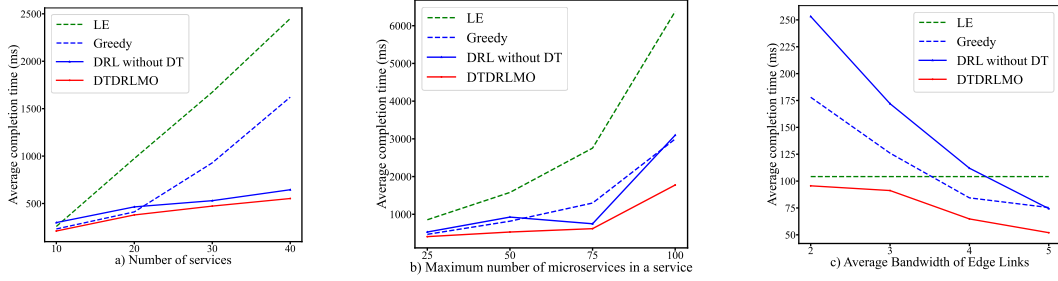


Fig. 5. Simulation Results for Synthetic Dataset

due to the absence of network flows. It is expected that DTDRLMO performs significantly better than baselines in low bandwidth condition. When the average bandwidth decreases to 2 Mbps, DTDRLMO achieves an ACT that is around 18.0% and 15.9% lower than that of DRL without DT and Greedy, respectively. This is because DRL without DT, lacking the digital twin technology, cannot adapt to the low bandwidth condition and thus makes inefficient offloading decisions, leading to a higher ACT. On the other hand, Greedy is not able to make full use of computation and bandwidth resources when offloading microservices to target nodes.

C. Performance in the Synthetic DataSet

Effect of Change in the Number of Services: Fig. 5(a) shows an increase in ACT for all algorithms of changing the number of services from 10 to 40. This is due to both increases in waiting time at the devices to execute the services and the total number of network flows. DTDRLMO is significantly better than LE. Because LE executes the service at the local device where the task is generated, resulting in an increase in waiting time for microservice execution. DTDRLMO's superiority is further evidenced as DRL without DT is around 22.4% higher than the DTDRLMO at microservice 20. This is because DRL without DT lacks the ability to adapt to dynamic changes in the network environment, leading to inefficient offloading decisions and consequently higher ACT. Furthermore, DTDRLMO averaged about 33.0% lower than Greedy in ACT when the number of services ranged from 10 to 40. This is because Greedy often leads to suboptimal offloading decisions without taking into account the overall network condition.

Effect of Change in the Number of Microservices: In Fig. 4(b), the maximum number of microservices in a service

DAG changes from 25 to 100 within each service DAG. DTDRLMO is significantly better than LE. This is observed because LE does not offload tasks, and the computational load on the edge nodes increases with the number of microservices, resulting in longer waiting times and hence a higher ACT. DTDRLMO, on the other hand, obtains around 35.1% higher performance than Greedy when the number of microservices increases to 50. This is because DTDRLMO, unlike the Greedy strategy, is capable of making more informed offloading decisions by considering the overall network condition rather than just the immediate completion time of each edge node. When the number of microservices increases to 100, DTDRLMO achieves an average delay of 41.7% lower than those of DRL without DT and Greedy. Because DRL without DT and the Greedy strategy, due to their lack of adaptability and short-sighted nature in dynamic environment respectively, are unable to efficiently manage the increased computational load, leading to a higher average delay.

Effect of Change in the Average Bandwidth of Edge Links: In Fig. 5(c), the average bandwidth of edge links changes from 2 Mbps to 5 Mbps. The ACT of LE is constant due to all microservices run on local edge nodes where microservices are generated, and network flow scheduling is ignored. When the average bandwidth increases to 5 Mbps, DTDRLMO achieves a delay of 50.1% lower than LE. This is because DTDRLMO, unlike LE, leverages the increased bandwidth to offload microservices more efficiently. The ACT of Greedy and DRL without DT is significantly higher than LE at low bandwidth of 2 Mbps. This is because both Greedy and DRL without DT rely on network bandwidth for offloading tasks, and their performance deteriorates when the bandwidth is limited. There is a decrease in ACT performance difference

between DTDRLMO and DRL without DT from about 66.2% at the average bandwidth of 2 Mbps to 31.0% at the average bandwidth of 5 Mbps. This is because DRL without DT, despite its limitations, can still benefit from the increased bandwidth to some extent. DTDRLMO is around 87.5% lower than Greedy in terms of ACT at the average bandwidth of 2 Mbps and 31.0% less at the average bandwidth of 5 Mbps. This is because DTDRLMO, with its ability to learn and adapt, can make more efficient use of the available bandwidth compared to the short-sighted Greedy algorithm.

VI. CONCLUSION

In this paper, we formulate an online joint microservice offloading and bandwidth allocation problem, which addresses the dynamic resource availability issues of microservice offloading in CEC. We propose a novel algorithm, DTDRLMO, that combines DRL and digital twin techniques, enabling real-time adaptation to changing load on edge nodes and network conditions. Simulation results on real-world and synthetic datasets highlight that DTDRLMO outperforms state-of-the-art heuristic and learning-based methods. Our future work will implement and test our algorithm in a real-world edge computing system to further validate its effectiveness.

VII. ACKNOWLEDGEMENT

This work was supported by the Research Institute for Artificial Intelligence of Things, The Hong Kong Polytechnic University, the Hong Kong (HK) Research Grant Council (RGC) General Research Fund with project code PolyU 15220922 and the HK RGC Research Impact Fund with project code R5060-19.

REFERENCES

- [1] L. U. Khan, I. Yaqoob, N. H. Tran, S. A. Kazmi, T. N. Dang, and C. S. Hong, "Edge-computing-enabled smart cities: A comprehensive survey," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 10200–10232, 2020.
- [2] M. Xu, W. C. Ng, W. Y. B. Lim, J. Kang, Z. Xiong, D. Niyato, Q. Yang, X. S. Shen, and C. Miao, "A full dive into realizing the edge-enabled metaverse: Visions, enabling technologies, and challenges," *arXiv preprint arXiv:2203.05471*, 2022.
- [3] Y. Sahni, J. Cao, L. Yang, and Y. Ji, "Multi-hop multi-task partial computation offloading in collaborative edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 5, pp. 1133–1145, 2020.
- [4] M. Zhang, J. Cao, L. Yang, L. Zhang, Y. Sahni, and S. Jiang, "Ents: An edge-native task scheduling system for collaborative edge computing," in *2022 IEEE/ACM 7th Symposium on Edge Computing (SEC)*, pp. 149–161, IEEE, 2022.
- [5] M. Zhang, J. Cao, Y. Sahni, Q. Chen, S. Jiang, and T. Wu, "Eaas: A service-oriented edge computing framework towards distributed intelligence," in *2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pp. 165–175, IEEE, 2022.
- [6] M. Tang and V. W. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Transactions on Mobile Computing*, 2020.
- [7] J. Wang, J. Hu, G. Min, W. Zhan, A. Zomaya, and N. Georgalas, "Dependent task offloading for edge computing based on deep reinforcement learning," *IEEE Transactions on Computers*, 2021.
- [8] Y. Wang, W. Fang, Y. Ding, and N. Xiong, "Computation offloading optimization for uav-assisted mobile edge computing: a deep deterministic policy gradient approach," *Wireless Networks*, vol. 27, no. 4, pp. 2991–3006, 2021.
- [9] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 856–868, 2018.
- [10] Y. Sahni, J. Cao, L. Yang, and Y. Ji, "Multihop offloading of multiple dag tasks in collaborative edge computing," *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 4893–4905, 2020.
- [11] L. U. Khan, W. Saad, D. Niyato, Z. Han, and C. S. Hong, "Digital-twin-enabled 6g: Vision, architectural trends, and future directions," *IEEE Communications Magazine*, vol. 60, no. 1, pp. 74–80, 2022.
- [12] S. Mi, Y. Feng, H. Zheng, Y. Wang, Y. Gao, and J. Tan, "Prediction maintenance integrated decision-making approach supported by digital twin-driven cooperative awareness and interconnection framework," *Journal of Manufacturing Systems*, vol. 58, pp. 329–345, 2021.
- [13] F. Tao, J. Cheng, Q. Qi, M. Zhang, H. Zhang, and F. Sui, "Digital twin-driven product design, manufacturing and service with big data," *The International Journal of Advanced Manufacturing Technology*, vol. 94, pp. 3563–3576, 2018.
- [14] T. Liu, L. Tang, W. Wang, Q. Chen, and X. Zeng, "Digital-twin-assisted task offloading based on edge collaboration in the digital twin edge network," *IEEE Internet of Things Journal*, vol. 9, no. 2, pp. 1427–1444, 2021.
- [15] J. Meng, H. Tan, C. Xu, W. Cao, L. Liu, and B. Li, "Dedas: Online task dispatching and scheduling with bandwidth constraint in edge computing," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 2287–2295, IEEE, 2019.
- [16] Z. Han, H. Tan, X.-Y. Li, S. H.-C. Jiang, Y. Li, and F. C. Lau, "Ondisc: Online latency-sensitive job dispatching and scheduling in heterogeneous edge-clouds," *IEEE/ACM Transactions on Networking*, vol. 27, no. 6, pp. 2472–2485, 2019.
- [17] L. Huang, S. Bi, and Y.-J. A. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Transactions on Mobile Computing*, vol. 19, no. 11, pp. 2581–2593, 2019.
- [18] Y. Zhan, S. Guo, P. Li, and J. Zhang, "A deep reinforcement learning based offloading game in edge computing," *IEEE Transactions on Computers*, vol. 69, no. 6, pp. 883–893, 2020.
- [19] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, "Fast adaptive task offloading in edge computing based on meta reinforcement learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 242–253, 2020.
- [20] A. Samanta and J. Tang, "Dyme: Dynamic microservice scheduling in edge computing enabled iot," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6164–6174, 2020.
- [21] L. Gu, D. Zeng, J. Hu, B. Li, and H. Jin, "Layer aware microservice placement and request scheduling at the edge," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pp. 1–9, IEEE, 2021.
- [22] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. Shen, "Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach," *IEEE Transactions on Mobile Computing*, vol. 20, no. 3, pp. 939–951, 2019.
- [23] L. Chen, Y. Xu, Z. Lu, J. Wu, K. Gai, P. C. Hung, and M. Qiu, "Iot microservice deployment in edge-cloud hybrid environment using reinforcement learning," *IEEE Internet of Things Journal*, vol. 8, no. 16, pp. 12610–12622, 2020.
- [24] Y. Dai, K. Zhang, S. Maharjan, and Y. Zhang, "Deep reinforcement learning for stochastic computation offloading in digital twin networks," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 7, pp. 4968–4977, 2020.
- [25] J. Li, S. Guo, W. Liang, Q. Chen, Z. Xu, W. Xu, and A. Y. Zomaya, "Digital twin-assisted, sfc-enabled service provisioning in mobile edge computing," *IEEE Transactions on Mobile Computing*, 2022.
- [26] K. Cheng, Y. Teng, W. Sun, A. Liu, and X. Wang, "Energy-efficient joint offloading and wireless resource allocation strategy in multi-mec server systems," in *2018 IEEE international conference on communications (ICC)*, pp. 1–6, IEEE, 2018.
- [27] D. G. Cattrysse and L. N. Van Wassenhove, "A survey of algorithms for the generalized assignment problem," *European journal of operational research*, vol. 60, no. 3, pp. 260–272, 1992.
- [28] T. Öncan, "A survey of the generalized assignment problem and its applications," *INFOR: Information Systems and Operational Research*, vol. 45, no. 3, pp. 123–141, 2007.
- [29] C. W. Cesar Ghali, "Random topology generator." [Online]. <https://github.com/cesarghali/topology-generator/>.
- [30] "Alibaba cluster-trace-v2018." [Online]. <https://github.com/alibaba/clusterdata/tree/master/cluster-trace-v2018>.