




# Reducing Offloading Latency for Digital Twin Edge Networks in 6G

Wen Sun , Senior Member, IEEE, Haibin Zhang , Member, IEEE, Rong Wang, Student Member, IEEE, and Yan Zhang , Fellow, IEEE

**Abstract**—6G is envisioned to empower wireless communication and computation through the digitalization and connectivity of everything, by establishing a digital representation of the real network environment. Mobile edge computing (MEC), as one of the key enabling factors, meets unprecedented challenges during mobile offloading due to the extremely complicated and unpredictable network environment in 6G. The existing works on offloading in MEC mainly ignore the effects of user mobility and the unpredictable MEC environment. In this paper, we present a new vision of Digital Twin Edge Networks (DITEN) where digital twins (DTs) of edge servers estimate edge servers' states and DT of the entire MEC system provides training data for offloading decision. A mobile offloading scheme is proposed in DITEN to minimize the offloading latency under the constraints of accumulated consumed service migration cost during user mobility. The Lyapunov optimization method is leveraged to simplify the long-term migration cost constraint to a multi-objective dynamic optimization problem, which is then solved by *Actor-Critic* deep reinforcement learning. Simulations results show that our proposed scheme effectively diminishes the average offloading latency, the offloading failure rate, and the service migration rate, as compared with benchmark schemes, while saving the system cost with DT assistance.

**Index Terms**—Mobile edge computing, deep reinforcement learning, digital twin, digital twin edge network.

## I. INTRODUCTION

THE sixth generation (6G) of telecom cellular networks will integrate emerging technological advances in wireless communications, to offer performance superior to 5G, and satisfy promising services and applications [1]. As one of the key communication infrastructure components, mobile edge computing (MEC) promises to support the stringent performance requirements, new functions, and new intelligent services for

all things big and small in future 6G networks, powered by artificial intelligence (AI) techniques [2], [3]. However, with the heterogeneous deployment of edge servers, the network dynamics become difficult to predict, which leads challenges to design the optimal offloading strategy, especially when a number of mobile IoT devices offload computation tasks to edge servers [4], [5]. Moreover, such intensive deployment of computing and storage servers in megacities at the edge of the network will inevitably cause the network configuration module heavy system cost [6], [7].

Besides enhanced and intelligent computing services, 6G is envisioned to empower edge clouds through the digitalization and connectivity of everything. Digital twin (DT), as an emerging digital technology, brings the virtual depiction or the digital representation of the real world equipment to create a mixed-reality virtual world [8]. Through the combination of MEC and DT, the state of the entire MEC network can be monitored in real time, and perception data can be directly provided for the decision-making module [9]. Compared with the work in 4G and 5G, the DTs of various objects created in the edge cloud constitute an important cornerstone of the digital world in the 6G era, optimizing artificial intelligence and analyzing decisions while making the interaction between the physical and virtual worlds more efficient and intuitive.

The existing works on MEC mainly focus on designing offloading schemes to tradeoff between computation latency and energy consumption of user devices. In the era of 6G, when user mobility is inevitably involved, it is more challenging to make a series of optimal offloading decisions, as the current offloading decision will affect subsequent decisions [10], [11]. In addition, the offloading decision management module should make optimal offloading decision according to the constantly perceiving environmental information around the user. That is the current offloading decision considers not only the state of user's surrounding environment in uploading tasks, but also the time-varying environment and user trajectory in the long run. There have been works on the design of the mobile offloading scheme in MEC. Wang *et al.* [12] proposed a dynamic mobile offloading solution to tradeoff between the service migration cost and the transmission cost caused by users being far away from the current service node. Zhao *et al.* [13] proposed a mobile offloading scheme based on multi-attribute decision-making, which balanced various performance indicators of edge servers to make appropriate mobile offloading decision. However, most

Manuscript received June 4, 2020; revised July 26, 2020; accepted August 16, 2020. Date of publication August 24, 2020; date of current version October 22, 2020. This work was supported in part by the National Natural Science Foundation of China under Grant 61771373, in part by Special Funds for Central Universities Construction of World-Class Universities (Disciplines), and in part by China 111 Project (B16037). The review of this article was coordinated by Prof. Liehuang Zhu. (Corresponding author: Yan zhang)

Wen Sun and Haibin Zhang are with the School of Cybersecurity, Northwestern Polytechnical University of World-Class Universities (Disciplines), and in part by China 111 Project (B16037). The review of this article was coordinated by Prof. Liehuang Zhu. (Corresponding author: Yan zhang)

Rong Wang is with the School of Cyber Engineering, Xidian University, Xi'an Shaanxi 710071, China (e-mail: rongwang\_xd@163.com).

Yan Zhang is with the University of Oslo, Oslo 0316, Norway, and also with the Simula Metropolitan Center for Digital Engineering, Oslo 0167, Norway (e-mail: yanzhang@ieee.org).

Digital Object Identifier 10.1109/TVT.2020.3018817

existing works ignore the edge servers' dynamics or assume the states of edge servers known in advance.

Note that DT can capture the essential functions of edge clouds, while providing an energy-efficient way to analyze and train the decision-making ability of the network management module. In this paper, we present a new vision of Digital Twin Edge Network (DITEN). A mobile offloading scheme is proposed based on deep reinforcement learning (DRL) in DITEN to minimize the offloading latency under the constraint of accumulated consumed service migration cost during user mobility. The main contributions are as follows.

- Firstly, we consider a DITEN scenario, where DTs assist the mobile offloading decision by estimating the states of edge servers and providing DRL agent training data. In addition, considering that a divergence exists between the true value of edge server state and its digital representation, we explore the effect of such divergence on the offloading decision.
- Secondly, we formalize the optimization problem of a series of offloading decisions during user mobility, to minimize the average offloading latency under the constraint of long-term migration cost. The Lyapunov dynamic queue optimization scheme is leveraged to simplify the long-term migration cost constraint to a multi-objective dynamic optimization problem.
- Thirdly, the offloading optimization problem is solved using *Actor – Critic* based DRL framework in an online manner, with the training agent achieved from DITEN. Extensive simulations are conducted to show that our proposed scheme effectively diminishes the offloading latency, task failure rate and the migration rate, while maintaining low system cost.

The remainder of this paper is organized as follows. We present the related work in Section II, the system model and problem formulation in Section III. Then, Section IV proposes the DRL-based online mobile offloading scheme and gives the training framework in DITEN. Section V evaluates the performance of the proposed scheme. Section VI concludes this paper.

## II. RELATED WORKS

Most existing works in MEC focus on offloading decision using game theory or machine learning methods to reduce the computational latency and energy consumption [14], [15]. Ndikumana *et al.* [16] considered a joint cache assignment and task offloading scheme, in which edge servers within a fixed hops collaborate to maximize resource utilization, while addressing insufficient computing power of a single edge server. Rodrigues *et al.* [17] studied server deployment solutions in 6G Internet of Things to deal with the huge number of nodes and servers. The proposed scheme with transmission power control achieves lower latency and higher resource efficiency. In order to reduce the time delay and failure rate of task execution in the green MEC system, Mao *et al.* [18] explored a low-complexity online computing offloading scheme for energy harvesting devices. Dinh *et al.* [19] developed an offloading framework that can minimize execution delay and energy consumption by optimizing

task allocation and computing resources. Considering the status changes of edge servers and the diversity of vehicle offloading modes, Zhang *et al.* [20] determined the target server and data transmission mode based on Q-learning.

When the mobility of users is involved, the significant issue becomes to decide a series of offloading decisions, which is related to the trajectory of users and the network dynamics [21]. Most existing works about mobile offloading focused on balancing the service migration cost and the communication cost by utilizing the formalized models. Rodrigues *et al.* [22] proposed a heuristic algorithm to maximize the cost-effectiveness, considering simultaneous communication, computation, and service migration. By means of the migration and power control strategies in MEC, the workload between edge servers is balanced. Lee *et al.* [23] studied a mobile offloading strategy based on perceived quality of service (QoS) in MEC networks, which leverages the Markov decision process (MDP) to select the optimal edge server to minimize migration and network costs. Brandherm *et al.* [24] developed a general mobile offloading architecture using reinforcement learning, which aimed to minimize the cost of service migration under energy constraints. These works mostly considered a static MEC scenario or deemed the edge servers' states known in advance, while ignoring the time-varying and unpredictable MEC environment in the real world.

DTs have been applied to bridge the gap between physical and digital worlds [25]. Since DTs generally help simplify the solution of practical problems or meet the actual intelligent requirements, DT has been widely expected as a key enablers in 6G in the complex internet of everything (IoE) [26]. Hofmann *et al.* [27] proposed a DT for logistics systems, which determined the optimal scheduling strategy through performance prediction based on simulation and achieved simple deployment and scalability by deploying DT as a cloud-based service. Gehrman *et al.* [28] proposed a DT for industrial automation and control system, and design-driven security requirements for data sharing and control based on DT are determined. Most existing works on DT ignore that a certain deviation exists between the true value and the estimated value of DT.

Most existing works on mobile offloading did not consider the correlation between the offloading decisions during user movement and the time-varying MEC environment. In this paper, we propose an online dynamic mobile offloading scheme for MEC based on DRL and introduce a DITEN architecture to predict the future status of MEC environment.

## III. SYSTEM MODEL AND PROBLEM FORMULATION

### A. Digital Twin Edge Network Model in 6G

Fig. 1 depicts the DITEN architecture in the future 6G wireless cellular network scenario, which consists of physical entity layer and DT layer. In the physical entity layer, edge servers, as small-scale computational units, are deployed on base stations (BSs) to provide computing services for mobile devices (MDs). Edge servers in the scenario are denoted as a set  $\mathbb{E}$ . MDs, connected to edge servers through wireless communication within the coverage of BSs, offload their computing tasks to edge servers

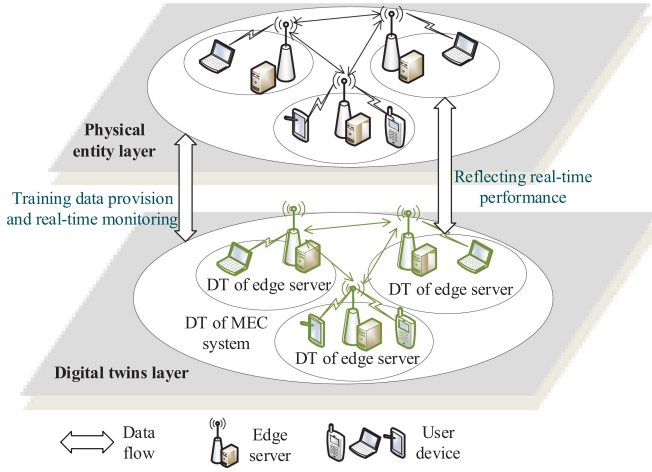


Fig. 1. The architecture of digital twin edge network.

with specific requirements during mobility. In the DT layer, the DTs of objects in MEC form the essential functionalities of the physical entity layer to assist the entire service process for achieving more efficient offloading decisions. In this paper, we consider two categories of DTs, i.e., DTs of edge servers and DT of the entire MEC system.

The DT of an edge server is a digital replica of the edge server, which interacts constantly with the edge server and updates itself with the actual network topology, the requests from MDs, etc. Such DTs within a geographical range may be stored on the edge servers themselves or other resource-rich edge servers. Note that DTs can't fully reflect the edge servers' state and may have an estimated deviation from the true value of the edge server state. In this paper, we use the deviation of available CPU frequency  $\hat{f}_i$  to describe the deviation between real edge server and its DT, which can be either positive or negative. For edge server  $i$ , its digital twin  $\mathcal{D}_i^e$  can be expressed as

$$\mathcal{D}_i^e = \Theta(f_i, \hat{f}_i), \quad (1)$$

where  $f_i$  is the estimated computation performance of edge server  $i$ .

Despite DTs of edge servers, we also assume a DT of the entire MEC system to map the complex interactions in the MEC system. The DT of MEC system can provide a relatively simple estimated output reflecting the current state of the system while providing training data for mobile offloading decision agent and monitoring the entire system state in real time. As shown in Fig. 1, DT of the system interacts and feedbacks with the MEC environment in real time, thus can learn the transformation rules of the environment. Thereby, DT can establish a digital representation system similar to the real environment for obtaining the estimated performance value of system without learning about the implementation details of MDs and edge servers in the system. The DT of the system  $\mathcal{D}^s$  can be expressed as

$$\mathcal{D}^s = \Theta(E^s, R^s), \quad (2)$$

where  $E^s$  is the edge servers' condition in the entire MEC system and  $R^s$  is the estimated rewards of choosing different edge servers.

## B. Offloading in DITEN

Fig. 2 depicts a user generates the offloading tasks to edge servers for computing service during mobility. The user is served by diverse edge servers during the movement based on the available computational performances of edge servers as well as the user's location. We assume that a user has offloading task  $k_t \triangleq \{\eta_t, \lambda_t, \gamma_t\}$  at time slot  $t$ , where  $\eta_t$  denotes the size of the offloading task measured in bits,  $\lambda_t$  is the total CPU cycle number required to accomplish the offloading task, and  $\gamma_t$  depicts the latency requirement of the task  $k_t$ . We use  $\mathbb{E}_t \subset \mathbb{E}$  to denote the available edge servers at time slot  $t$  and  $e_t \in \mathbb{E}_t$  to express the served edge server for the MD at slot  $t$ . The mobile offloading decision module is responsible for deciding a series of service edge servers  $\{e_1, \dots, e_\tau\}$  (assume the entire journey of the user takes  $\tau$  time slots) during user mobility based on the estimated computational performances of servers and the requirements of tasks at  $t$ .

Due to the user mobility and the network dynamics, a user should offload the ongoing service to another appropriate edge server when moving far away from the current edge server. For example, a user following a certain trajectory in Fig. 2 at  $t$  slot offloads task to edge server 1, then at  $t + 1$  slot, makes the migration decision to offload the task to another edge server with lighter load. The service migration that a user changes the served edge server to another edge server incurs the migration cost (i.e., re-authentication cost). For simplicity, we use  $\mathcal{C}$  to represent the cost of a service migration from an edge server to another. Then the service migration cost for a MD of selecting  $e_t$  as the served edge server can be expressed as

$$C(e_t) = h_t \mathcal{C}, \quad (3)$$

where  $h_t \in \{0, 1\}$  is the migration decision result of the service migration module.  $h_t$  is equal to 1 when the server in slot  $t$  is different from the edge server in slot  $t - 1$ , otherwise,  $h_t = 0$ .

## C. Communication Delay Model

The offloading of computing tasks from a MD to an edge servers produces the uplink communication latency, while returning results to the MD produces the downlink communication latency. Particularly, the uplink latency consists of the wireless transmission latency  $T_u^{\text{com}}(e_t)$  from the user to the wireless access BS  $b_t$  and the wired transmission latency  $T_p^{\text{com}}(e_t)$  from the wireless access BS  $b_t$  to the served edge server  $e_t$ , which can be expressed as

$$T^{\text{com}}(e_t) = T_u^{\text{com}}(b_t) + T_p^{\text{com}}(e_t). \quad (4)$$

Assume a MD is always access to the nearest BS  $b_t$  at time slot  $t$ . Given the Signal Interference Noise Ratio (SINR) and the channel bandwidth  $W$  between the user and the access BS in the uplink wireless transmission model, the maximum achievable uplink wireless transmission rate is obtained by Shannon's theorem. The wireless transmission latency of offloading task  $k_t$  can be expressed as

$$T_u^{\text{com}}(b_t) = \frac{\eta_t}{W \log_2(1 + \text{SINR}_t)}. \quad (5)$$



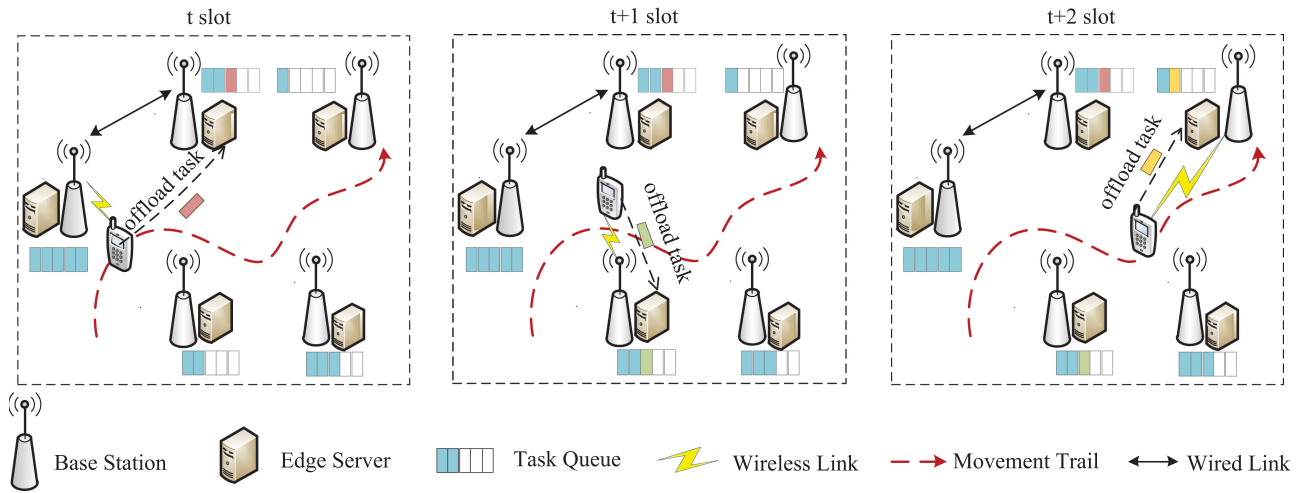


TABLE I  
NOTATION SETTING

Notation	Meaning
$\mathbb{E}$	Set of edge servers
$\mathbb{E}_t$	Set of candidate edge servers at slot $t$
$e_t$	Service edge server at slot $t$
$\mathcal{D}_i^e$	DT of edge server $i$
$\mathcal{D}^s$	DT of MEC system
$\eta_t$	Offloading task size at slot $t$
$\lambda_t$	CPU number of computing unit data at slot $t$
$\gamma_t$	Required delay of the task at slot $t$
$f_{e_t}$	Estimated CPU frequency of edge server $e_t$ by its DT
$\hat{f}_{e_t}$	Estimation error between edge server $e_t$ and its DT
$T^{com}(e_t)$	Communication latency at slot $t$
$G^{cmp}(e_t)$	Latency gap between estimation and real value
$\tilde{T}^{cmp}(e_t)$	Estimated computation latency of edge server $e_t$
$T^{cmp}(e_t)$	Computation latency at slot $t$
$T^{glob}(e_t)$	Offloading latency at slot $t$
$C(e_t)$	Consumption service migration cost at slot $t$
$\beta$	Upper limit of migration rate
$Q(t)$	Migration cost deficit queue length at slot $t$
$\mathbf{S}$	State set of MDP
$\mathbf{A}$	Action set of MDP
$\mathbf{R}$	Reward function set of MDP

that the user can only choose the optimal computing service edge server within an appropriate optional range. Table I shows the main notation settings in the paper.

#### IV. DRL-BASED MOBILE OFFLOADING WITH DT ASSISTANCE

In this section, we first simplify the optimization problem **P1** to a dynamic decision problem by Lyapunov queue optimization technique. Then, we use MDP to model the entire environment and DRL to solve the simplified problem in an online manner.

##### A. Problem Simplification

The optimization problem **P1** cannot be directly solved. Three main challenges are described as follows.

- Firstly, the long-term migration cost limitation (11a) is a challenge to solve **P1**, as the offloading decisions of a user at different time slots become correlated with each other. If too much service migration cost is currently used, it will lead to a shortage of future service migration cost.
- Secondly, to achieve the optimal solution of **P1**, it is necessary but difficult to know the state of the entire MEC environment during the user's movement process, including the user computing task each time slot, user's trajectory, and edge server load condition.
- Thirdly, **P1** is a non-linear programming problem and the complexity of solving **P1** increases exponentially with the length of the user's entire movement process. Therefore, even if the entire MEC environment and user information are known, solving **P1** is complicated.

In order to cope with the challenge of solving the optimization problem **P1** brought by the limitation of long-term service migration cost, the Lyapunov dynamic deficit queue optimization method is used. Lyapunov optimization is widely used in control theory to ensure system stability in different forms. Generally, the Lyapunov function, which measures the multi-dimensional state of the system, is defined as becoming larger when the system moves to a bad state, and system stability can be achieved by taking a control action to shift it toward the negative zero direction. The first step of Lyapunov optimization is to construct virtual queues to meet the constraints of the problem to be solved, and then define the Lyapunov function to describe the square of the backlog of all virtual queues in time slot  $t$ . Next, define the difference of the Lyapunov function between the two time slots, i.e., the Lyapunov drift. Finally, minimize the Lyapunov drift of each slot, and always push the backlog to a lower congestion state to intuitively maintain the stability of the queue. The virtual dynamic deficit queue divides the long-term migration cost limit into the available migration cost of each time slot while adaptively and dynamically balancing latency performance and service migration cost to make the optimal decisions.

We design a dynamic virtual migration cost deficit queue to guide the mobile offloading decision module for making appropriate decision. The length of the migration cost deficit queue can be defined as the deviation between the used migration cost and available migration cost. The user's entire journey is divided into  $\tau$  time slots and the limit of the total migration cost is  $\beta C\tau$ , where  $\beta$  is the upper limit of the service migration rate. Thus, the migration cost available in one slot is  $\beta C$ . The evolution of the migration cost deficit queue  $Q(t)$  can be expressed as

$$Q(t+1) = \max\{Q(t) + C(e_t) - \beta C, 0\}, \quad (12)$$

where  $C(e_t) - \beta C$  is the deviation of the migration cost during time slot  $t$ . According to the designed dynamic migration cost deficit queue, Lyapunov optimization technology transforms the original problem **P1** into an optimization problem without long-term constraints. The simplified objective function can be expressed as

$$\mathbf{P2} : \arg \min_{\{e_1, \dots, e_\tau\}} \sum_{t=1}^{\tau} [v T^{glob}(e_t) + Q(t)C(e_t)], \quad (13)$$

$$\text{s.t. } T^{glob}(e_t) \leq \gamma_t \quad (13a)$$

$$e_t \in \mathbb{E}_t \quad (13b)$$

where  $v$  and  $Q(t)$  are positive control parameters to balance the tradeoff between the processing latency performance and migration cost consumption dynamically.

Note that  $Q(t)$  in **P2** plays a dynamic weighting role. If the value of  $Q(t)$  is large, it attempts to make the subsequent  $C(e_t)$  smaller in order to keep the overall minimization of **P2**. By solving **P2**, we can make an optimal decision on the sequence of served edge servers when the user's trajectory is known. This is not only the current optimal decision, but also an optimal decision in the long run under the constraint of accumulated service migration cost.

### B. Formalization of MDP

The solution of the simplified optimization function **P2** requires the accurate future information of MEC environment. However, due to the dynamic requests and network topology, it is difficult to predict the state of the future environment. Therefore, it is necessary to design an online decision-making mobile offloading scheme to make the optimal decision without the future environment information. Fortunately, the goal of reinforcement learning is to obtain online decision-making ability through continuous interactive learning with the environment, while making the optimal decision sequence for optimizing the long-term future cumulative reward. In this section, we first use a discrete time MDP to describe the dynamic MEC scenario, and then use the DRL algorithm based on *Actor – Critic* to deal with the long-term formalized optimization problem **P2**.

It is generally believed that changes in available computing resources and changes in the described wireless environment follow Markov property. Those three key elements of the learning environment system, namely state set **S**, action set **A**, and reward function set **R** can be expressed as follows.

a) *State*: The state of MDP describes the running status of available edge servers around a MD and the offloading cost if the MD turns to the edge server  $e_t$  for service. At each decision epoch  $t$ , the state of the MEC system can be characterized by

$$S_t = \{d_{e_t}, f_{e_t}, \hat{f}_{e_t}, Q(t) | e_t \in \mathbb{E}_t\}, \quad (14)$$

where  $d_{e_t}$  is the distance between the user and the edge server  $e_t$ ,  $f_{e_t}$  is the status value of edge server  $e_t$  estimated by its DT,  $\hat{f}_{e_t}$  is the estimated deviation between the true value of edge server and its DT,  $Q(t)$  is the length of migration cost deficit queue. (14) assumes that the DT of an edge server provides the estimation of the edge server's state, while has deviation  $\hat{f}_{e_t}$ . The states of the entire journey of the MD  $t \in [1, \tau]$  form the overall state space  $\mathbf{S} \triangleq \{S_t | t \in [1, \tau]\}$ .

b) *Action*: The action describes the policy of the agent. Based on the current system state  $S_t$ , mobile offloading module makes an *action*  $A_t$ , which can be expressed as

$$A_t = \{e_t | e_t \in \mathbb{E}_t\}, \quad (15)$$

where  $A_t$  is the chosen action from the possible set  $\mathbb{E}_t$ . It means that at slot  $t$ , the selected service edge server is  $e_t$ . The actions on the set of time slots  $t \in [1, \tau]$  form the overall action space  $\mathbf{A} \triangleq \{A_t | t \in [1, \tau]\}$ .

c) *Reward*: The reward function reflects what action decisions benefit the system. According to the current system state  $S_t$  and the chosen action  $A_t$  at time epoch  $t$ , the reward obtained by MD can be expressed as

$$R_t = -vT^{\text{glob}}(e_t) - Q(t)C(e_t). \quad (16)$$

The reward function set  $\mathbf{R} \triangleq \{R_t | t \in [1, \tau]\}$  is the set of the obtained rewards in all time slot  $t \in [1, \tau]$ . Since the reward function measures offloading latency and consumption of migration cost, obtaining a series of actions to maximize the cumulative reward can achieve our objective.

It should be noted that the offload delay is a component of the reward function, which is affected by the DT deviation.

Therefore, the DT deviation may cause mistakes in the reward setting, which cannot accurately motivate the system to make useful decisions and may lead to learning failure. To deal with this issue, we use (10) to calibrate the computation delay and use the TD deviation as one of the input states of the training network. In this way, the corresponding reward is calibrated to eliminate the effect of DT deviation on model learning.

The state value function is the expected cumulative rewards at current state  $S_t$  and can be denoted as  $V^\pi(S_{t+1}) = R_t + \varepsilon V^\pi(S_t)$  where  $\varepsilon$  is the reward decaying factor. The state value function can measure the benefits of the current state. By solving the MDP, we can obtain a policy that minimizes offload delay and migration costs.

### C. The DRL Framework Based on Actor – Critic Networks

Traditional reinforcement learning methods such as Q-learning use a Q-table to store the reward value corresponding to each state and action. In DITEN scenario, the state of the environment is complex and changeable, which leads to the rapid increase in the storage space required by Q-table if traditional reinforcement learning methods are used. DRL algorithm combines deep learning network and reinforcement learning method, which uses deep network to replace the role of Q-table. Thus, the storage space of Q-table can be saved. The DRL agent interacts with the DITEN to learn the rules of the environment.

We use the *Actor – Critic* algorithm [29] as a framework for DRL algorithm. The *Actor – Critic* algorithm is based on an idea similar to the adversarial training neural network to train the *Actor* neural network for generating decision actions and the *Critic* neural network for estimating the quality of the current system state. Specifically, the *Critic* network is responsible for estimating the value of the state value function under current state  $S_t$ , and updating its parameters based on the reward value  $R_t$  given by the environment. The *Actor* network plays the role of decision-maker and is responsible for generating corresponding action decisions based on the current state  $S_t$ . The *Actor* network adjusts its parameters according to the judgment value given by the *Critic* network to increase or decrease the output probability of the chosen action. Obviously, *Actor* and *Critic* agents need to continuously interact with the DITEN environment during the training process to achieve the purpose of making optimal decisions. In the *Actor – Critic* algorithm, the learning agent consists of two separate entities: *Actor* and *Critic*. In the following, the training process of these two entities will be discussed in detail.

*Critic Agent*: The objective of the *Critic* agent is to estimate the expected cumulative reward value corresponding to the environment state  $S_t \in \mathbf{S}$ . *Critic* agent relies on neural network to approximate the state value function  $V^\pi(S_t)$ , that is, a linear combination to fit the state value function corresponding to the environmental state. Denote the estimated state value function by *Critic* network parameters as  $\tilde{V}(S_t, \mathbf{W}_c)$ , where  $\mathbf{W}_c$  is its parameters and the state  $S_t$  of the DITEN environment is its input.

The *Critic* network is trained based on the Temporal Difference (TD) learning. The TD error can be calculated by the state

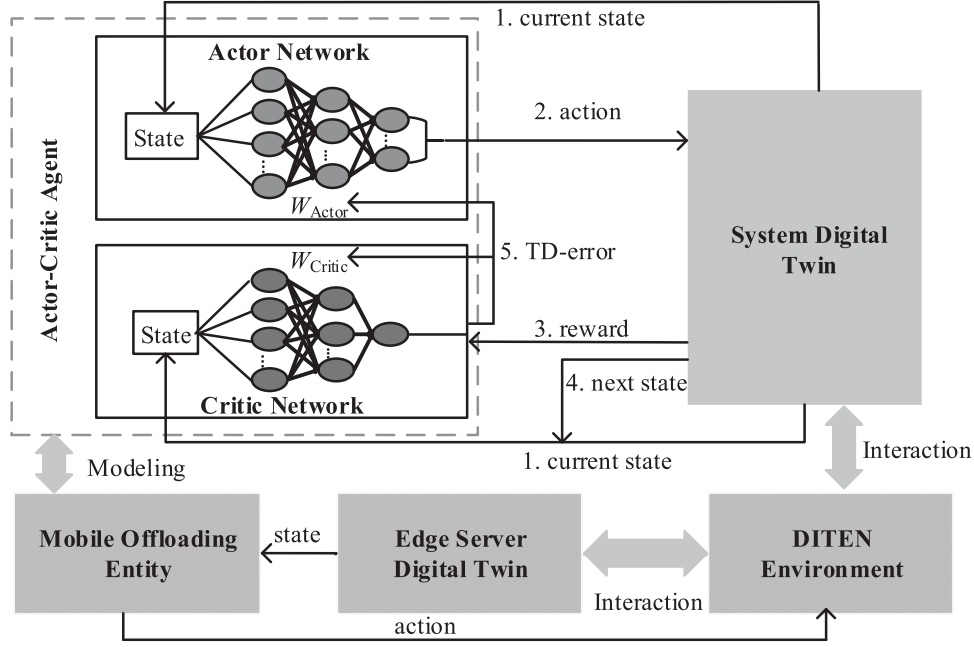


Fig. 3. Illustration of the training framework of DRL-based offloading in DITEN.

value function of two adjacent time slot and the current reward, which can be written as

$$\theta_t = R_{t+1} + \tilde{V}(S_{t+1}, \mathbf{W}_c) - \tilde{V}(S_t, \mathbf{W}_c). \quad (17)$$

The *Critic* network updates its parameters according to  $\theta_t$ . Intuitively, the *Critic* network considers  $R_{t+1} + \tilde{V}(S_{t+1}, \mathbf{W}_c)$  as the true cumulative rewards in state  $S_t$  and  $\tilde{V}(S_t, \mathbf{W}_c)$  as the prediction cumulative rewards of the neural network in the state  $S_t$ . Thus, it reasonably updates the parameters of the neural network by the loss squared between the true value and the estimated value. The *Critic* network parameters are updated as

$$\mathbf{W}_c = \mathbf{W}_c + l_c \theta_t \nabla_{\mathbf{W}_c} \tilde{V}(S_t, \mathbf{W}_c), \quad (18)$$

where  $l_c$  is the learning rate.

**Actor Agent :** The *Actor* network is a policy-based network, which can output the probability distribution of each action. The *Actor* agent selects the action based on the output probability distribution of *Actor* network. The probability distribution of optional actions output by the *Actor* network can be denoted as  $\pi(S_t, \mathbf{W}_a)$ .

The purpose of the *Actor* agent is to make the action with optimal cumulative rewards in the each state  $S_t$ . Thus, the value function in the current state has the largest value after taking the optimal action. That is the action with high rewards should get a higher probability in the output strategies of *Actor* network. The *Actor* network is different from the traditional policy-based network. It uses the TD error  $\theta_t$  given by the *Critic* network to control the update range and direction of its parameters, while realizing the single-step update of the *Actor* network parameters. The *Actor* network parameters are updated as

$$\mathbf{W}_a = \mathbf{W}_a + l_a \theta_t \nabla_{\mathbf{W}_a} [\log(\pi(S_t, \mathbf{W}_a))], \quad (19)$$

where  $\nabla_{\mathbf{W}_a} [\log \pi(S_t, \mathbf{W}_a)]$  is the direction gradient vector, and the *Actor* adjusts the update parameters according to the direction vector. Intuitively speaking, the *Actor* neural network uses  $R_{t+1} + \tilde{V}(S_{t+1}, \mathbf{W}_c)$  in TD-error as the actual cumulative rewards,  $\tilde{V}(S_t, \mathbf{W}_c)$  as an expectation of cumulative rewards. Thus, when  $\theta_t$  is positive, the current chosen action can obtain a greater cumulative rewards. The network parameter  $\mathbf{W}_a$  is adjusted in the direction where the output probability of this action is greater. Conversely, when the difference between the actual rewards and the expected rewards is negative, the current action is not a good decision, the action is a poorly performing action and the network parameter  $\mathbf{W}_a$  should be adjusted to the direction where the output probability of this action becomes smaller.

#### D. DRL-Based Mobile Offloading Scheme

Fig. 3 illustrates the interactive framework between dynamic offloading decision entity, edge server DT, system DT, DITEN environment and *Actor – Critic* agent. As can be seen from Fig. 3, DTs of edge servers and system interact with the MEC environment in real time to build the digital representation of the real environment. Meanwhile, the *Actor – Critic* agent interacts with the DT of system to learn the optimal dynamic offloading decision. The input of the *Actor* network is state, and the output is action where softmax is used to ensure that the output probability of each action adds up to 1. The input of *Critic* network is state and action, and the output is Q value. Both the *Actor* network and the *Critic* network have 3 hidden layers, and the activation function of all hidden layers is relu. It should be noted that the *Critic* network structure is more complex than the *Actor* network because the hidden layer 1 consists of two parts, one of which receives the state and the other receives the action. Specifically, the *Actor* agent observes the state of the



**Algorithm 1: The Actor – Critic Training Algorithm.**


---

**Input:** learning rate  $l_a, l_c$ , Pre-training times  $K_1$ , combined training times  $K_2$ .  
**Output:** The trained Actor – Critic model parameters  $\mathbf{W}_a, \mathbf{W}_c$ .

- 1: Initialize the Actor and Critic network parameters  $\mathbf{W}_a, \mathbf{W}_c$
- 2: **for**  $episode = 1$  to  $K_1$  **do**
- 3:   Initialize state  $S_1$
- 4:   **for**  $t = 1$  to  $\tau$  **do**
- 5:      $A_t \leftarrow \text{Random}(\mathbb{E}_t)$
- 6:     Execute action  $A_t$  in emulator
- 7:     Observe reward  $R_t$  and new state  $S_{t+1}$
- 8:      $\theta_t \leftarrow (R_t + \varepsilon \tilde{V}(S_{t+1}, \mathbf{W}_c) - \tilde{V}(S_t, \mathbf{W}_c))$
- 9:      $\mathbf{W}_c \leftarrow \mathbf{W}_c + l_c \theta_t^2 \nabla_{\mathbf{W}_c} V(S_t, \mathbf{W}_c)$
- 10:   **end for**
- 11: **end for**
- 12: **for**  $episode = 1$  to  $K_2$
- 13:   Initialize state  $S_1$
- 14:   **for**  $t = 1$  to  $\tau$  **do**
- 15:     Obtain  $\tilde{\pi}(S_t, \mathbf{W}_a)$  by Actor network
- 16:      $A_t \sim \tilde{\pi}(S_t, \mathbf{W}_a)$
- 17:     Execute action  $A_t$  in emulator
- 18:     Observe reward  $R_t$  and new state  $S_{t+1}$
- 19:      $\theta_t \leftarrow (R_t + \tilde{V}(S_{t+1}, \mathbf{W}_c) - \tilde{V}(S_t, \mathbf{W}_c))$
- 20:      $\mathbf{W}_c \leftarrow \mathbf{W}_c + l_c \theta_t^2 \nabla_{\mathbf{W}_c} \tilde{V}(S_t, \mathbf{W}_c)$
- 21:      $\mathbf{W}_a \leftarrow \mathbf{W}_a + l_a \nabla_{\mathbf{W}_a} \log \tilde{\pi}(S_t, \mathbf{W}_a) \theta_t$
- 22:   **end for**
- 23: **end for**
- 24: **return** The trained  $\mathbf{W}_a, \mathbf{W}_c$

---

**Algorithm 2: Mobile Offloading Scheme Based on Actor – Critic.**


---

**Input:** The trained Actor – Critic network.  
**Output:** The sequence of optimal offloading decisions  $\{e_1, e_2, \dots, e_\tau\}$ , the training pool *Array*.

- 1: Initialize the training pool *Array*  $\leftarrow \emptyset$  and state  $S_1$
- 2: **for**  $t = 1$  to  $\tau$  **do**
- 3:   Calculate the available actions' probability distribution  $\tilde{\pi}(S, \mathbf{W}_a)$  by Actor network
- 4:    $A_t = e_t \leftarrow \arg \max_{A_t} (\tilde{\pi}(S, \mathbf{W}_a))$
- 5:   Execute action  $A_t$  in environment emulator
- 6:   Observe reward  $R_t$  and next state  $S_{t+1}$
- 7:    $\text{Array} \leftarrow \text{Array} \cup \langle S_t, A_t, R_t, S_{t+1} \rangle$
- 8: **end for**
- 9: **return**  $\{e_1, e_2, \dots, e_\tau\}, \text{Array}$

---

environment in system DT and outputs the optimal offloading decision to act on the system DT, while *Critic* agent calculates the TD-error to update the two agents' parameters according to the current state, next state and the feedback reward. The training center server sends the trained model to the offloading decision module. The offloading decision entity makes decision according to the state given by DT of edge server and the chosen

action acts on the real environment. It is obvious that the agent can achieve the same training result as in the real environment through the DT medium with low cost.

In the training step, the Actor – Critic network directly interacts with the DT of system in the DITEN environment to learn the environment knowledge. The Actor – Critic network takes the state of the DITEN environment as input. Specifically, the Actor network executes the action  $A_t$  in the DT of the system according to the action probability  $P_t$  and the state  $S_t$ , and obtains the reward and state output given by the DT of the system, and feeds the output back to the Critic network. The error TD-error calculated by the Critic network using the output feedback of the Actor network is leveraged to update its own parameters and is also transmitted to the Actor network to update the parameters with the probability  $P_t$ . If the time difference value TD-error is large, it means that the advantage of the action selected by the current Actor network is larger, and a greater probability of occurrence is needed to reduce the time difference value. This process is repeated until the end of training. The detailed training process is shown in Algorithm 1.

After the training, the proposed mobile offloading decision agent is deployed on the mobile offloading management module to make the optimal decision according to the available edge servers' DTs around user. In addition, during the running process, the user's state conversion data is collected and put into the training pool for the agent's retraining. The specific operation process is shown in Algorithm 2. Firstly, the decision agent collects the state of edge servers around the user as input of the trained Actor network (line 3), obtains the outputs the action probability distribution by Actor network, and chooses the action with the largest output probability as the executing action (line 4). Then the chosen action is executed in the real environment and the received environmental feedback value is stored in the training pool to facilitate the agent's retraining (line 5 to line 6).

Generally, the complexity of DRL algorithms is considered from the perspective of computation theory. The computational complexity refers to the consumption of computing resources for obtaining the decision-making result from DRL agent, which can be expressed as the complexity of the matrix multiplication operation by network during calculation. The trained DRL decision-making agent works in an online manner and its decision complexity is positively related to the network structure and the number of candidate edge servers. In the designed network structure, we only add the number of input layer and output layer neurons by default when subjoining the candidate edge servers, which will cause only the dimension of the weight matrix of the input and output layers. Therefore, the complexity of the decision can be expressed as  $\mathcal{O}(n^2)$  related to the number of optional edge servers  $n$ .

## V. NUMERICAL RESULTS

This section evaluates the performance of the proposed scheme by extensive simulations. BSs deployed with edge servers are distributed evenly with the density of  $40/\text{km}^2$  in a  $5 \text{ km} \times 5 \text{ km}$  square area and each BS provides wireless access



TABLE II  
SIMULATION PARAMETERS

Parameter	Value
Transmission channel bandwidth $W$	20 MHz
The sum frequency of servers	[15, 25] GHz
User device transmit power $p_t$	[0.2, 0.6] W
Noise power $\sigma^2$	$2 \times 10^{-12}$ W
Task data volume	[600, 800]kB
CPU cycles required for unit data volume	[200, 400]
Router queuing latency	2ms
Task request latency	[150, 250] ms
Mean of DTs' estimation error $\hat{f}_{et}$	0.5

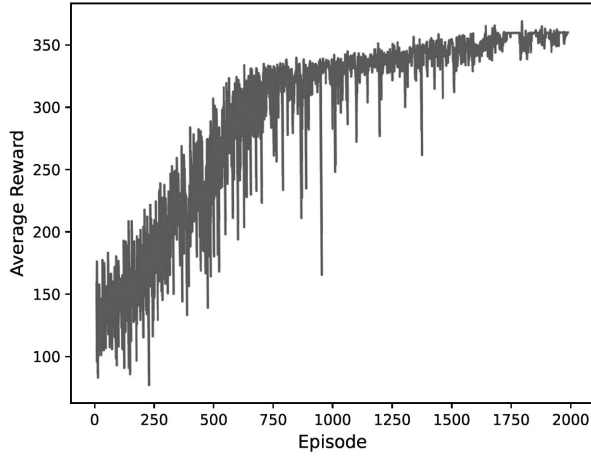


Fig. 4. The convergence performance of *Actor – Critic* agent.

services for users within a radius of 150 m. The trajectories collected in the Microsoft T-Drive project [30] are scaled to our simulation area as the users' trajectories. Users offload computing tasks to edge servers every time slot during mobility. The system DT, which reflects the running state of the entire edge network, provides the DRL agent with training data to achieve efficient training. All simulations are performed using Python 3.6.1 installed on the Windows 7 operating system. The other simulation parameters are shown in Table II. In order to demonstrate the superiority of the proposed scheme, we choose the traditional nearest first optimal mobile offloading scheme as the comparison benchmark, in which the users always select the nearest edge server to perform task offloading. Some of the parameters as variables are explained in the specific experiments. To alleviate the randomness of results, all numerical results take the average of 5000 simulations.

Fig. 4 reflects the convergence trend of the reward function. The reward is set according to (16) considering the latency and migration cost. A small reward means that the task processing latency and migration cost are relatively small, which means that the user can be provided with a better offloading service in an energy-saving manner. As can be seen from Fig. 4, the reward function tends to converge after 1750 rounds of training. This shows that the trained model is suitable for DTEN and has good convergence performance. As the proposed scheme

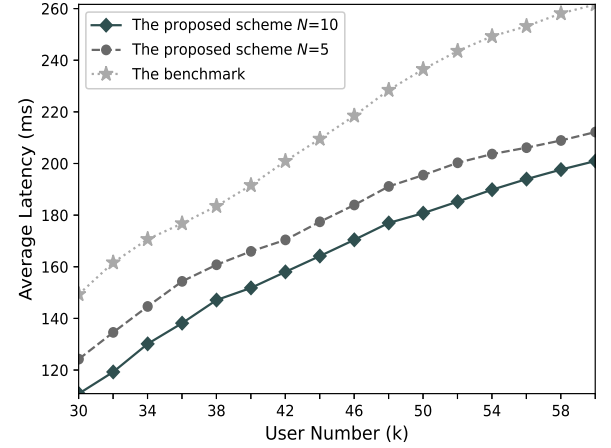


Fig. 5. Comparison of the average latency with the varying user number and candidate edge server number  $N$ .

is placed in the mobile offloading decision module after the training step, the optimal mobile offloading decision runs in an online manner, which ensures that the model is usable and meets the delay requirements.

Fig. 5 compares the average latency of the proposed scheme and the benchmark scheme over the varying user number and candidate edge server number  $N$ . The average latency refers to the ratio of the total latency of offloading tasks to the number of tasks in the entire journey. It measures the user's average satisfaction with task offloading service throughout the trip, where the latency of a single offloading task is calculated by (4) and (10). From a vertical point of view, when the number of users remains unchanged, the average latency of the proposed scheme is always lower than the benchmark scheme. When the number of selectable edge servers is 5 ( $N = 5$ ), the average latency of the benchmark scheme is about 1.2 times, even 1.5 times, that of the proposed scheme and when  $N = 10$ . This is because the proposed scheme using DRL makes an optimal decision with considering the future state of the MEC system, which is not only the current optimal, but also the optimal decision in the long run. The available resources of a single user will be limited as the number of users increases, resulting in a higher task failure rate (the ratio of the number of tasks that failed to offload to the total number of tasks) and making the average latency approach to the task request latency. In addition, with the redundant computing resources of more edge servers, the average latency will be intuitively reduced.

Fig. 6 measures the average latency over the varying deployment density of BSs and the DT estimation error  $\hat{f}_{et}$ . Under the condition that the DT estimation error is constant, as the BS deployment density increases, the corresponding average latency will decrease. This is because users can connect to more BSs, and choose the BS with better service quality to offload tasks. On the other hand, when the BS deployment density is constant, the corresponding average latency is inversely correlated with the DT evaluation error. This can be explained from (9) and (10) that a large positive error means the estimated value of DT is worse than the actual value, thus the actual latency is less than the estimated value.

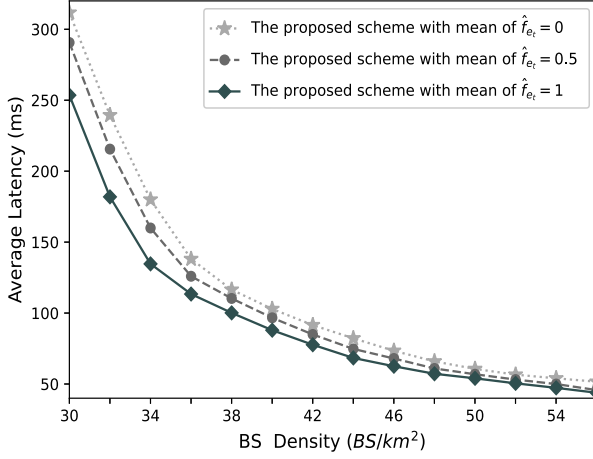


Fig. 6. Comparison of the average latency with the varying BS density and mean of DT estimation error  $\hat{f}_{er}$ .

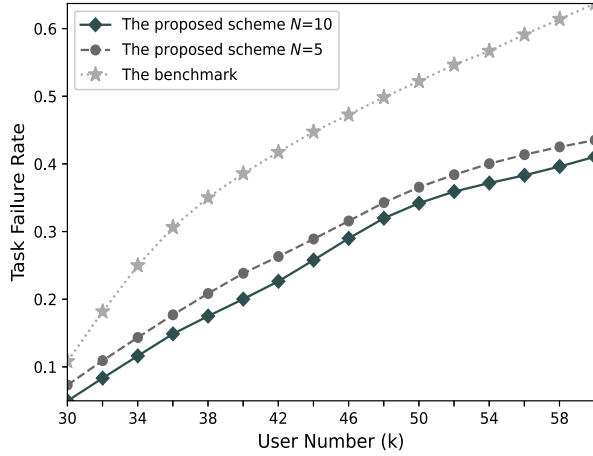


Fig. 7. Comparison of the task failure rate with the varying user number and candidate edge server number  $N$ .

Fig. 7 depicts the the task failure rate changing with the number of users and number of available edge servers  $N$ . The task failure rate is defined as the ratio of the number of failed tasks to the total tasks, which can measure the instantaneous performance of a single offloading task. As can be seen from Fig. 7, the task failure rate of the proposed scheme is always lower than the benchmark scheme, as the proposed scheme always chooses the edge servers from a larger candidate set and carefully considers the dynamic characteristics of edge servers. The benchmark scheme did not consider the dynamics of the MEC environment well. Meanwhile, it can be seen from Fig. 7 that with the increase of the number of users in the simulation area, the task failure rate is on the rise since the large user number results in the overwhelming of edge server. In addition, with the number of users remaining the same, more available candidate edge servers may bring lower task failure rate.

Fig. 8 shows how the service migration rate changes with user movement speed and the migration threshold  $\beta$ . The service migration rate refers to the ratio of the changed number of served edge servers to the total number of slots the entire journey. As re-authentication may be performed when the user offloads the

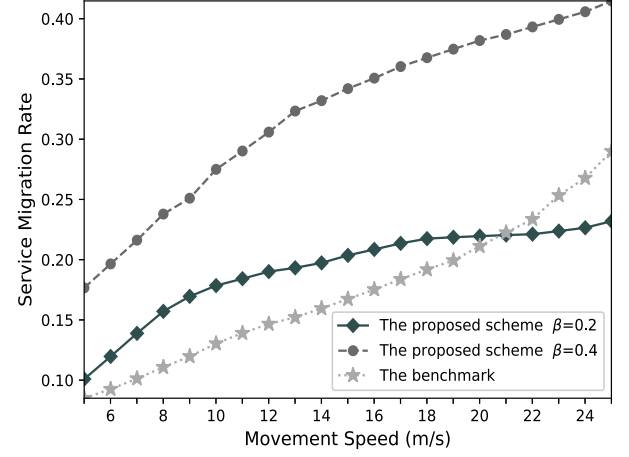


Fig. 8. Comparison of the service migration rate with the varying movement speed and preset migration threshold  $\beta$ .

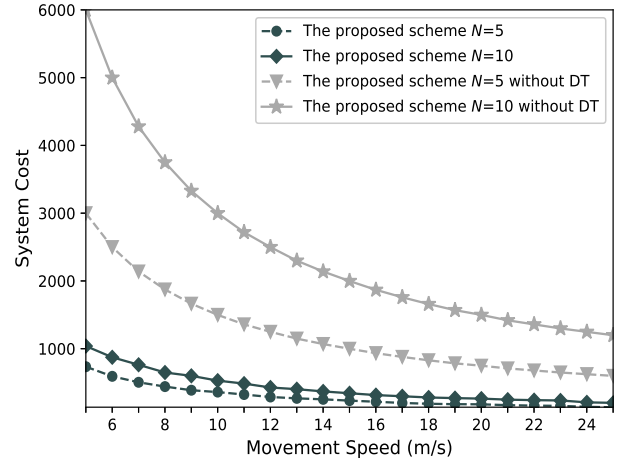


Fig. 9. Comparison of the system cost with the varying movement speed and candidate edge server number  $N$ .

computing task to another edge server, low service migration rate is preferred. As can be seen, if a user moves fast in the application area, the service migration rate would be higher than the slow case. It is noted that with the  $\beta = 0.2$ , the case of movement speed exceed 21 m/s in proposed scheme shows lower service migration rate than the benchmark scheme. This is because the proposed scheme with a lower service migration threshold  $\beta$  means the less available service migration cost, the decision making agent may be choose fewer service migrations to save the cost. In addition, as the increase in movement velocity triggers frequent handovers and severely affects the offload decision of the benchmark scheme, the service migration rate of the benchmark scheme quickly increases and eventually exceeds that of the proposed scheme.

Fig. 9 illustrates how the system cost (the total cost of estimating the states of candidate edge servers during the user's journey) varies with the movement speed of a MD. As shown in Fig. 9, no matter how the moving speed changes, the system cost of DT-assisted scheme is always lower than that of the no DT scheme. This is because without DT the offloading module always estimates the candidate edge servers' information by

querying each candidate edge server, while the DT-assisted scheme simply requires the estimated computing power from the edge servers that store DTs of these candidate edge servers. In addition, Fig. 9 also shows that more available candidate edge servers may bring higher system cost, which is caused by the increase of the number of candidate edge servers leading to more computing power queries.

## VI. CONCLUSION

In this paper, we have developed a digital twin edge network (DITEN) architecture for 6G to efficiently perceive the MEC environment. In DITEN, the mobile offloading issue was formalized to minimize the offloading latency under the constraint of accumulated migration cost, when a user moves across edge servers and requiring computing task offloading. The formal problem of dynamic multi-objective optimization is simplified by using Lyapunov, and the simplified problem is solved by deep reinforcement learning, i.e., *Actor – Critic* network. Finally, we performed simulations to assess the performances as compared with other schemes. Numerical results show that our proposed scheme effectively reduces the delay and task failure rate of computational tasks under the constraints of low system cost.

## REFERENCES

- [1] I. Tomkos, D. Klonidis, E. Pikasis, and S. Theodoridis, "Toward the 6G network Era: Opportunities and challenges," *IT Prof.*, vol. 22, no. 1, pp. 34–38, 2020.
- [2] K. Zhang, S. Leng, X. Peng, P. Li, S. Maharjan, and Y. Zhang, "Artificial intelligence inspired transmission scheduling in cognitive vehicular communications and networks," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1987–1997, Apr. 2019.
- [3] K. B. Letaief, W. Chen, Y. Shi, J. Zhang, Y. A. Zhang, "The roadmap to 6G: AI empowered wireless networks," *IEEE Commun. Mag.*, vol. 57, no. 8, pp. 84–90, Aug. 2019.
- [4] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 450–465, Feb. 2018.
- [5] W. Sun, J. Liu, Y. Yue, and Y. Jiang, "Social-aware incentive mechanisms for D2D resource sharing in IIoT," *IEEE Trans. Ind. Informat.*, vol. 16, no. 8, pp. 5517–5526, Aug. 2020.
- [6] K. Zhang, Y. Zhu, S. Maharjan, and Y. Zhang, "Edge intelligence and blockchain empowered 5G beyond for industrial internet of things," *IEEE Netw. Mag.*, vol. 33, no. 5, pp. 12–19, Sep./Oct. 2019.
- [7] T. K. Rodrigues, K. Suto, H. Nishiyama, J. Liu, and N. Kato, "Machine learning meets computation and communication control in evolving edge and cloud: Challenges and future perspective," *IEEE Commun. Surveys Tut.*, vol. 22, no. 1, pp. 38–67, Jan.–Mar. 2020.
- [8] F. Wang, R. Qin, J. Li, Y. Yuan, and X. Wang, "Parallel societies: A computing perspective of social digital twins and virtual real interactions," *IEEE Trans. Comput. Social Syst.*, vol. 7, no. 1, pp. 2–7, Feb. 2020.
- [9] C. Gehrman and M. Gunnarsson, "A digital twin based industrial automation and control system security architecture," *IEEE Trans. Ind. Informat.*, vol. 16, no. 1, pp. 669–680, Jan. 2020.
- [10] F. Zhang, G. Liu, X. Fu, and R. Yahyapour, "A survey on virtual machine migration: Challenges, techniques, and open issues," *IEEE Commun. Surv. Tut.*, vol. 20, no. 2, pp. 1206–1243, 2018.
- [11] V. Huy Hoang, T. M. Ho, and L. B. Le, "Mobility-aware computation offloading in MEC-based vehicular wireless networks," *IEEE Commun. Lett.*, vol. 24, no. 2, pp. 466–469, Feb. 2020.
- [12] S. Wang, R. Ugaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge computing based on markov decision process," *IEEE/ACM Trans. Netw.*, vol. 27, no. 3, pp. 1272–1288, Jun. 2019.
- [13] D. Zhao, T. Yang, Y. Jin, and Y. Xu, "A service migration strategy based on multiple attribute decision in mobile edge computing," *IEEE 17th Int. Conf. Commun. Technol.*, 2017, pp. 986–990.
- [14] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys*, vol. 19, no. 3, pp. 1628–1656, Jul.–Sep. 2017.
- [15] W. Sun, J. Liu, and Y. Yue, "AI-enhanced offloading in edge computing: When machine learning meets industrial IoT," *IEEE Netw.*, vol. 33, no. 5, pp. 68–74, Sep./Oct. 2019.
- [16] A. Ndikumana, S. Ullah, T. LeAnh, N. H. Tran, and C. S. Hong, "Collaborative cache allocation and computation offloading in mobile edge computing," in *Proc. 19th Asia-Pacific Netw. Oper. Manag. Symp.*, 2017, pp. 366–369.
- [17] T. Koketsu Rodrigues, K. Suto, and N. Kato, "Edge cloud server deployment with transmission power control through machine learning for 6G internet of things," *IEEE Trans. Emerging Topics Comput.*, to be published.
- [18] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3590–3605, Dec. 2016.
- [19] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. S. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Trans. Commun.*, vol. 65, no. 8, pp. 3571–3584, Aug. 2017.
- [20] K. Zhang, Y. Zhu, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Deep learning empowered task offloading for mobile edge computing in urban informatics," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 7635–7647, Oct. 2019.
- [21] Z. Rejiba, X. M. Bruin, and E. M. Tordera, "A Survey on mobility-induced service migration in the fog, edge, and related computing paradigms," *ACM Comput. Survey*, vol. 52, no. 2, pp. 90–123, 2019.
- [22] T. G. Rodrigues, K. Suto, H. Nishiyama, N. Kato, and K. Temma, "Cloudlets activation scheme for scalable mobile edge computing with transmission power control and virtual machine migration," *IEEE Trans. Comput.*, vol. 67, no. 9, pp. 1287–1300, Sep. 2018.
- [23] J. Lee, J. Kim, Y. Tae, and S. Pack, "QoS-aware service migration in edge cloud networks," in *Proc. IEEE Int. Conf. Consum. Electron. - Asia (ICCE-Asia)*, 2018, pp. 206–212.
- [24] F. Brandherm, L. Wang, and M. Mühlhäuser, "A learning-based framework for optimization service migration in mobile edge clouds," in *Proc. 2nd Int. Workshop Edge Syst., Anal. Netw. (EdgeSys)*, 2019, pp. 12–17.
- [25] H. J. Uhlemann, C. Lehmann, R. Steinhilper, "The digital twin: Realizing the cyber-physical production system for industry 4.0," *Procedia CIRP*, vol. 61, pp. 335–340, 2017.
- [26] H. Viswanathan and P. E. Mogensen, "Communications in the 6G Era," *IEEE Access*, vol. 8, pp. 57063–57074, 2020.
- [27] W. Hofmann and F. Branding, "Implementation of an IoT- and cloud-based digital twin for real-time decision support in port operation," *12th Int. Conf. Comput. Sci. Edu.*, 2017, pp. 277–282.
- [28] C. Gehrman and M. Gunnarsson, "A digital twin based industrial automation and control system security architecture," *IEEE Trans. Ind. Informat.*, vol. 16, no. 1, pp. 669–680, 2020.
- [29] I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Trans. Veh. Technol.*, vol. 42, no. 6, pp. 1291–1307, Nov. 2012.
- [30] J. Yuan, Y. Zhang, and C. Zhang, "T-drive: Driving directions based on taxi trajectories," *Sigspatial Int. Conf. Advances in Geographic Inf. Syst. ACM*, 2010, pp. 99–108.



**Wen Sun** (Senior Member, IEEE) received the B.E. degree from the Harbin Institute of Technology, in 2009 and the Ph.D. degree in electrical and computer engineering from the National University of Singapore, in 2014. She is currently a Full Professor with the School of Cybersecurity, Northwestern Polytechnical University. Her research interests cover a wide range of areas including wireless mobile communications, IoT, 5G, and blockchain. She has published more than 50 peer-reviewed papers in various prestigious IEEE journals and conferences, including IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, IEEE NETWORK, IEEE WIRELESS COMMUNICATIONS. She was the recipient of the Best Paper Award of GlobeCom2019. She serves as Publicity Chair for WiMob2019 and CNS2020, and TPC Member for ICC and GlobeCom, in 2018 and 2019, respectively.





**Haibin Zhang** (Member, IEEE) received the B.S. degree in applied mathematics from the Ocean University of China, in 2003, and the Ph.D. degree in computer science and technology from Xidian University, in 2007. He is currently a professor in Northwestern Polytechnical University. His research interests cover a wide range of areas including artificial intelligence security, distributed computing and trusted computing. He has published more than 40 peer-reviewed papers in various prestigious IEEE journals and conferences, including IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, *IEEE Communications Magazine*, IEEE INTERNET OF THINGS JOURNAL, IEEE SYSTEMS JOURNAL.



**Rong Wang** (Student Member, IEEE) received the B.E. degree in software engineering from Chang'an University, China, in 2017 and the M.Sc. degree from Xidian University, China, in 2020. Her research interests include mobility management and reinforcement learning.



**Yan Zhang** (Fellow, IEEE) received the Ph.D. degree from the School of Electrical and Electronics Engineering, Nanyang Technological University, Singapore. He is currently a Full Professor with the Department of Informatics, University of Oslo, Oslo, Norway. His current research interests include next-generation wireless networks leading to 5G beyond/6G, green and secure cyber-physical systems (e.g., smart grid and transport). He is an Editor (or Area Editor, Associate Editor) for several IEEE publications, including *IEEE Communications Magazine*, *IEEE Network Magazine*, IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, IEEE TRANSACTIONS ON GREEN COMMUNICATIONS AND NETWORKING, IEEE COMMUNICATIONS SURVEY AND TUTORIALS, IEEE INTERNET OF THINGS JOURNAL, IEEE SYSTEMS JOURNAL, *IEEE Vehicular Technology Magazine*, and IEEE BLOCKCHAIN TECHNICAL BRIEFS. He is a Symposium/Track Chair in a number of conferences, including IEEE ICC 2021, IEEE Globecom 2017, IEEE PIMRC 2016, IEEE SmartGridComm 2015. He is an IEEE Vehicular Technology Society Distinguished Lecturer during 2016-2020 and he is named as CCF 2019 Distinguished Speaker. He is the Chair of IEEE Communications Society Technical Committee on Green Communications and Computing (TCGCC). He is an elected member of CCF Technical Committee of Blockchain. In both 2019 and 2018, Prof. Zhang was a recipient of the Global Highly Cited Researcher Award (Web of Science top 1% most cited worldwide). He is Fellow of IET, Elected Member of Academia Europaea (MAE), and Elected Member of the Norwegian Academy of Technological Sciences (NTVA).