2021 13th International Conference on Wireless Communications and Signal Processing (WCSP) | 978-1-6654-0785-4/21/$31.00 ©2021 IEEE | DOI: 10.1109/WCSP52459.2021.9613687

A Cybertwin-Driven Task Offloading Scheme Based on Deep Reinforcement Learning and Graph Attention Networks

Xiaoxu Zhong and Yejun He⋆
Guangdong Engineering Research Center of Base Station Antennas and Propagation
Shenzhen Key Laboratory of Antennas and Propagation
College of Electronics and Information Engineering, Shenzhen University, 518060, China
Email:1031772642@qq.com, heyejun@126.com⋆

***Abstract*—With the improvement of mobile computing capability, the service demands of user terminals are also increasing. Therefore, task offloading in mobile edge computing (MEC) has always been a research focus. However, most of task offloading schemes are confined to current network paradigms. During the shaping of 6G, it is of value to explore novel task offloading architectures and methods which fit future communication system. In this paper, we present a cybertwin-based system that allows flexible resource deployment and coordination. In addition, by targeting at the graph structure of this system, we propose an energy-efficient and security-enhanced partial-offloading scheme based on a combination of deep reinforcement learning (DRL) and modified graph attention network (GAT). Numerical results demonstrate that the proposed training method converges better and the trained GAT model achieves utility improvement.**

***Index Terms*—Cybertwin, partial offloading, GAT, twin delayed deep deterministic policy gradient (TD3)**

I. Introduction

As the commercialized 5G and its related technologies mature gradually, the activities to shape the next generation of wireless communication are initiated worldwide and 6G is outlined as “hyper-connected experience for all” [1]. Facilitated by the advancement of mobile communication, prospering artificial intelligence (AI) has proliferated smart devices and sophisticated applications. However, there remains mismatch between rigid requirements of these applications and limited computing capability of user devices. Therefore, task offloading should always be available for compute-intensive services.

Mobile edge computing (MEC) was brought up to alleviate or resolve congestion caused by huge data swarming into core networks when assuring satisfactory quality of service (QoS). As edge intelligence emerges and more edge infrastructures are deployed for wireless services, MEC-based frameworks [2]–[4] are discussed and investigated in the past decade. A pure MEC-based network consisting of one energy-harvesting-enabled IoT device and multiple edge devices was investigated in [2]. However, the single-user-centric system does not coincide with the common case that is both multi-user and multi-edge. In addition, this scheme omitted transmission delay and energy consumption of downlink, which is less preferred when considering non-negligible downlink delay or greener objectives. Edge-cloud collaboration architectures were proposed in [3], [4], where tasks could be partially processed at edge nodes and partially offloaded to the cloud via wired links or backhaul links. However, the two architectures neglected that smart vehicles or users are capable of processing some tasks locally, which can be more cost-efficient.

In respect to existing offloading architectures, two key challenges to be resolved are as follows:

- Cloud-centric architectures are reasonable for cloud’s powerful capability, but a part of offloading benefit in these architectures is usually offset by communication cost. As user terminal becomes smarter, end-edge-cloud orchestration mechanism is worth investigating.
- Most existing frameworks are confined to the current network architecture. However, if 6G wants to work for Internet of Everything (IoE) with capability increase by a factor of 10-100 [5] and better energy efficiency, new network architectures are needed [6].

Various reinforcement learning (RL)-based algorithms [7]–[9] that aimed at allocating resource and making task offloading decisions were also investigated. Huang *et al.* proposed a power management scheme based on double Q-learning to realize dynamic voltage and frequency scaling (DVFS) [7]. To optimize the combined utility of the mobile device, a task offloading algorithm based on deep Q-network and actor-critic framework was proposed [8]. Targeting at continuous task offloading control, Ke *et al.* proposed an adaptive computation offloading method based on deep deterministic policy gradient (DDPG) to minimize the utility of energy consumption, transmission delay and allocated bandwidth [9]. Nonetheless, many deep reinforcement learning (DRL)-based algorithms are confined to deep neural network (DNN) and convolutional neural network (CNN), without considering the graph structure of task offloading data.

In this paper, the performance of task offloading algorithms is improved by better extracting features from original data, and a novel DRL-based scheme is proposed. The main contributions are summarized as follows:

- We propose an end-edge-cloud orchestration system with cybertwin components and a scheduling center. It is a flexible, cache-considered, security-enhanced and energy-efficient framework that supports simultaneous partial offloading of multi-task.
- To improve the performance of current DRL-based task offloading algorithms, we take the graph structure of the scenario into consideration and combine the twin delayed DDPG (TD3) framework and the modified graph attention network (GAT).
- Comparing with other baseline algorithms and strategies, we show that the proposed method achieves performance boost.

The remainder of this paper is organized as follows. The system model is elaborated in section II and the objective function is formulated in section III. Then, a novel DRL-based algorithm integrated with GAT is proposed in section IV. Numerical results are analyzed in section V and followed by a conclusion.

II. System Model

To fit for future communication system, our system is built on top of the main framework in [6]. But our system is customized with features that suit centralized task offloading problem, such as a scheduling center, cloud resource partition and cache-offloading-combined mechanism, and details that are needed for task offloading to work out, including components of edge nodes and data flow.

As shown in Fig. 1, this is a hierarchical system consisting of service layer, edge layer and cloud layer. The bottom is where services are requested and implemented. It is a set of mobile users and denoted as $U = \{1, 2, ..., N\}$. The edge layer and the cloud layer are two intertwined but different layers. In the edge layer, each edge node is a pack of equipment, including but not confined to base stations (BS), edge servers, cameras, road side units (RSU), biometric readers, bar-code or QR-code scanners. Edge nodes are local and densely deployed. Their role is to collect environment information and provide task offloading assistance in three ways. First, an edge node is a host of cybertwins [6] who work as facilitators and windows to handle offloading requests and deliver services to users. Second, establishing connections between users and cybertwins may have to go through identification and verification process, and the two functions are realized by biometric readers, facial recognition devices or other implements equipped in edge nodes. Third, light servers deployed in edge nodes can support some task offloading. The top layer contains a scheduling center and clusters of multiple regional clouds. The scheduling center collects task information from multiple cybertwins and issues instructions on how to offload tasks and how to allocate resources. Regional clouds are equipped with big databases and powerful servers. Entities in the edge layer and the cloud layer comprise a set of servers $S = \{1, 2, ..., M\}$.

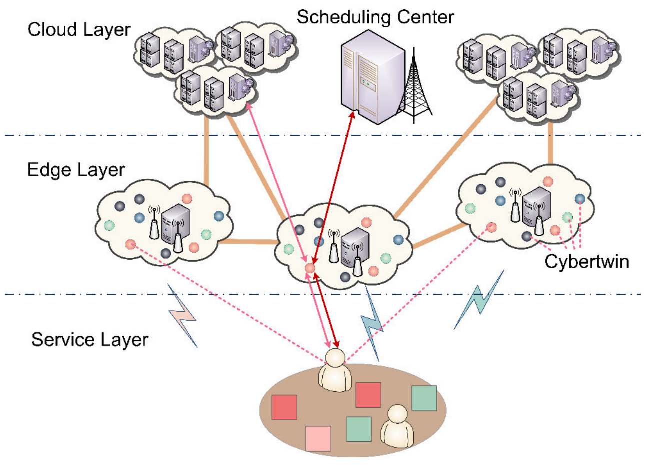


Fig. 1. Network model.

Since the wireless communication coverage of some edge nodes are overlapped, several cybertwins are available to one user. When a user broadcasts its offloading request to nearby cybertwins, those cybertwins reply with current channel state information (CSI) and potentially, verification requests. The user automatically selects one cybertwin according to best transmission rate or other preference settings. If the user is verified, the cybertwin will forward the task information to the scheduling center. Based on instructions from the scheduling center, the user offloads its task to the designated server and receives task results through its cybertwin. Advantages of this system are as follows:

- Cloud Resource Partition: Different regional clouds store contents for disparate services and such partitioning allows flexible infrastructure deployment based on network throughput or QoS requirements. In addition, different clusters of regional clouds are resource centers for d-ifferent regions and hence we assume no direct wired connection between different clusters.
- Multiple but Restrained Resource Accesses: Some edge nodes are wired-connected due to vicinity or other requirements. Additionally, each edge node is connected to some but not all regional clouds, which ensures stable resource sharing within a certain area and avoids unnecessary trans-area content sharing. Nevertheless, a cybertwin has access to multiple resources.
- Cache-Offloading-Combined Mechanism: When servers in regional clouds or edge nodes have access to the data necessary for tasks from users, these users will be spared from uploading, namely the time and energy cost of uploading data.
- Enhanced Security: A cybertwin will require verification if there is a new connection request or a reconnection request after a certain time span. The length of the time span depends on users' category and trust level. By shunning unverified users, security is enhanced and more resource is reserved for valid tasks.

III. Problem Formulation

A task is featured by task size L, computing density D, ratio of input to output ϑ and maximum tolerable delay $T_{\max}$. By introducing D and ϑ, the system implicitly takes multiple task types into consideration. The offloading proportion of user n to server m is introduced to divide tasks into local computing part and offloading computing part, and denoted as $\alpha_{n,m}$. Offloading proportion and server selection comprise offloading decision, and server selection is given by

$$[o_{n,m}] = \begin{cases} 1, & \text{if } o_{n,m} = m \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

Therefore, local computing delay of task n is given by

$$T_n^{(l)} = \frac{(1 - \alpha_{n,m}\,[o_{n,m}])L_n D_n}{f_n^{(l)}} \tag{2}$$

where $f_n^{(l)}$ is computing capability of user n. Similarly, computing delay of task n in server m is denoted as

$$T_{n,m}^{(r)} = \frac{\alpha_{n,m}\,[o_{n,m}] L_n D_n}{f_{m,n}^{(r)}} \tag{3}$$

where $f_{m,n}^{(r)}$ is the computing resource that server m allocates to user n. Servers of this system are virtual-machine-enabled, so multiple tasks can be processed in parallel. It is assumed that resource in each server is averaged among all arriving tasks and queuing is not considered.

Since completely decoupled uplink and downlink are considered, there are uplink bandwith $W_n^{(u)}$, uplink SINR $\gamma_n^{(u)}$, downlink bandwidth $W_n^{(d)}$ and downlink SINR $\gamma_n^{(d)}$. Accordingly, the uplink rate $r_n^{(u)}$ and downlink rate $r_n^{(d)}$ between user n and its cybertwin are given by

$$r_n^{(u)} = W_n^{(u)} \log_2\left(1 + \gamma_n^{(u)}\right) \tag{4}$$

$$r_n^{(d)} = W_n^{(d)} \log_2\left(1 + \gamma_n^{(d)}\right) \tag{5}$$

Correspondingly, the uplink delay is denoted as

$$T_n^{(u)} = \frac{\eta_n \alpha_{n,m}\,[o_{n,m}] L_n}{r_n^{(u)}} \tag{6}$$

where $\eta_n = 0$ indicates that content required for this task is already cached and can be accessed by available servers. But unlike many researches that ignored the delay of results delivery, we consider the downlink delay:

$$T_n^{(d)} = \frac{\vartheta_n \alpha_{n,m}\,[o_{n,m}] L_n}{r_n^{(d)}} \tag{7}$$

Therefore, the total delay of task n is calculated by

$$T_n = \max\left(T_n^{(l)}, \quad T_n^{(u)} + T_{n,m}^{(r)} + T_n^{(d)}\right) \tag{8}$$

and the energy consumption of task n is given by

$$E_n = P_n^{(l)} T_n^{(l)} + P_n^{(u)} T_n^{(u)} + P_m^{(r)} T_{n,m}^{(r)} + P_n^{(d)} T_n^{(d)} \tag{9}$$

where $P_n^{(l)}, P_n^{(u)}$ and $P_n^{(d)}$ represent the local computing power, uplink transmission power, downlink transmission power of user n, respectively; and $P_m^{(r)}$ denotes the computing power of server m. $P_n^{(l)}$ is given by

$$P_n^{(l)} = \kappa \cdot \left(f_n^{(l)}\right)^3 \tag{10}$$

where κ is a capacitance coefficient and set as 10^{-28} according to [8].

A trade-off between delay and energy consumption is a rational objective to optimize. However, unlike researches that optimized delay or energy consumption directly, we target at delay efficiency and energy consumption efficiency, because task size and computing density may vary over a large range. Moreover, we consider different types of upper bound in two efficiency terms. The maximum tolerable delay $T_{\max}$ is used in delay efficiency, and the energy consumption of complete local computing is employed in energy consumption efficiency. The utility function is defined as

$$\begin{gathered} \mathcal{U} = \frac{1}{N} \sum_{n \in U, m \in S} \omega \frac{\rho_n (T_{\max} - T_n)}{T_{\max}} + (1-\omega) \frac{\xi_n (E_{AL,n} - E_n)}{E_{AL,n}} \\ \rho_n = 0 \quad \text{if} \quad T_n > T_{\max} \\ \xi_n = 0 \quad \text{if} \quad E_n > E_{AL,n} \end{gathered} \tag{11}$$

where ω signifies a trade-off coefficient between delay cost and energy cost; $E_{AL,n}$ represents the energy consumption when task n is computed completely in local; ρ_n and ξ_n are timeout indicator and excessive energy consumption indicator of task n, respectively. The problem is thus formulated as:

$$\max_{\alpha_{n,m},\ o_{n,m}} \mathcal{U} \tag{12}$$

IV. DRL-based task offloading algorithms

A. The Proposed DRL-based Approach

To solve the formulated problem (12), we propose a dynamic task offloading algorithm based on DRL framework. RL is to maximize cumulative rewards through continual interactions between agent and environment, and the interacting process includes five basic components: state, action, reward, return and Q-function.

At time step t, the state of the environment, denoted by s_t, includes task size $L(t)$, computing density $D(t)$, ratio of input to output $\vartheta(t)$, local computing capacity $f^{(l)}(t)$, remote computing resource $f^{(r)}(t)$, uplink rate $r^{(u)}(t)$, uplink transmission power $P^{(u)}(t)$, downlink rate $r^{(d)}(t)$, downlink transmission power $P^{(d)}(t)$, local computing power $P^{(l)}(t)$, computing power of remote servers $P^{(r)}(t)$, cache indicator $\eta(t)$, and a list of available servers $A(t)$.

The action a_t corresponds to $o_{n,m}(t)$ and $\alpha_{n,m}(t)$, namely which server and at what proportion to offload.

After performing the action a_t, the environment transfers to a new state s_{t+1} and returns an immediate reward $r(s_t, a_t)$. We define this reward as the utility $r(s_t, a_t) = \mathcal{U}(t)$ and the return as the discounted sum of rewards $R_t = \sum_{i=t}^{T} \zeta^{i-t} r(s_t, a_t)$, where ζ is a discount factor that determines the importance of future rewards.

In RL, Q-function or value function is the expected return when performing action a according to policy π for the given state s, which is $Q^{\pi}(s,a) = \mathbb{E}[R_t|s,a]$. Many RL approaches learns based on Bellman equation which can be transformed to $Q^{\mu}(s_t,a_t) = \mathbb{E}[r(s_t,a_t) + \zeta Q^{\mu}(s_{t+1},a_{t+1})]$ in TD3 [12]. Specifically, μ is a function approximator parameterized by θ. So the value function Q_θ is learned by minimizing the loss $L(\theta^Q) = \mathbb{E}[(Q(s_t,a_t|\theta^Q) - y_t)^2]$, where $y(t) = r(s_t,a_t) + \zeta Q(s_{t+1}, \mu(s_{t+1})|\theta^Q)$. The complete process of TD3 specified in Algorithm 1.

B. The Modified Graph Attention Network

Many existing DRL-based task offloading algorithms choose DNN or CNN as their network structures. The underlying data representation of DNN is sequence and that of CNN is grid-like structure. However, the data in task offloading scenario presents graph structure and using graph neural network may achieve further performance boost.

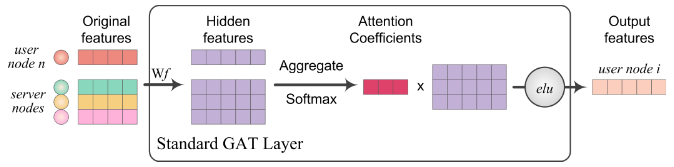
A task offloading system can be modelled by a graph featuring nodes (users and servers) and edges (relations between each user and each server). Specifically, each user is represented by a feature vector while each server is represented by another feature vector. Their relations, namely whether one server is available to one user, are expressed by an adjacency matrix $J = (j_{nm}) \in \mathbb{R}^{N \times M}$. For example, $j_{nm} = 1$ indicates that server m is available to user n, or in other words, server m is a neighbor of user n. The aforementioned state s_t is thus re-categorized into three types:

- user features: $\{L(t), D(t), \vartheta(t), f^{(l)}(t), P^{(l)}(t), r^{(u)}(t), r^{(d)}(t), P^{(u)}(t)\}$;
- server features: $\{f^{(r)}(t), P^{(r)}(t), P^{(d)}(t)\}$;
- relations between users and servers: $J(t)$

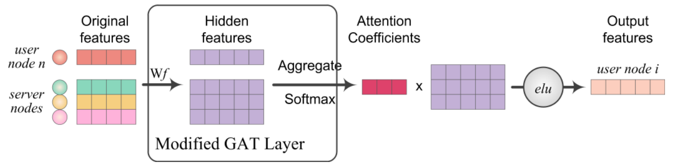
Among various types of graph neural networks, we choose GAT for its simple implementation and leave other sophisticated ones for future exploration. The key idea of a standard GAT layer is extracting implicit features of nodes to form the corresponding representations. The standard GAT layer is portrayed as Fig. 2(a) and it includes three parts:

- Map the original features of every node (user and server) into a new feature space and obtain hidden features;
- Aggregate the hidden features of user n with that of server m, where $m \in \mathcal{N}_n$ is a neighbor of user n; feed the aggregation into a softmax function to obtain an attention coefficient that indicates the importance of server m to user n;
- Linearly combine these attention coefficients with the corresponding hidden features to get output features.

Since our objective is to select the most important neighbor rather than extract senior representation, we remove the linear combination of the third part. The new layer is named as



(a) Standard GAT Layer



(b) Modified GAT Layer

Fig. 2. Standard GAT layer and modified GAT layer.

modified GAT layer and depicted in Fig. 2(b). Be noted that multi-head attention mechanism used in [11] is also implemented in our approach. Further, unlike the standard GAT network that consists of two standard GAT layers, our modified GAT network is constructed by one standard GAT layer and one modified GAT layer. The actor network and the target actor network in our algorithm are modified GAT networks. The critic networks and target critic networks are also based on the modified GAT network, but each of them concatenates one more fully-connected layer.

Algorithm 1: GAT-TD3-based Task Offloading Algorithm

```
Randomly initialize critic networks $Q_{\theta_1}, Q_{\theta_2}$ and actor network $\pi_\phi$
Initialize target critic networks $Q_{\theta'_1} \leftarrow Q_{\theta_1}, Q_{\theta'_2} \leftarrow Q_{\theta_2}$
Initialize target actor network $\pi_{\phi'} \leftarrow \pi_\phi$
for $t \in \mathcal{T}$ do
    For $s_t$, select an action $a_t \leftarrow \pi_\phi(s_t) + \epsilon_1, \epsilon_1 \sim \mathcal{N}(0,\sigma)$
    Observe reward $r_t$ and new state $s_{t+1}$
    Store transition $[s_t, a_t, r_t, s_{t+1}]$ in replay buffer $\mathcal{B}$
    Sample $K$ transitions $[s_t, a_t, r_t, s_{t+1}]$ from $\mathcal{B}$
    $a'_{t+1} \leftarrow \pi_{\phi'}(s_{t+1}) + \epsilon_2, \epsilon_2 \sim clip(\mathcal{N}(0,\sigma'), 0, c)$
    $y \leftarrow r_t + \gamma \min_{z=1,2} Q_{\theta'_z}(s_{t+1}, a'_{t+1})$
    Update critics $\theta_z \leftarrow \arg\min_{\theta'_z} \frac{1}{K} \sum (y - Q_{\theta_z}(s_t,a_t))^2$
    if $t$ mod $d$ then
        Update $\phi$ by deterministic policy gradient:
        $\nabla_\phi J(\phi) = \frac{1}{K} \sum \nabla_{a_t} Q_{\theta_1}(s_t,a_t)\mid_{a_t=\pi_\phi(s_t)} \nabla_\phi \pi_\phi(s_t)$
        Update target networks:
        $\theta'_z \leftarrow \tau\theta_z + (1-\tau)\theta'_z$
        $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$
    end if
end for
```

V. Simulation Results

A. Parameter Settings

In the proposed cybertwin-based architecture, we consider a system deployed with 2 cloud servers and 3 edge servers, and assume the system can support 50 users at maximum. The uplink rate and downlink rate (Gbps) are set as an uniform distribution between 0 and 1. User computing capacity (GHz) follows a uniform distribution between 0.8 and 1. Considering that resource in servers may be consumed by other control systems, computing resource (GHz) of edge servers is set between 0 and 10, and that of cloud servers 10 to 50. To mimic the scenario where users have to offload sometimes, data size (bit) and computing density (cycle/bit) of each task are both set as a random variable between 500 to 1500. The ratio ϑ is set as a random variable between 0 and 0.5 to provide occasional situation where downlink delay is non-negligible. The uplink and downlink transmission power (W) are set as 0.5 and 5, respectively. The computing power (W) of edge servers and cloud servers are set as 1 and 2, respectively. The trade-off coefficient ω is set as 0.5. $T_{\max}$ is set as 1 ms.

In the modified GAT network, the standard GAT layer involves 8 hidden features and 8 heads, and the modified GAT layer involves 5 hidden features and 1 head. To evaluate how GAT fits task offloading data, the proposed algorithm is compared with DNN-TD3-based algorithm and CNN-TD3-based algorithm. The DNN consists of two 64-neuron hidden layers as the one in [13]. The CNN contains three convolutional layers and each layer is composed by 32 filters of 1×16-size. Both CNN and DNN adopt ELU as activation function of hidden layers. These three models are chosen because they respectively outperformed their counterparts, namely other GATs, DNNs and CNNs, in our preliminary experiment. Other hyperparameters are listed in Table I:

TABLE I
THE LIST OF HYPERPARANMETERS

Hyperparameters	*GAT*	*DNN*	*CNN*
Critic Learning Rate	$5 \cdot 10^{-4}$	$5 \cdot 10^{-5}$	$5 \cdot 10^{-5}$
Actor Learning Rate	$5 \cdot 10^{-4}$	$5 \cdot 10^{-5}$	$5 \cdot 10^{-5}$
Optimizer	*Adam*		
Target Update Rate (τ)	$5 \cdot 10^{-3}$		
Batch Size	*128*		
Discount Factor	*0.99*		
Exploration Policy (ϵ_1)	$\mathcal{N}(0, 0.1)$		
Smooth Noise (ϵ_2)	*clip*$(\mathcal{N}(0, 0.2), 0, 0.2)$		

B. Results and Discussion

As shown in Fig. 3, TD3 with GAT converges faster and better compared to TD3 with CNN or DNN, because graph convolution presents better feature extraction capability on graph data. This also proves that it is viable to improve task offloading method from the perspective of data structure.

Be noted that average episodic reward in the following experiments is obtained by going through 100 time steps and averaged over 1000 episodes. Besides, some abbreviations are used in legends: cache-considered is CC for short and non-cache-considered NCC; Both All_Off and Ran_Off indicate that each user randomly chooses one server to offload, but

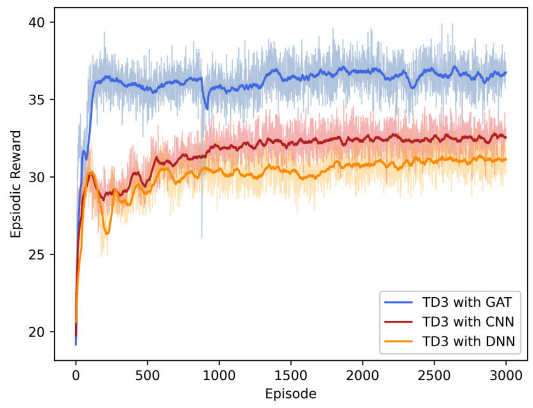


Fig. 3. Average utility over training episodes.

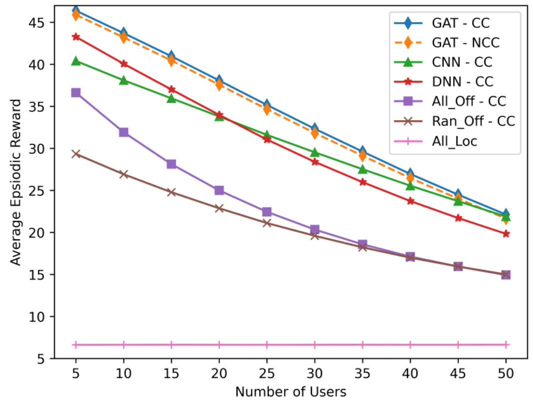


Fig. 4. Average utility over users.

the former offloads a whole task and the latter offloads at a random proportion.

Fig. 4 shows how different schemes perform over different system scale. TD3 with GAT (blue and orange) outperforms other methods. It is worth noted that GAT-CC is a notch above GAT-NCC, because the cache-considered mechanism spares some users from the delay and energy consumption of uploading data. Since not all data required is cached in servers, the gap looks minor.

Fig. 5 and Fig. 6 demonstrate how workload affects the system performance. As task size or computing density grows, the average utility of all methods decreases. Still, TD3 with GAT is the best among them.

In Fig. 7 and Fig. 8, deteriorating transmission rate does compromise the system performance because users struggle to upload data or download results. Nevertheless, TD3 with GAT achieves better performance compared to other methods. Moreover, the cache-considered mechanism avoids uploading when required data is already in servers, making the system less vulnerable to worsening uplink.

VI. Conclusion

In this paper, we have designed an end-edge-cloud orchestration system on top of a cybertwin-based framework to support simultaneous multi-task offloading in future network.

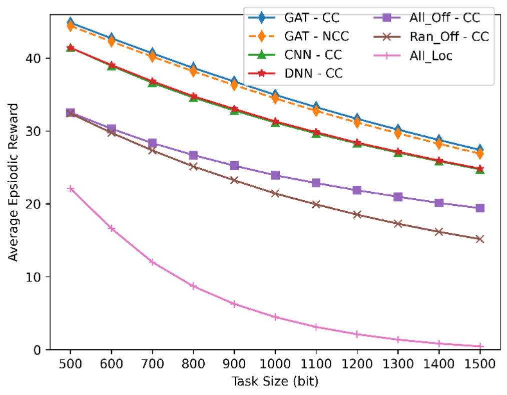


Fig. 5. Average utility over task size.

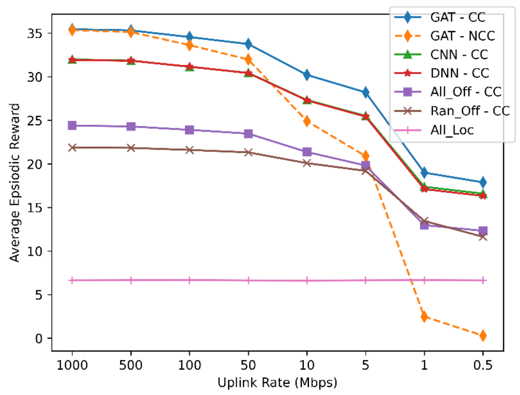


Fig. 7. Average utility over uplink rate.

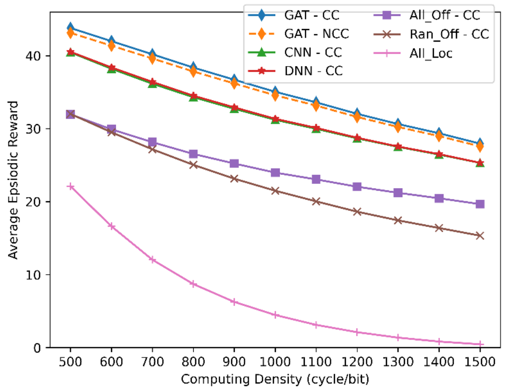


Fig. 6. Average utility over computing density.

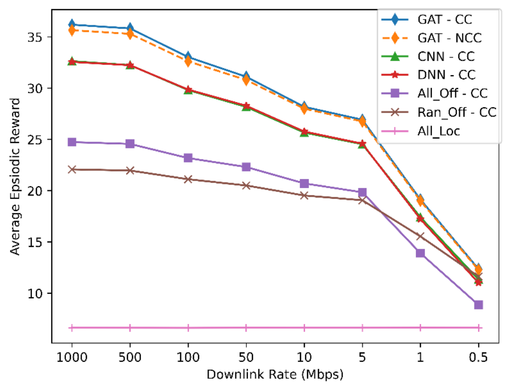


Fig. 8. Average utility over downlink rate.

To maximize the trade-off utility between delay efficiency and energy consumption efficiency, a novel method that incorporates the modified GAT network into the TD3 framework has been proposed. Finally, simulation results show that the proposed method outperforms DNN-TD3- and CNN-TD3-based algorithms as well as other baseline strategies.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China under Grant 62071306, in part by the Mobility Program for Taiwan Young Scientists under Grant RW2019TW001, and in part by the Shenzhen Science and Technology Program under Grants JCYJ20200109113601723, JSGG20210420091805014 and GJHZ20180418190529516.

References

[1] Samsung Research, “The Next Hypper–Connected Experience for All,” 2020. [Online]. Available: https://cdn.codeground.org/nsr/downloads/researchareas/20201201_6G_Vision_web.pdf
[2] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu and W. Zhuang, “Learning-Based Computation Offloading for IoT Devices With Energy Harvesting,” *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1930-1941, Feb. 2019.
[3] H. Ke, J. Wang, L. Deng, Y. Ge and H. Wang, “Deep Reinforcement Learning-Based Adaptive Computation Offloading for MEC in Heterogeneous Vehicular Networks,” *IEEE Transactions on Vehicular Technology*, vol. 69, no. 7, pp. 7916-7929, July 2020.
[4] J. Ren, G. Yu, Y. He and G. Y. Li, “Collaborative Cloud and Edge Computing for Latency Minimization,” *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 5031-5044, May 2019.
[5] U. of OULU, Key Drivers and Research Challenges for 6G Ubiquitous Wireless Intelligence, 6G white paper, 2015.
[6] Q. Yu, J. Ren, H. Zhou and W. Zhang, “A Cybertwin based Network Architecture for 6G,” in *Proc. 2nd 6G Wireless Summit (6G SUMMIT)*, Levi, Finland, pp. 1-5, 2020.
[7] H. Huang, M. Lin, L. T. Yang and Q. Zhang, “Autonomous Power Management With Double-Q Reinforcement Learning Method,” *IEEE Transactions on Industrial Informatics*, vol. 16, no. 3, pp. 1938-1946, March 2020.
[8] L. Xiao, X. Lu, T. Xu, X. Wan, W. Ji and Y. Zhang, “Reinforcement Learning-Based Mobile Offloading for Edge Computing Against Jamming and Interference,” *IEEE Transactions on Communications*, vol. 68, no. 10, pp. 6114-6126, Oct. 2020.
[9] H. Ke, J. Wang, L. Deng, Y. Ge and H. Wang, “Deep Reinforcement Learning-Based Adaptive Computation Offloading for MEC in Heterogeneous Vehicular Networks,” *IEEE Transactions on Vehicular Technology*, vol. 69, no. 7, pp. 7916-7929, July 2020.
[10] Q. Yu, J. Ren, Y. Fu, Y. Li and W. Zhang, “Cybertwin: An Origin of Next Generation Network Architecture,” *IEEE Wireless Communications*, vol. 26, no. 6, pp. 111-117, December 2019.
[11] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, “Graph Attention Networks,” in *Proc. International Conference on Learning Representations (ICLR 2018)*, pp. 1-12, April 30-May 3, 2018, Canada.
[12] S. Fujimoto, H. van Hoof, and D. Meger, “Addressing Function Approximation Error in Actor-Critic Methods,” in *Proc. International Conference on Machine Learning (ICML 2018)*, pp. 1-15, Jul. 10-15, 2018, Sweeden.
[13] Q. Qi *et al.*, “Knowledge-Driven Service Offloading Decision for Vehicular Edge Computing: A Deep Reinforcement Learning Approach,” *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 4192-4203, May 2019.