applied sciences

MDPI

Article

Intelligent Computation Offloading Based on Digital Twin-Enabled 6G Industrial IoT

Jingjing Wu * and Ruiyong Zuo

School of Computer Science and Engineering, Northeastern University, Shenyang 110819, China
* Correspondence: wujingjing@cse.neu.edu.cn

Abstract: Digital twin (DT) technology, which can provide larger and more accurate amounts of data, combined with the additional computility brought by virtual environments, can support more complex connected industrial applications. Simultaneously, the development and maturity of 6G technology has driven the development of industrial manufacturing and greatly improved the operational efficiency of the industrial internet of things (IIoT). Nevertheless, massive data, heterogeneous IoT device attributes, and the deterministic and bounded latency for delay sensitive applications are major barriers to improving the quality of services (QoS) in the IIoT. In this article, we first construct a new DT-enabled network architecture and computation offloading delay model in the IIoT. Then, the computation offloading problem is formulated with the goal of minimizing the overall task completion delay and achieving resource allocation. Since the formulation is a joint optimization problem, we use deep reinforcement learning (DRL) to solve the original problem, which can be described by a Markov decision process (MDP). Numerical results show that our proposed scheme is able to improve the task success rate and reduce the task processing end-to-end delay compared to the benchmark schemes.

Keywords: 6G industrial internet of things (IIoT); digital twin; edge computing; computation offloading; deep reinforcement learning (DRL)

Citation: Wu, J.; Zuo, R. Intelligent Computation Offloading Based on Digital Twin-Enabled 6G Industrial IoT. *Appl. Sci.* **2024**, *14*, 1035. https://doi.org/10.3390/app14031035

Academic Editor: Nikos D. Lagaros

Received: 24 December 2023
Revised: 23 January 2024
Accepted: 24 January 2024
Published: 25 January 2024

1. Introduction

Following the initial commercial implementation of 5G networks, industry and academic organizations have carried out their work to the architecture, requirements, and applications of 6G mobile networks. The international telecommunication union (ITU) released a white paper on the 6G [1] vision and candidate technologies in June 2021. As the next generation communication technology, the direction of 6G technology research and development has become more and more clear [2]. The industrial internet of things (IIoT) is an intelligent manufacturing model that integrates the sensors, networking technologies, and data analytics into the industrial production process. The rapid development of artificial intelligence (AI) technology has brought an explosive growth of industrial data, and the large amount of data generated during the industrial production process is a challenge for the IIoT [3–5]. How to effectively collect, analyze, and process these data, and extract the guiding recommendations for industrial production from them is the core of the design. Realizing such rigorous and diverse service requirements requires the design of a special-purpose IIoT network that incorporates a large number of specialized functions and technical support. This is exactly what 6G technology is trying to do today [6,7]. This innovative architecture of the 6G IIoT network compensates for the shortcomings of previous wireless network technology applied to industrial field environments, providing higher security, reliability, and real-time performance.

However, due to the limited computing resources and processing capabilities of IIoT devices, it presents serious challenges for running computationally intensive applications locally. In order to meet the ever-increasing computing demands, mobile edge computing has

been proposed as a promising approach that deploys cloud-like computing resources at the edge of the network to provide real-time computing services to nearby IoT devices [8–10]. The system can offload whole or part of the industrial computing tasks generated on the IIoT devices to the edge servers, utilizing superior computing resources to accomplish the tasks [11]. Computation offloading is a promising approach, but it faces a number of challenges. Firstly, the computing tasks generated by IIoT devices are heterogeneous, such as the amount of data, task priority, required resources, and so on [12]. Secondly, the computation offloading decision needs to sense the network condition in real time and dynamically select the appropriate edge server [13]. Finally, how to optimize the offloading decision online to maximize the system performance is also a problem to be solved. The emergence of digital twin (DT) technology can well meet these challenges.

Terminology for digital twins was first presented by Michael Grieves [14] of the University of Michigan as the "Information Mirroring Model", which later evolved into the term "digital twin". Digital twins are virtual representations of the state of a physical entity or system by collecting data from physical entities through sensors and digitizing them to create a highly simulated virtual model [15–18]. Digital twins are theories and technologies with universal applicability, which can be applied to many fields, especially in system design, product manufacturing, data analysis, and engineering construction, etc. [19,20]. In view of the above characteristics, a number of studies combining DT and edge computing have recently emerged. Ref. [21] focused on minimizing the computational offload latency of DT wireless edge networks in IIoT environments through ultra-reliable and low-latency communication links. Ref. [22] proposed a mobile edge computing framework based on network digital twins that would enable intelligent resource management. Ref. [23] studied the digital twin service caching and request routing problem along with fairness awareness.

In fact, the above solutions for DT combined with edge computing are still in their infancy and have a lot of room for improvement. First, there is still relatively little research on using DT for the prediction and optimization of offloading decisions. Second, the approach adopted generally assumes that each IIoT device performs a single computing task, regardless of the randomness of task generation. As a new research method to solve time-varying problems, deep reinforcement learning (DRL) can be used to solve dynamic computation offloading and resource allocation problems. DRL can deal with the problem of the continuous action space and learn more complex offloading strategies, which cannot be handled by traditional algorithms. Inspired by the above discussion, this article considers a DT-assisted edge computing network consisting of multiple mobile IIoT devices. Computation offloading consists of three steps, offloading, processing, and feedback. Offloading refers to the use of wireless transmission technology to offload tasks to edge servers (deployed near the base station, not fixed) via mobile access points (APs). Processing requires that the edge servers have sufficient computational resources. Feedback is to return the processing results to the physical device through the AP after processing by the edge server. Generally, the returned data is much smaller than the offloaded data, because the offloaded data is the entire data collected by the physical device, whereas the returned data may only be an instruction. It may not be the same AP that returns the results since the physical devices are movable. The main innovations of this article are summarized as follows:

1. We propose an integrated digital twin and IIoT architecture to model the network systems and all physical devices in the IIoT. The architecture enables more efficient and optimized network operation.
2. We explore the impact of the heterogeneity of physical devices and the execution time of different tasks on offloading decisions. A solution based on deep deterministic policy gradient (DDPG) is employed to realize effective computation offloading. The solution focuses more on policy exploration than traditional algorithms to improve the adaptability of offloading algorithms and reduce the communication cost.
3. We have conducted extensive experiments on the proposed learning algorithm. The results show that by considering various delay constraints, more tasks can be better

executed, and unnecessary contention and the wasted resources phenomenon can be greatly reduced.

The rest of this article is organized as follows. The relevant research status is reviewed in Section 2. The problem is then described in Section 3. The formulations used by the MDP to describe the proposed problem are listed in Section 4, and the associated learning algorithm is also given in this section. Subsequently, simulation results are tested and analyzed in Section 5. Finally, we summarize the paper in Section 6.

2. Related Work

In this section, we review the literature on computation offloading with DT-enabled architecture using cloud-edge collaboration or edge servers. In addition, we present some research on computation offloading using AI techniques. In Table 1, we compare existing studies to further discuss them from different perspectives.

Table 1. Comparison of existing studies.

Reference	Offloading Position	Algorithm	Optimization Objective
[24]	Cloud-edge-client	DRL	Minimize latency performance
[25]	Cloud-edge-client	FL	Minimize the system cost
[26]	Cloud-edge-client	C^3-FLOW	Minimize communication cost
[27]	Cloud-edge	MA-DATD3	Minimize delay and energy
[28]	Cloud-edge-client	DRL	Minimize execution delay
[29]	Edge-client	SCA	Minimize the average delay
[30]	Edge-client	two-stage	Maximize total profits
[31]	Edge	SCA	Minimize latency
[32]	Edge	UCB	Minimize delay
[33]	Edge-client	non-convex optimization	Minimize the average delay
[34]	Edge	A3C	Minimize system energy cost
[35]	Edge	MINLP	Minimize the system latency
[36]	Edge	GatMARL	Maximize QoS
[37]	Edge	AAC	Minimize long term energy
[38]	Edge	DDQN	Minimize energy consumption
[39]	Edge-client	DRL	Minimize long term income

2.1. DT-Enabled Computation Offloading with Cloud

In recent years, DT can well replicate the network topology and the physical entities within it, which has been extended to industrial manufacturing, smart factories, and the Internet of Things. Therefore, given the technical advantages of DT, offloading locally-heavy computational tasks to DT-enabled network architectures has been widely studied. The authors in [24] proposed for the first time a digital twin dual empowered system that included end users, base stations, and cloud servers, and utilized transfer learning to solve the edge association problem. In [25], the authors presented a privacy-enhancing federated learning framework and developed DT-assisted multi-intelligent DRL-based resource scheduling for federated client association and channel allocation. The authors in [26] proposed a cloud edge device collaboration, reliable, and communication efficient DT for low carbon electrical device management. DT-empowered satellite-terrestrial cooperative edge computing networks were investigated in [27], in which computational tasks from terrestrial users could be partially offloaded to the associated base station edge servers or the associated satellite edge servers. In order to satisfy the dynamic service requirements, Ref. [28] proposed a deep reinforcement learning-based offloading mechanism for cloud edge collaborative mobile computing.

2.2. DT-Enabled Computation Offloading with Edge

By using the base stations closer to the physical entity as the edge servers, the problem of too wide coverage and heavy transmission burden of cloud computing can be alleviated. Edge computing usually has the advantage of lower costs of transmission, computation, and storage. In [29], the offloading problem was transformed into an equivalent convex problem with tractable solutions, in which users offloaded their tasks to edge servers using grant-free random access. The authors of [30] developed a two-stage incentive mechanism to encourage mobile devices to offload more computation tasks to edge servers when making optimal computational offloading and resource allocation decisions. In [31], a digital twin-assisted edge computing framework for fair perception delay minimization was presented and a low-complexity iterative algorithm was developed using successive convex approximation. Considering the mapping deviation between DTs and actual network states, Ref. [32] formulated the computational offloading problem as an online matching problem in an uncertain bipartite graph. By optimizing the subchannels, Ref. [33] aimed to minimize the total task delay, while accomplishing the interconnection of all IIoT devices, edge correlation, computational capacity allocation, and transmission power allocation.

2.3. Edge Computation Offloading Supported by AI

Integrating artificial intelligence into DT-enabled edge computing and deploying it on edge servers can better optimize offloading policies. The authors of [34] considered service caching and task dependencies and proposed a DT-based mobile edge computing architecture that can support mobile users in offloading dependency-aware tasks. In [35], the authors satisfied the latency tolerance for latency-sensitive tasks by considering both computation offloading and service caching. The authors in [36] proposed an algorithm for learning optimal task offloading and service caching policies with a multi-intelligence reinforcement approach based on graph attention. The authors in [37] solved the computation offloading and resource allocation problems by using the AAC algorithm. They used Liapunov optimization techniques to equivalently transform the original problem into a deterministic per time slot problem. The authors in [38] effectively solved the problem of intelligent computation offloading for UAVs. A new approach was proposed in [39] that allowed distributed mobile edge servers to collaborate on peer-to-peer offloading tasks to balance compute workloads on edge servers.

By analyzing these previous works, we can summarize the relevant research as follows. First, a DT-enabled network architecture can provide a solution for some risky field environments. Second, an AI algorithm can effectively solve the computation offloading problem in a complex dynamic environment. This is exactly what we will address next. In addition, we consider for the first time the heterogeneity of tasks generated by IIoT devices and how to optimize offloading strategies for such tasks.

3. System Model

3.1. DT-Enabled Network Model

The proposed digital twin-enabled network architecture is shown in Figure 1, which is composed of three main elements:

1. The bottom physical device network includes client devices in the IIoT, such as smart machines, sensors, vehicles, and IIoT devices. It also contains devices such as switches, access points, and databases that assist communication between devices. Sensors collect data from the industrial field environment and connect to the nearest AP according to the network topology. Cells led by AP are scattered throughout the whole work region, operated for monitoring and control among the machines and the robots.
2. The edge server element acts as a bridge between the network of physical devices and the digital twin. Digital twin networks are built and maintained with the help of edge servers. They are also the key to achieving virtual–physical interactions. The edge server element connects the network service application and the physical device

network through standardized interfaces, completes the real-time information collection and control of the physical network, and provides timely diagnosis and analysis. We can complete the digital modeling of the factory communication environment before the transmission of data, and then simulate the transmission in the virtual digital space to evaluate the frequency allocation and channel tolerance. We can also import additional traffic data to evaluate whether the transmission scheme can meet the requirements of the service.

3. The digital twin network can predict the optimal transmission channel through continuous acquisition and intelligent analysis of field data. A digital twin system can be roughly divided into three modules according to the functions it can achieve.

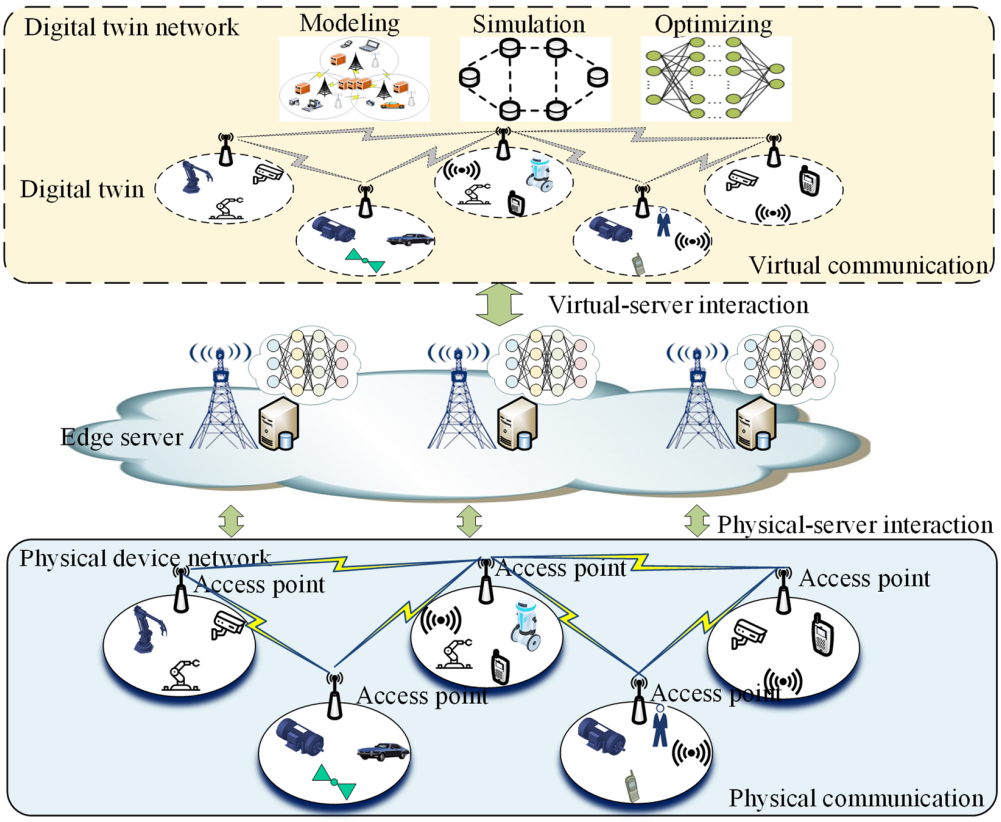


Figure 1. Digital twin-enabled network architecture.

The modeling function module should precisely digitize physical space and realize the virtual–real interaction between the physical entity and the digital model through the edge gateway. At this stage, the transmission of data does not necessarily need to be completely real-time. Data can be collected locally and transmitted periodically in a short period. The core technology of this module is replication modeling and IoT device sensing technology. Through sensing, mapping, and process modeling, the system can recognize the physical objects, complete the digitization of physical objects, and build the corresponding mechanism model.

In the simulation functional module, the system can create a virtual replica of the physical entity according to the actual application requirements. The DT analyzes and understands the data involved, and tests, alerts, and adjusts problems that have occurred or are about to occur. This function can realize the status tracking, analysis, and problem diagnosis of the physical space. This module focuses on building a global operational model based on data from the digital twin without transferring the raw local data.

The optimizing function module combines the results of the model operation with the dynamic system operation for self-learning and updating. This module predicts, simulates,

and debugs potentially undiscovered and future new modes of physical space in digital space based on known modes of physical space operation. After establishing the prediction of future development, the digital twin will present the prediction content in the digital space in a way that can be understood and perceived. This module has the function of active learning. Based on the learned existing knowledge, the system deduces unknown knowledge. The core technologies involved are concentrated in the fields of machine learning, image recognition, big data analysis, etc.

The digital twin-enabled network architecture brings obvious efficiency gains and cost reductions to industrial manufacturing. With the support of 6G technology, closer connections can be established between edge servers and physical terminals. This also means that more data will be collected and aggregated.

3.2. Computation Offloading Model

The first step in creating a DT is to develop an accurate and comprehensive virtual model from the collected field data. After the DT is created, the DT model can be updated in real time with the data collected by the industrial equipment through the edge servers, and the decisions in the DT can be mapped to the physical entity.

A directed graph $G(\mathcal{N}, \mathcal{L})$ is used to represent the physical device network, where $\mathcal{N} = \{1, 2, \cdots, n, \cdots, N\}$ is the set of physical devices (including smart machines, switches or access points), and $\mathcal{L}$ is the set of links. Accordingly, the virtual topology in the DT model can be represented as $G^{DT}(\mathcal{N}^{DT}, \mathcal{L}^{DT})$. Let $\mathcal{F}$ be the set of tasks in the network, where $\mathcal{F} = \{1, 2, \cdots, f, \cdots, F\}$ and F is the total number of tasks. A task $f \in \mathcal{F}$ is defined as a quadruple $f \triangleq \left\{d_f, \omega_f, \tau_f, n\right\}$, where d_f represents the computational requirement to process the task, ω_f represents the transmitted data size of task f, τ_f is the maximum delay bound of task f, $n \in \mathcal{N}$ is the location of the task. A DT model refers to a network consisting of DT nodes (i.e., virtualized physical nodes) where the packets forwarding delay in transmission and processing delay on edge servers are computable. The centralized computation offloading scheme provides deterministic and easily computable latency services. In the DT model, the delay caused by forwarding packets consists of the following parts:

1. Queuing delay, which is the waiting time in the output port queue due to the accumulation of packets generated by different applications to the same output port.
2. Transmission delay, which is the time consumed to put the packet on the wireless link and send it from the source to the destination.
3. Update delay, which refers to the delay caused by synchronizing heterogeneous physical devices when updating the global DT model.
4. Processing delay, which is the time consumption associated with completing a computation task and can be measured in terms of the number of CPU cycles.

In 6G scenarios, physical devices are mobile and may be in the coverage range of multiple APs. Computation offloading between physical devices and edge servers is realized through wireless channels. To simplify the research, we only consider the uplink transmissions without backhaul, and do not consider communication interference between devices. Some advanced industrial applications, such as digitalized operations and predictive maintenance, require interconnection and cooperation of IIoT devices. When these tasks are offloaded to the edge server, the local time will deviate from the global time due to the heterogeneity of the physical devices. In order to ensure coherent data processing on digital twins, they need offset compensation before processing. For an IIoT device, the appropriate edge server will be selected to offload the task based on a set of criteria, including its physical location, the computing capabilities of the edge server, and the channel state information. As illustrated in Figure 2, more communication resources can be allocated to device 2 to mitigate the imbalance in communication performance. Servers with more computing resources can also be allocated to the computationally-heavy device 3 to reduce its processing delay.

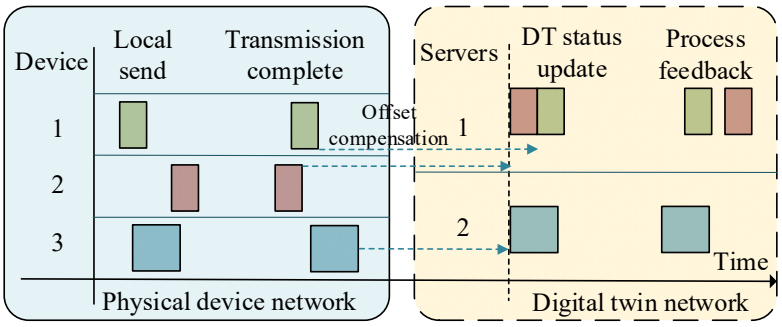


Figure 2. An illustration of computation offloading processes.

3.3. Problem Formulation

In this section, we first define a total end-to-end delay for a task and then describe it with mathematical formulations. Our goal is to minimize this total delay while ensuring the maximum delay bound of this task. The important symbols and variables used in this article are shown in Table 2. As described earlier, the end-to-end delay consists of four parts, a queuing delay, transmission delay, synchronization delay, and processing delay. The physical device generates running data and synchronizes it with the corresponding digital twin running on the server. For a physical device n, its digital twin DT_n is modeled in its nearby server, as shown in (1)

$$DT_n = \left\{ M_n, q_{n,f}, s_n, \Delta s_n \right\} \tag{1}$$

Table 2. Explanation of symbols and variables.

Symbol	Description
$G(\mathcal{N}, \mathcal{L})$, $G^{DT}(\mathcal{N}^{DT}, \mathcal{L}^{DT})$	Network topology
DT_n	The DT model for a physical device n
$M_n, q_{n,f}, s_n, \Delta s_n$	DT attribute element of n to process task f: DT model, QoS requirements, essential state, deviation
$\mathcal{F} = \{1, 2, \cdots, f, \cdots, F\}$	The set of tasks
d_f, ω_f, τ_f	Task attribute element of task f: computation requirement, transmitted data size, maximum delay bound
$\mathcal{N} = \{1, 2, \cdots, n, \cdots, N\}$	The set of nodes in the network
μ_n, λ_f	Queue parameter
$b^k, p_n^k, h_{n,i}^k, \sigma^2, r_{n,i}, \alpha$	Wireless communication parameters
$L_n^{res}, P_n^{res}, D_n^{res}, C_i^{res}$	Resource upper limits for various nodes
B	Environmental bandwidth in DT model
T_G	DT global clock
$t_{n,i,f}^{arr}$	Task arrival time
$t_{i,f}^{ser}$	Processing start time
$x_{i,f}, y_{i,f}^k$	Boolean variable
$D_{n,i,f}^{que}, D_{n,i,f}^{trans}, D_{n,i,f}^{update}, D_{n,i,f}^{pro}$	The composition of total end-to-end delay
$\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$	MDP attribute element: state, action, transition matrix, reward, discount factor
$c(t),\ p(t), b(t)$	Distribution of network resources at the current time
π	Deterministic policy
μ, μ'	Parameters of the Actor network
θ, θ'	Parameters of the Critic network
$Q_\theta(s_m, a_m)$	Maximum Q-value
ρ	A small step factor

The server collects operational data from the physical device and constructs the DT model M_n of the device n. $q_{n,f}$ represents the quality of service (QoS) requirements to process task f, s_n is the essential state data required to run digital twin applications, and Δs_n is the estimated deviation from data updating.

For simplicity, we assume that there is only one queue on the output port of one AP. We use the queuing theory to model the queuing delay as an M/M/1 queuing system. We define a binary variable $x_{i,f}$ to represent edge server $i \in \mathcal{N}$ that is selected to offload the computation task of f; $x_{i,f} = \{0, 1\}$ is equal to 1 if task f is offloaded to edge server i, otherwise, it equals 0. For a task f, its queuing delay is denoted as:

$$D_{n,i,f}^{que} = x_{i,f} \cdot \frac{1}{\mu_n - \lambda_f} \tag{2}$$

Here, the arrival time intervals of task f obeys a negative exponential distribution of with parameter λ_f, and the node's processing time for the task also has a negative exponential distribution with parameter μ_n.

The transmission delay is closely related to the uplink rate, which is affected by the amount of traffic and the transmission distance. The uplink transmission rate satisfies Shannon's theorem and is related to node transmission power, wireless interference, and environmental bandwidth. Here, we consider device-to-device communication based on the shortest distance broadcast channel. The wireless communication data rate between two devices can be expressed as

$$R_{n,i,f} = y_{i,f}^{k} \cdot b^{k} log_2(1 + \frac{p_n^k h_{n,i}^k r_{n,i}^{-\alpha}}{\sigma^2 + I}) \tag{3}$$

We define a binary variable $y_{i,f}^{k}$ to represent the channel allocation status; $y_{i,f}^{k} = \{0, 1\}$ is equal to 1 if channel k is assigned to edge server i to transmit task f, otherwise, it equals 0. b^k represents the bandwidth allocated to the k subchannel. The channel gain is denoted as $h_{n,i}^k$, which can be considered to be constant, and varies with each update. p_n^k is the transmitted power of physical device n and σ^2 means the power of additive Gaussian white noise. $r_{n,i}$ means the communication distance between the sender and the receiver. α is the path loss coefficient. $I = \sum_{n',i' \in \mathcal{N} \setminus \{n,i\}} p_{n'}^k h_{n',i'}^k r_{n',i'}^{-\alpha}$ is the communication interference between other physical devices and edge servers.

Thus, we can calculate the average transmission delay as

$$D_{n,i,f}^{trans} = \frac{\omega_f}{R_{n,i,f}} \tag{4}$$

When the task f arrives at the edge server where the task is to be offloaded, it has to wait for the DT to coordinate it. We consider a global time scale, as shown by the dotted lines in Figure 2, and this global clock is updated periodically with update intervals τ. Tasks waiting to be processed by the edge server enter the DT's processing queue in the latest update:

$$D_{n,i,f}^{update} = T_G(\tau) - t_{n,i,f}^{arr} + t_{i,f}^{ser} \tag{5}$$

where $T_G(\tau)$ is the global clock at time τ, $t_{n,i,f}^{arr}$ denotes the time when the task f generated from physical device n arrives at the receiving port of edge server i, and $t_{i,f}^{ser}$ represents the start time of task processing by server i. T_G is updated periodically, so choosing different edge servers brings different offset compensations and service queue lengths. Finally, the processing time of the task by the edge server is expressed as

$$D_{n,i,f}^{pro} = \frac{d_f}{C_i} \tag{6}$$

where C_i is the CPU cycle frequency of the edge server i.

Therefore, the total end-to-end delay of the task f generated by physical device n can be expressed as

$$D_{n,i,f}^{total} = D_{n,i,f}^{que} + D_{n,i,f}^{trans} + D_{n,i,f}^{update} + D_{n,i,f}^{pro} \tag{7}$$

Now, we intend to minimize the end-to-end delay by formalizing it as an optimization problem. The constraints of the optimization problem are shown in (9)–(18) according to the order of appearance of the variables.

$$\mathrm{P}: \ \text{minimize} \sum_{f=1}^{F} D_{n,i,f}^{total} \tag{8}$$

$$\text{s.t.} \ \sum_{i \in \mathcal{N}} x_{i,f} = 1, \ \forall f \in \mathcal{F} \tag{9}$$

$$\frac{\lambda_f}{\mu_n} < 1, \forall f \in \mathcal{F}, \forall n \in \mathcal{N} \tag{10}$$

$$\frac{\lambda_f}{\mu_n - \lambda_f} \leq L_n^{res}, \ \forall f \in \mathcal{F}, \forall n \in \mathcal{N} \tag{11}$$

$$0 \leq b^k \leq \frac{B}{|K|}, \ \forall k \tag{12}$$

$$0 \leq p_n^k \leq P_n^{res}, \forall n \in \mathcal{N}, \forall k \tag{13}$$

$$0 \leq d_f \leq D_n^{res}, \ \forall f \in \mathcal{F}, \forall n \in \mathcal{N} \tag{14}$$

$$T_G(\tau - 1) \leq t_{n,i,f}^{arr} \leq T_G(\tau), \ \forall f \in \mathcal{F}, \forall n, i \in \mathcal{N} \tag{15}$$

$$T_G(\tau) \leq t_{i,f}^{ser}, \ \forall f \in \mathcal{F}, \forall i \in \mathcal{N} \tag{16}$$

$$0 \leq C_i \leq C_i^{res}, \ \forall i \in \mathcal{N} \tag{17}$$

$$0 \leq D_{n,i,f}^{total} \leq \tau_f, \ \forall f \in \mathcal{F}, \forall n, i \in \mathcal{N} \tag{18}$$

Constraint (9) indicates that the computation task can be offloaded to only one edge server. Constraint (10) describes the service intensity of the physical node for task processing in unit time and ensures that it has a steady-state solution. Constraint (11) restricts the queue length on a device n to not exceed the upper limit of queue resources. Constraint (12) ensures that the allocated bandwidth is non-negative and cannot exceed the bandwidth utilization limit of the network, assuming that there are $|K|$ subchannels over the whole bandwidth B. Constraints (13) and (14) guarantee that the transmit power and computation data size are non-negative and no more than the limits of their generation nodes. Constraints (15) and (16) describe the time limit for a task to reach the edge server, which must arrive before the next update to be processed in this round. Constraint (17) guarantees that the computation capability of the edge server is non-negative and cannot exceed the upper bound. Constraint (18) guarantees each task must be processed within its maximum delay bound.

4. DRL-Driven Offloading Scheme Based on DT

According to the analysis of problem (8), the computation offloading problem is a joint optimization problem, which requires strict assumptions and constraints. The performance of traditional optimization approaches depends heavily on the selection of thresholds, which is a challenging task for multi-constraint problems. Because of its powerful intelligent decision-making ability, DRL has been widely used in the field of user association and environment interaction in recent years. Mathematically, we would normalize such an optimization problem as a Markov decision process (MDP). The DRL algorithm is used to find the solution effectively. In order to improve the flexibility of the system and solve the delay minimization problem, we propose a DT-enabled DDPG algorithm to find the optimal solution. The trained network model can be used for optimization and can handle dynamic system states.

4.1. Digital Twin Simulated MDP

In reinforcement learning, the DRL agent learns the current state from the environment and takes actions that lead the system to the next state. The environment gives the agent a reward based on the action. In order to solve the problem (8), the system first establishes

an MDP, i.e. $MDP = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, here $\gamma \in [0,1]$ is the discount factor. From Figure 3, the current state s_t is constructed by the digital twin and output to the DRL agent.

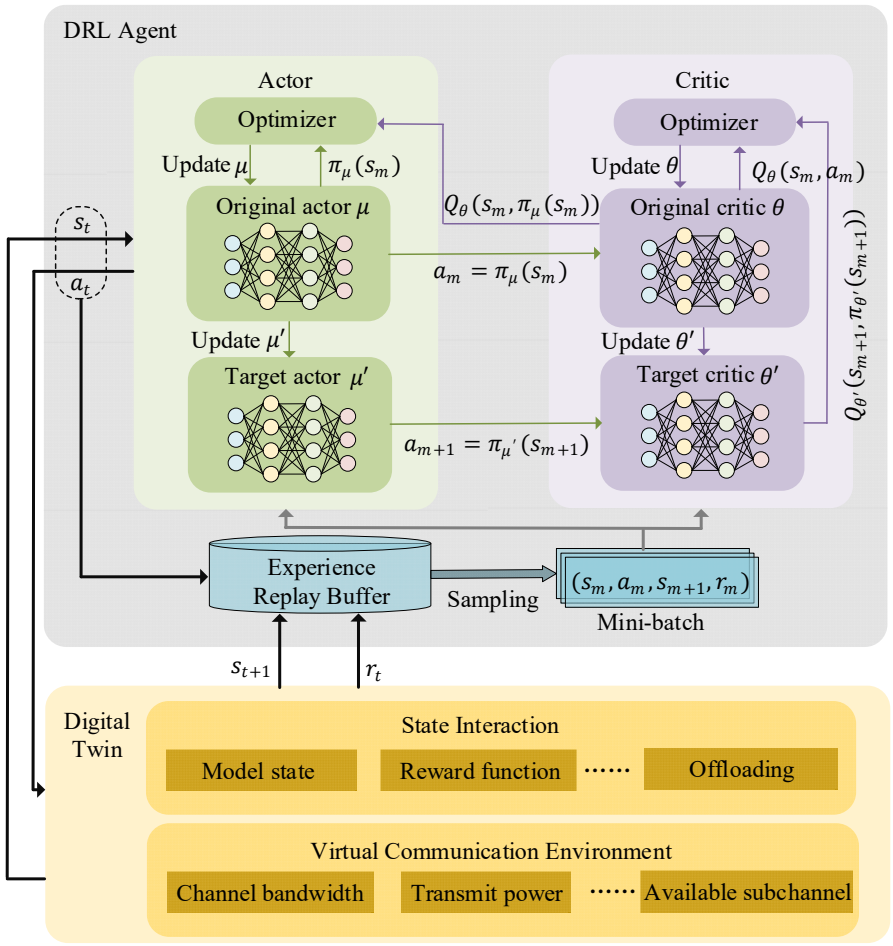


Figure 3. Structure of the DDPG agent.

The current state $s_t \in \mathcal{S}$ is expressed as

$$s_t = \left\{ \sum_{n=1}^{|N^{DT}|} DT_n(t), G^{DT}(t), B \right\} \tag{19}$$

In (19), $DT_n(t)$ represents the DT model of the corresponding physical device. $G^{DT}(t)$ represents the current state of the network, including the available resources of the nodes and links. B is the wireless channel bandwidth, which is a constant.

The action space describes the DRL agent's strategy for resource allocation. We define the action space as $a_t = \left\langle a_t^{off}, a_t^{rsrc} \right\rangle$. In the tuple, $a_t^{off} = \left\{ a_{i,f}^{off}(t) \in [0,1] \middle| i \in \mathcal{N}, f \in \mathcal{F} \right\}$ denotes whether task f has been offloaded at node i. $a_t^{rsrc} = \{c(t), p(t), b(t)\}$ indicates the resource utilization in the network, where $c_i(t) \in c(t)$ represents the computation resources of edge servers, $p_n(t) \in p(t)$ represents the transmit power distribution of the node at the current time, and $b^k(t) \in b(t)$ represents the bandwidth allocated to the current subchannel.

Through DT, an immediate reward r_t is returned, and the system is transferred to a new state s_{t+1} with state transition matrix $\mathcal{P}$. The goal of the DRL agent is to find an optimal policy that can maximize the long-term cumulative reward $\mathcal{R}$. Our goal is to minimize the total end-to-end delay. Therefore, the immediate reward function is defined as

$$r_t = -\sum_{n,i \in \mathcal{N}, f \in \mathcal{F}} D_{n,i,f}^{total} \tag{20}$$

The reward described in (20) is the delay criteria; the smaller the processing delay, the higher the reward value. If the service delay of a task exceeds its maximum delay bound, the task will be discarded. In this case, we set the punish reward $D_{n,i,f}^{total} = \tau_f$.

4.2. Solution Based on DDPG

In this section, the DT-enabled network is considered as a DRL agent to resolve tasks and make decisions. At each time step t, the agent is given a set of states and learns the policy that allows the agent to maximize its rewards in this environment. After executing the action, the agent obtains feedback from the environment and accesses the value of the policy to optimize the parameters of the neural network. The commonly used DDPG algorithm is an actor–critic algorithm that supports continuous action spaces. It uses two networks to generate actions and evaluate the current policy. The structure of the DRL agent is shown in Figure 3. Both the actor and the critic networks contain an original network and a target network.

Let us start with the actor network. At time step t, the agent executes a_t in the DT virtual environment and then obtains the next state s_{t+1} and immediate reward r_t. Existing known tuples (s_t, a_t, s_{t+1}, r_t) are stored in the experience replay buffer for further training. When the experience buffer overflows, DDPG constructs a mini-batch by randomly selecting M samples in the buffer. Let (s_m, a_m, s_{m+1}, r_m) be the m sample in the mini-batch. At the same time, the actor takes as input s_m and outputs a deterministic action, and the network that produces this action is defined as $a_m = \pi_\mu(s_m)$. Here, π denotes the deterministic policy and μ represents the parameters of the Original Actor network. When this action is input to the Critic network, it can obtain the maximum value $Q_\theta(s_m, a_m)$. The ideal output of the network is an accurate evaluation of the policy, so that the Actor network can adjust the policy accordingly and discard the actions with poor Q value feedback. The Actor network is designed to maximize the output Q value of the Critic network, so it is updated based on gradient boosting. The loss function is given a minus sign to minimize the error.

$$L(\mu) = \mathbb{E}\left[-Q_\theta(s_m, \pi_\mu(s_m))\right] \tag{21}$$

The gradient expression of Q with μ can be obtained by the chain derivation rule.

$$\nabla_\mu J(\mu) = \frac{1}{M}\sum_{m=1}^{M}\left(\nabla_{a_m} Q_\theta(s_m, a_m)\nabla_\mu \pi_\mu(s_m)\right) \tag{22}$$

Moreover, it is the Target Actor network that updates the value network Critic, which has the same structure as the Original Actor network except that its parameter is μ'. Both Actor networks output actions, which soft update the parameters of the Target Actor network.

$$\mu' \leftarrow \rho\mu + (1-\rho)\mu' \tag{23}$$

Here ρ is a small step factor that controls the updating speed.

Now consider the Critic network; its role is to fit the value function $Q_\theta(s_m, a_m)$. Similarly, the Critic network has an original network and a target network with parameters θ and θ', respectively. The input sides of the two Critic networks are from different sources, and both output sides are the Q value of the current state. The input of the Original Critic network is the action output from the current Original Actor network. There are two Target Critic network inputs, the observed value of the current state and the action output from the Target Actor network.

The optimal value of the state–action pair is obtained according to Bellman equation:

$$Q^*(s_m, a_m) = \mathbb{E}(r(s_m, a_m) + \gamma Q^*(s_{m+1}, a_{m+1})) \tag{24}$$

where Q^* denotes the optimal value. In DDPG, we directly approximate Q^* with a neural network Q_θ with parameter θ. Through training, the neural network can output an accurate estimate of Q_θ for a given state–action pair. Then, we build a loss function measured

by mean squared error through temporal difference (TD) learning, which can be used to measure the estimated deviation of $Q_\theta(s_m, a_m)$ to the Bellman equation.

$$L(\theta) = \mathbb{E}[(r(s_m, a_m) + \gamma Q^*(s_{m+1}, a_{m+1}) - Q_\theta(s_m, a_m))^2] \tag{25}$$

Next, we obtain the gradient for the loss function

$$\nabla_\theta J(\theta) = \mathbb{E}[2(r(s_m, a_m) + \gamma Q^*(s_{m+1}, a_{m+1}) - Q_\theta(s_m, a_m))\nabla Q_\theta(s_m, a_m)] \tag{26}$$

With the gradient, we can use gradient descent to solve the above optimization problem. The TD target $r(s_m, a_m) + \gamma Q^*(s_{m+1}, a_{m+1})$ changes after each network parameter update, which is not conducive to the stability of training. Therefore, it is common to use two sets of networks: an original network (parameter θ) and a target network (parameter θ') to estimate $Q_\theta(s_m, a_m)$ and $Q_{\theta'}(s_{m+1}, a_{m+1})$, respectively. That is, the parameters of the target network are not updated immediately with each update but are synchronized with the original network parameters after a certain interval.

$$\nabla_\theta J(\theta) = \frac{2}{M}\sum_{m=1}^{M}[r(s_m, a_m) + \gamma Q_{\theta'}(s_{m+1}, a_{m+1}) - Q_\theta(s_m, a_m)]\nabla Q_\theta(s_m, a_m) \tag{27}$$

Soft updates to the parameters of the Target Critic network are given by

$$\theta' \leftarrow \rho\theta + (1-\rho)\theta' \tag{28}$$

The workflow of the proposed algorithm is presented in Algorithm 1.

Algorithm 1: DRL-driven computation offloading algorithm with DT-enabled architecture

Input: The original Actor parameter μ. The target Actor parameter μ'. The original Critic parameter θ. The target Critic parameter θ'. Discount factor γ. Soft update step factor ρ. Mini-batch size M.
Output: Computation offloading scheme and network resource allocation.

```
Initialize Actor networks and Critic networks.
for epi = 1 to threshold_E do
    Digital twin observes its DT_n.
    Normalize the state S and receive the initial state S_1.
    for step = 1 to threshold_T do
        Choose and execute action a_t and map it onto DT_n.
        Calculate reward r_t and receive next station s_{t+1}.
        if the experience replay buffer is not full then
            Store transition (a_t, s_t, s_{t+1}, r_t) in the buffer.
        else
            Randomly replace a tuple in the buffer.
            Sample a minibatch of M transition (a_m, s_m, s_{m+1}, r_m).
            for m=1 to M do
                Calculate the target value based on
                 Q*(s_m, a_m) = E(r(s_m, a_m) + γQ*(s_{m+1}, a_{m+1})).
                Update the Original Critic network by
                 L(θ) = E[(r(s_m, a_m) + γQ*(s_{m+1}, a_{m+1}) - Q_θ(s_m, a_m))^2].
                Update the Original Actor network by
                 L(μ) = (1/M) Σ_{m=1}^{M} (∇_{a_m} Q_θ(s_m, a_m) ∇_μ π_μ(s_m)).
            end for
        end if
        Soft update the Target Actor network by
         μ' ← ρμ + (1 − ρ)μ' .
        Soft update the Target Critic network by
         θ' ← ρθ + (1 − ρ)θ' .
    end for
end for
```

5. Experiments and Discussion

In this section, we carry out extensive experiments to simulate the proposed DT-enabled DDPG algorithm. These experiments are used to evaluate the learning process and network performance of the proposed computation offloading scheme in a 6G IIoT scenario.

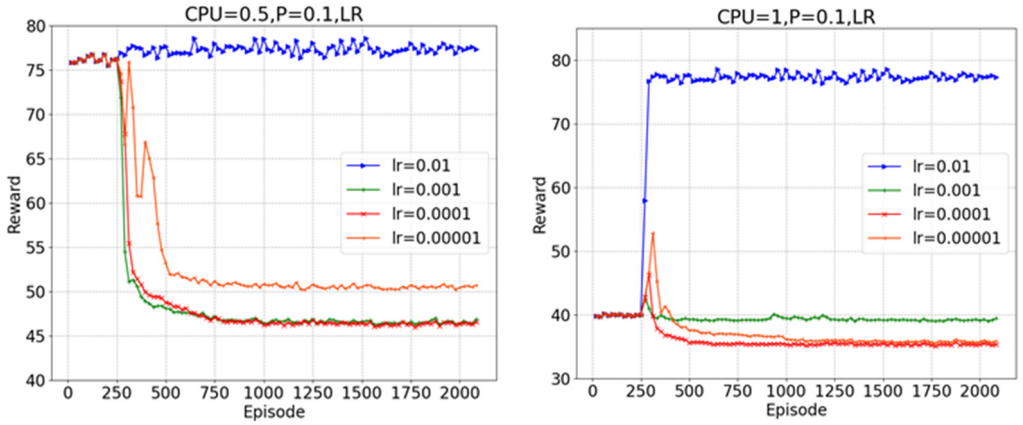
We use Python 3.8 and TensorFlow 2.4 to evaluate our proposed algorithm. We consider a network with a maximum of 12 edge servers. The physical devices are connected to the edge servers through access points and the number of IIoT devices is between 40 and 50. The process by which IIoT devices generate computing tasks follows a Poisson distribution. The arrival rate of tasks on each access point is between 1 and 5 per second. We consider Rayleigh fading channels. Three fully connected hidden layers with [256 256 512] neurons are deployed for DDPG, and its output layer is activated by the tanh function. The batch size is 32 and the total training episode is 2000. All the experimental results have been performed by extracting 100 data points on a PC with 2.81 GHz CPUs, 8.00 GB RAM, and Windows 10 OS. The key parameters employed in the simulations are summarized in Table 3, unless specified.

Table 3. Key parameters in experiments.

Parameters	Value
Maximum number of edge servers	12
Number of physical devices	40~50
CPU cycle frequency of IIoT	(0, 1] GHz
CPU cycle frequency of edge server	(1, 2] GHz
Task computation requirement	[100, 200] M
Bandwidth	20 MHz
Gaussian white noise power	10^{-11} mW
Batch size	32
Episodes	2000
Sample rate	0.1

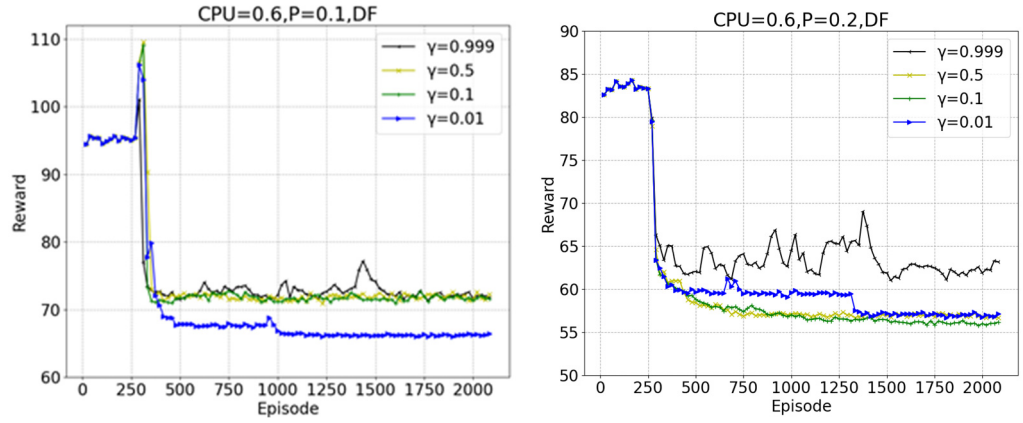
We first tested the learning relationships of the DDPG algorithm with different parameter settings to get the best rewards. Figure 4 describes the influence of the proposed algorithm on the convergence performance at different learning rates. The proposed algorithm achieves poor performance after convergence when the learning rate is 0.01. The reason is that the greater learning rate will cause both the actor network and the critic network to adopt large updating steps. The algorithm converges faster and can obtain better cumulative rewards when the learning rate is 0.001. When the learning rate is very small, e.g., 0.0001 and 0.00001, the algorithm produces large fluctuations in the convergence process. The reason for this is that the lower learning rate makes the network update more slowly and therefore requires more training steps to obtain better results. In subsequent experiments, we set the learning rate to 0.001.

Next, we evaluate the effect of the discount factor on the system rewards. Figure 5 describes the influence of different discount factors on the convergence performance of the proposed algorithm. The discount factor is used to adjust the weight of the effect of long-term and short-term rewards on the system. The larger the γ value, the more the agent considers long-term training factors, and the higher the training difficulty. The smaller the γ value, the more the agent focuses on short-term benefits, and the lower the training difficulty. From the figure, we can see that the trained rewards value has the best performance when the discount factor is 0.01 (Figure 5a) and 0.1 (Figure 5b), respectively. The reason is that the environment of different periods is quite different, so the generalization ability of different periods is poor. The discount factor is not a universal hyperparameter, and its optimal value may be affected by many factors such as task, environment, and algorithm. In different scenarios, the discount factor needs to be adjusted and optimized to maximize the performance of the algorithm. In the following experiment, our communication scenario is set as a discount factor of 0.01.



(**a**) (**b**)

Figure 4. System rewards under different learning rates: (**a**) $C_n = 0.5$ GHz, $p_n^k = 0.1$ W; (**b**) $C_n = 1$ GHz, $p_n^k = 0.1$ W.



(**a**) (**b**)

Figure 5. System rewards under different discount factors: (**a**) $C_n = 0.6$ GHz, $p_n^k = 0.1$ W; (**b**) $C_n = 0.6$ GHz, $p_n^k = 0.2$ W.

For performance comparison, we enumerate two benchmark schemes: (1) the local processing scheme and (2) the nearest edge server selection scheme.

(1) Local processing scheme: all computing tasks are performed by local devices, and DT is not enabled. The simulation scenario is a normal 6G IIoT network.
(2) Nearest edge server selection scheme: in a DT-enabled network architecture, the nearest edge server is selected to offload the computation tasks.

Figure 6 depicts the end-to-end delays under different offloading strategies. The proposed DT-enabled DDPG algorithm is compared with two benchmark algorithms. From the figure, we can see that in all three scenarios, the end-to-end delays lengthen as the amount of task computation requirement increases. When the network throughput reaches equilibrium, the end-to-end delays stay at their maximum values, which are 175.51 s, 108.23 s, and 79.82 s, respectively. This is because as the amount of task computation requirement increases, the system needs more time to process the data. When the task computation requirement is a constant, the end-to-end delay of the proposed DT-enabled DDPG algorithm is the smallest, followed by the DT-enabled nearest selection, and the end-to-end delay of the local processing is the longest. This is because the local processing capability is limited, and although the transmission delay associated with offloading tasks is avoided, the processing delay is the longest. The factors to consider in DT-enabled

nearest selection are relatively simple, which may cause an edge server to become too busy and increase its processing delay.

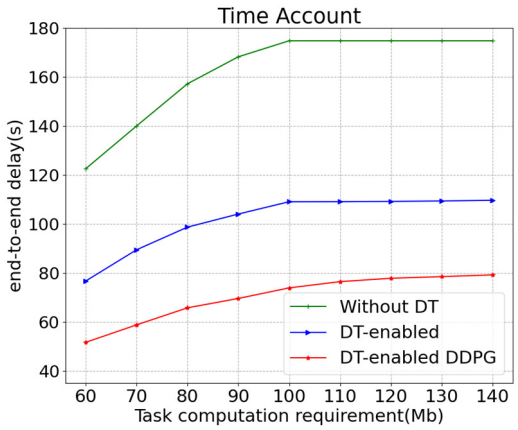


Figure 6. End-to-end delays under different task computation requirements.

Figure 7 illustrates the variation of end-to-end delay with the change in the number of edges servers in different scenarios. Since this scenario requires the participation of the edge server, we compare the variation of end-to-end delays when the transmission powers are different. As can be seen from the figure, when the transmission power is larger, the end-to-end delay will be reduced, because the transmission delay is reduced. The gap between the two schemes increases as the number of edge servers increases, with a maximum gap of about 33.52 s. Similarly, when the transmission power is constant, the delay of DT-enabled DDPG is smaller than that of the DT-enabled nearest selection. For example, when the transmission power is 0.1, the maximum difference between the two schemes can reach 36.25 s. As the number of edge servers increases, it brings more system processing capability, so the end-to-end delay decreases.

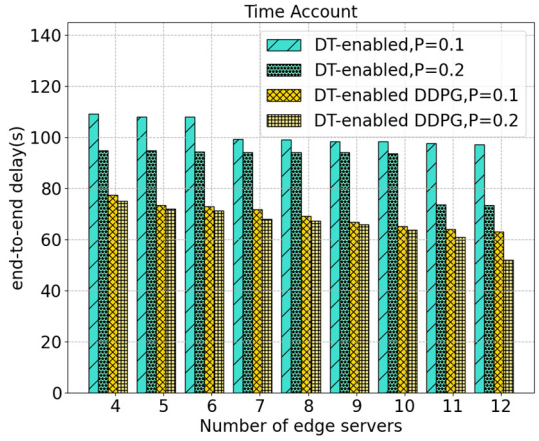


Figure 7. End-to-end delays under different numbers of edge servers.

Figure 8 depicts the variation of end-to-end delay with the change of the edge server CPU cycle frequency. Since the changes in the processing capability of the edge servers also have no impact on the local processing, we set the scenarios to be the same as in Figure 7. As can be seen from the figure, as the CPU cycle frequency increases, the end-to-end delay decreases. The value is reduced from a maximum of 98.01 s (0.1 W) to a minimum of 49.98 s (0.2 W). When the transmission power remains the same, the difference between the two schemes is about 20 s to 30 s. This is due to the increased processing capability of edge servers, which reduces the processing delay in the total end-to-end delay.

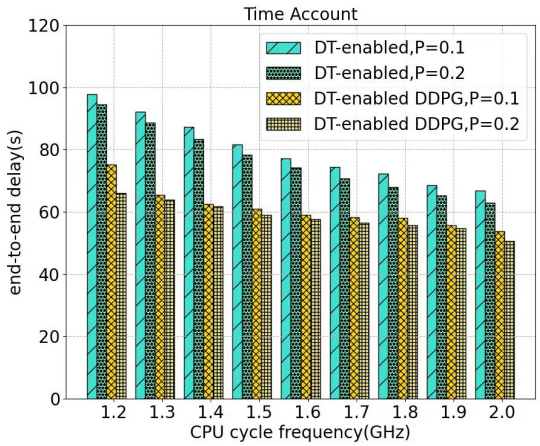


Figure 8. End-to-end delays under different CPU cycle frequencies.

Figure 9 shows the influence of different offloading strategies on the task success rate. It can be seen from the figure that the task success rate decreases as the amount of data generated for the task increases. The increased amount of data means that since more network resources are needed to process it, some tasks are discarded because they exceed the maximum delay bound. This can be improved by increasing the processing capability of the CPU (both local and edge). In the case of the same amount of task computation requirement data, local processing has the longest end-to-end delay, so it has the lowest success rate, which is from 86.43% to 57.03%. Algorithm DT-enabled nearest selection has an intermediate success rate, which is from 92.47% to 63.87%. Algorithm DT-enabled DDPG has the highest success rate due to the comprehensive consideration of various factors, which is from 98.03% to 95.01%.

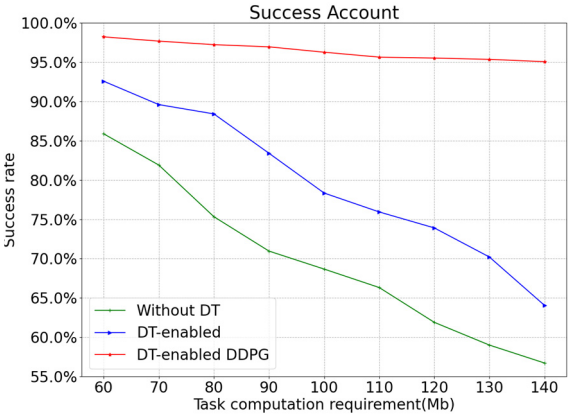


Figure 9. Success rates under different offloading strategies.

6. Conclusions

In this article, we propose a computational offloading strategy in 6G IIoT networks with DT-enabled network architecture by considering end-to-end delay as an optimization objective. This strategy provides a real-time and effective method for processing heterogeneous task requests in 6G networks. The main innovations include the following two. First, the mobile IIoT device finds the optimal offload edge server in the network system enabled by DT. Second, we design a DRL-based offloading system that employs the DDPG algorithm to train the offloading strategy and minimize the end-to-end delay of the task by jointly solving the optimization combination problem. Numerical results show that compared with the benchmark schemes, the proposed algorithm has achieved significant improvement in both task success rate and end-to-end delay.

As a future research direction, we will consider the load balancing problem on edge servers and the collaboration among edge servers. We will consider migrating the tasks on computationally heavy servers partially or entirely to other servers for processing. How to select the right server for task migration will be a challenging research direction.

Author Contributions: J.W. designed the experiment scheme, analyzed the experimental results, and wrote the paper. R.Z. conducted simulation experiments. All authors have read and agreed to the published version of the manuscript.

Funding: This paper was support by National Key Research and Development Program of China (Grant No. 2021 YFB3401004).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data is contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest and the funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Wei, J.; Bin, H.; Mohammad, A.H.; Hans, D.S. The road towards 6G: A comprehensive survey. *IEEE Open J. Commun. Soc.* **2021**, *2*, 334–366.
2. Aqeel, T.J.; Rihab, M.; Lamia, C. A comprehensive survey on 6G and beyond: Enabling technologies, opportunities of machine learning and challenges. *Comput. Netw.* **2023**, *237*, 110085.
3. Xiaoheng, D.; Leilei, W.; Jinsong, G.; Ping, J.; Xuechen, C.; Feng, Z.; Shaohua, W. A review of 6G autonomous intelligent transportation systems: Mechanisms, applications and challenges. *J. Syst. Architech.* **2023**, *142*, 102929.
4. Di, Z.; Min, S.; Jiandong, L.; Zhu, H. Aerospace integrated networks innovation for empowering 6G: A survey and future challenges. *IEEE Commun. Surv. Tutor.* **2023**, *25*, 975–1019.
5. Jerry, C.W.L.; Gautam, S.; Yuyu, Z.; Youcef, D.; Moayad, A. Privacy preserving multiobjective sanitization model in 6G IoT environments. *IEEE Internet Things* **2021**, *8*, 5340–5349.
6. Bomin, M.; Jiajia, L.; Yingying, W.; Nei, K. Security and privacy on 6G network edge: A survey. *IEEE Commun. Surv.* **2023**, *25*, 1095–1127.
7. Demos, S.; Mohsen, K.; Tim, W.C.B.; Rahim, T. Terahertz channel propagation phenomena, measurement techniques and modeling for 6G wireless communication applications: A survey, open challenges and future research directions. *IEEE Commun. Surv. Tutor.* **2022**, *24*, 1957–1996.
8. Xiaofei, W.; Yiwen, H.; Victor, C.M.L.; Dusit, N.; Xueqiang, Y.; Xu, C. Convergence of edge computing and deep learning: A comprehensive survey. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 869–904.
9. Houming, Q.; Kun, Z.; Nguyen, C.L.; Changyan, Y.; Dusit, N.; Dong, I.K. Applications of auction and mechanism design in edge computing: A survey. *IEEE T. Cogn. Commun.* **2022**, *8*, 1034–1058.
10. Yuyi, M.; Changsheng, Y.; Jun, Z.; Kaibin, H.; Khaled, B.L. A survey on mobile edge computing: The communication perspective. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 2322–2358.
11. Binayak, K.; Widhi, Y.; Ying, D.L.; Asad, A. A survey on offloading in federated cloud-edge-fog systems with traditional optimization and machine learning. *arXiv* **2022**, arXiv:2202.10628.
12. Quyuan, L.; Shihong, H.; Changle, L.; Guanghui, L.; Weisong, S. Resource scheduling in edge computing: A survey. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 2131–2165.
13. Yao, C.; Yi, Z.; Hao, L.; Tse, Y.C.; Guan, H.C.; Huan, T.C.; Yan, J.W.; Hung, Y.W.; Chun, T.C. Management and orchestration of edge computing for IoT: A comprehensive survey. *IEEE Internet Things* **2023**, *10*, 14307–14331.
14. Yiwen, W.; Ke, Z.; Yan, Z. Digital twin networks: A survey. *IEEE Internet Things* **2021**, *8*, 13789–13804.
15. Fengxiao, T.; Xuehan, C.; Tiago, K.R.; Ming, Z.; Nei, K. Survey on digital twin edge networks (DITEN) toward 6G. *IEEE Open J. Commun. Soc.* **2022**, *3*, 1360–1381.
16. Stefan, M.; Mahnoor, Y.; Dang, V.H.; William, D.; Praveer, T.; Mohsin, R.; Mehmet, K.; Balbir, B.; Dattaprasad, S.; Raja, V.P.; et al. Digital twins: A survey on enabling technologies, challenges, trends and future prospects. *IEEE Commun. Surv. Tutor.* **2022**, *24*, 2255–2291.
17. Yuntao, W.; Zhou, S.; Shaolong, G.; Minghui, D.; Tom, H.L.; Yiliang, L. A survey on digital twins: Architecture, enabling technologies, security and privacy, and future prospects. *IEEE Internet Things* **2023**, *10*, 14965–14987.
18. Hansong, X.; Jun, W.; Qianqian, P.; Xinping, G.; Mohsen, G. A survey on digital twin for industrial internet of things: Applications, technologies and tools. *IEEE Commun. Surv. Tutor.* **2023**, *25*, 2569–2598.

19. Julia, R.; Cristian, M.; Manuel, D. Opentwins: An open-source framework for the development of next-gen compositional digital twins. *Comput. Ind.* **2023**, *152*, 104007.
20. Md, M.H.S.; Sajal, K.D.; Safwat, M.C. Design, development, and optimization of a conceptual framework of digital twin electric grid using systems engineering approach. *Electr. Pow. Syst. Res.* **2023**, *226*, 109958.
21. Van-Dinh, N.; Saeed, R.K.; Vishal, S.; Octavia, A.D. URLLC edge networks with joint optimal user association, task offloading and resource allocation: A digital twin approach. *IEEE Trans. Commun.* **2022**, *70*, 7669–7682.
22. Bin, T.; Lihua, A.; Min, W.; Jiaxi, W. Toward a task offloading framework based on cyber digital twins in mobile edge computing. *IEEE Wirel. Commun.* **2023**, *30*, 157–162.
23. Long, C.; Shaojie, Z.; Yalan, W.; Hong-Ning, D.; Jigang, W. Resource and fairness-aware digital twin service caching and request routing with edge collaboration. *IEEE Wirel. Commun. Lett.* **2023**, *12*, 1881–1885.
24. Yunlong, L.; Sabita, M.; Yan, Z. Adaptive edge association for wireless digital twin networks in 6G. *IEEE Internet Things* **2021**, *8*, 16219–16230.
25. Lindong, Z.; Shouxiang, N.; Dan, W.; Liang, Z. Cloud-edge-client collaborative learning in digital twin empowered mobile networks. *IEEE J. Sel. Areas Commun.* **2023**, *41*, 3491–3503.
26. Haijun, L.; Zhenyu, Z.; Nian, L.; Yan, Z.; Guangyuan, X.; Zhenti, W.; Shahid, M. Cloud-edge-device collaborative reliable and communication-efficient digital twin for low-carbon electrical equipment management. *IEEE Trans. Ind. Inform.* **2023**, *19*, 1715–1724.
27. Zhe, J.; Sheng, W.; Chunxiao, J. Cooperative multi-agent deep reinforcement learning for computation offloading in digital twin satellite edge networks. *IEEE J. Sel. Areas Commun.* **2023**, *41*, 3414–3429.
28. Siguang, C.; Jiamin, C.; Yifeng, M.; Qian, W.; Chuanxin, Z. Deep reinforcement learning-based cloud-edge collaborative mobile computation offloading in industrial networks. *IEEE Trans. Signal Inf. Process. Netw.* **2022**, *8*, 364–375.
29. Nikos, A.M.; Vasilis, K.P.; Panagiotis, D.D.; Trung, Q.D.; George, K.K. Digital twin-aided orchestration of mobile edge computing with grant-free access. *IEEE Open J. Commun. Soc.* **2023**, *4*, 841–853.
30. Kai, P.; Hualong, H.; Muhammad, B.; Xiaolong, X. Distributed incentives for intelligent offloading and resource allocation in digital twin driven smart industry. *IEEE Trans. Ind. Inform.* **2023**, *19*, 3133–3143.
31. Dang, V.H.; Van-Dinh, N.; Saeed, R.K.; George, K.K.; Trung, Q.D. Distributed communication and computation resource management for digital twin-aided edge computing with short-packet communications. *IEEE J. Sel. Areas Commun.* **2023**, *41*, 3008–3021.
32. Bowen, W.; Yanjing, S.; Haejoon, J.; Long, D.N.; Nguyen-Son, V.; Trung, Q.D. Digital twin-enabled computation offloading in UAV-assisted MEC emergency networks. *IEEE Wirel. Commun. Lett.* **2023**, *12*, 1588–1592.
33. Long, Z.; Han, W.; Hongmei, X.; Hongliang, Z.; Qilie, L.; Dusit, N.; Zhu, H. Digital twin-assisted edge computation offloading in industrial internet of things with NOMA. *IEEE Trans. Veh. Technol.* **2023**, *72*, 11935–11950.
34. Lingxiao, C.; Qiangqiang, G.; Kai, J.; Liang, Z. A3C-based and dependency-aware computation offloading and service caching in digital twin edge networks. *IEEE Access* **2023**, *11*, 57564–57573.
35. Xiaolong, X.; Zhongjian, L.; Muhammad, B.; Vimal, S.; Houbing, S. Computation offloading and service caching for intelligent transportation systems with digital twin. *IEEE Trans. Intell. Transp. Syst.* **2022**, *23*, 20757–20772.
36. Zhixiu, Y.; Shichao, X.; Yun, L.; Guangfu, W. Cooperative task offloading and service caching for digital twin edge networks: A graph attention multi-agent reinforcement learning approach. *IEEE J. Sel. Areas Commun.* **2023**, *41*, 3401–3413.
37. Yueyue, D.; Ke, Z.; Sabita, M.; Yan, Z. Deep reinforcement learning for stochastic computation offloading in digital twin networks. *IEEE J. Sel. Areas Commun.* **2021**, *17*, 4968–4977.
38. Bin, L.; Yufeng, L.; Ling, T.; Heng, P.; Yan, Z. Digital twin assisted task offloading for aerial edge computing and networks. *IEEE Trans. Veh. Technol.* **2022**, *71*, 10863–10877.
39. Yongchao, Z.; Jia, H.; Geyong, M. Digital twin-driven intelligent task offloading for collaborative mobile edge computing. *IEEE J. Sel. Areas Commun.* **2023**, *41*, 3386–3400.