Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE TYPE

Trust-enabled Decentralized Task Offloading for Collaborative Edge Computing Using Blockchain and Deep Reinforcement Learning

Genyuan Yang[1] | **Wenjuan Li**[1] | **Qifei Zhang**[2] | **Minxian Xu**[3] | **Chengjie Pan**[1] | **Ben Wang**[1]

[1]School of Information Science and Technology, Hangzhou Normal University, Zhejiang, China

[2]School of Software Engineering, Zhejiang University, Zhejiang, China

[3]Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China

Correspondence

Corresponding author: Wenjuan Li.

Email: liwenjuan@hznu.edu.cn

Present address

No. 2318 Yuhangtang RD, Hangzhou 311121, Zhejiang, China.

Abstract

Collaborative edge computing (CEC) addresses resource limitations of single-node architectures by integrating resources from multiple edge nodes. However, ensuring reliable task offloading in the CEC environment remains a significant challenge. Existing solutions struggle to balance intelligence and trustworthiness in offloading decisions, leading to poor performance and low task success rates, especially when tasks are offloaded to malicious nodes. To tackle these challenges, this paper proposes a trust-enabled decentralized task offloading scheme combining blockchain technology and deep reinforcement learning (DRL). We introduce a blockchain-based reputation mechanism with smart contracts to facilitate trusted collaboration among nodes, a beta distribution-based three-factor reputation update (BTRU) algorithm for accurate reputation evaluation, and a decentralized trust-enabled task offloading (DTTO) algorithm that uses on-chain reputation data to guide trustworthy offloading policies. Based on Kubernetes and Ethereum, we developed a testbed to assess the performance of the proposed scheme. Experimental results demonstrate that BTRU reduces malicious nodes' average reputation by 97.54%, with 9.94% improvement over competitors, while DTTO increases task success rates by at least 3.04%, reaching 5.41% improvement over competitors when malicious nodes comprise 40% of the network.

KEYWORDS

collaborative edge computing, decentralized task offloading, deep reinforcement learning, blockchain, reputation management

1 | INTRODUCTION

With the rapid proliferation of Internet of Things (IoT) devices (e.g., wearable devices, smart home devices), a vast number of new compute-intensive and latency-sensitive applications (e.g., telemedicine, virtual reality) have emerged[1,2]. Due to the stringent demands of these applications for high bandwidth and low latency, edge computing has become the preferred paradigm for processing end-user tasks at the network edge[3]. However, the computing resources of a single edge node are often limited, and the workloads of different edge nodes are highly dynamic and geographically unbalanced. Traditional edge computing paradigms struggle to consistently provide stable and efficient services in complex and ultra-dense smart environments. Fortunately, CEC can effectively improve resource utilization and the quality of experience (QoE) for users by enabling collaboration among edge nodes. For instance, overloaded edge nodes can offload some of their workload to other less-loaded edge nodes, thereby achieving workload balancing at the edge layer[4,5].

Abbreviations: CEC, collaborative edge computing; DRL, deep reinforcement learning; BTRU, beta distribution-based three-factor reputation update; DTTO, decentralized and trust-enabled task offloading; IoT, Internet of Things; QoE, quality of experience; MEC, mobile edge computing; DDQN, double deep Q-network; SAC, soft actor-critic; PoR, proof of reputation; first-in-first-out, FIFO; DPOMDP, decentralized partially observable Markov decision process; PPO, proximal policy optimization; LSTM, long short-term memory; GAE, generalized advantage estimation; FC, fully-connected; BRMTO, blockchain-based reputation management task offloading.

However, CEC faces the following issues: 1) Edge nodes may originate from different manufacturers or service providers, introducing the risk of malicious behaviors. Some edge nodes may reject or delay processing computing requests from other edge nodes, or even handle them incorrectly. 2) In the dynamic and decentralized CEC environment, the reliability of edge nodes is difficult to evaluate due to environmental changes and the absence of centralized coordination. Consequently, trustworthy task offloading is essential. In this paper, we define it as the decision process that, under resource constraints, maximizes the probability of tasks completed before their deadline, while simultaneously minimizing the likelihood of offloading tasks to malicious nodes.

Despite existing task offloading research for CEC has achieved many results, these solutions still exhibit certain limitations. Specifically, centralized approaches[6,7] rely on global knowledge of the system state to make decisions for each task. However, in large-scale network scenarios, due to the rapid changes in system state, obtaining and transmitting global information will bring extremely high communication overhead. Mathematical optimization-based decentralized approaches[8,9] are often highly model-dependent and exhibit poor adaptability to dynamic and complex environments. Heuristic-based decentralized approaches[10,11,12] may suffer from performance degradation in high-dimensional spaces and primarily focus on greedy optimization, making them prone to local optima. Similarly, game theory-based decentralized approaches[13,14] rely on strict assumptions to compute static Nash equilibrium and are difficult to accurately describe real-world scenarios.

DRL provides a promising solution to address the shortcomings of these traditional methods. DRL enables adaptive decision-making based on dynamic environments and optimizes objectives from a long-term perspective. However, most existing DRL-based studies[15,16,17] primarily focus on minimizing latency or energy consumption in CEC while overlooking the presence of potentially malicious nodes in the network. Since not all nodes in the edge network are trustworthy, malicious nodes may disrupt collaboration among nodes, thereby reducing overall system performance.

Recently, blockchain technology has gained widespread attention for its potential to enhance trust and security in decentralized systems[18,19,20]. Some studies[21,22,23] have leveraged blockchain-based reputation systems to maintain reputation values, using them as key factors in selecting target collaboration nodes. Although reputation-based decision-making schemes can go some way toward mitigating the offloading of tasks to malicious nodes, their heavy reliance on reputation data limits their flexibility in dynamic and complex environments. In particular, these solutions struggle to efficiently coordinate and utilize trusted computing resources, resulting in many tasks violating deadlines. Therefore, task offloading schemes for CEC need to simultaneously be intelligent and trustworthy to address various challenges in dynamic CEC environments.

To address the aforementioned challenges, this paper proposes a trust-enabled decentralized task offloading scheme based on blockchain and DRL. First, we introduce a blockchain-based reputation mechanism into the CEC architecture to ensure trust-enabled collaboration among nodes. This mechanism maintains reputation values based on historical performance and interactions of nodes. Secondly, we propose a beta distribution-based three-factor reputation update (BTRU) algorithm to accurately and efficiently evaluate the reputation of edge nodes. Furthermore, we design a decentralized and trust-enabled task offloading (DTTO) algorithm based on DRL, which effectively prevents task offloading to malicious nodes while efficiently coordinating and utilizing trusted computing resources to improve task success rates. The main contributions of this paper are as follows:

- This paper introduces a blockchain-based reputation mechanism into the CEC architecture to ensure trust-enabled collaboration among edge nodes. Each edge node acts as a computing resource provider and a blockchain node. Smart contracts are mainly used for reputation management. This creates a trusted platform for edge collaboration, ensuring the transparency and security of the task offloading.
- To ensure the efficiency and accuracy of reputation evaluation in dynamic CEC environments, this paper proposes a beta distribution-based three-factor reputation update algorithm (BTRU). By comprehensively considering three key factors (historical performance, historical task success rate, and task difficulty), the proposed algorithm enhances the efficiency and accuracy of reputation evaluation, helping agents effectively identify malicious nodes.
- To ensure trustworthy task offloading and improve task reliability, this paper proposes a decentralized and trust-enabled task offloading (DTTO) algorithm based on DRL, which incorporates reputation data as a crucial part of both observation and reward, enabling the agent to consider the long-term behavioral performance and reputation levels of edge nodes when making decisions. It not only minimizes the likelihood of offloading tasks to malicious nodes but also efficiently coordinates and utilizes trusted computing resources while maximizing task success rates.
- A Kubernetes and Ethereum-based testbed for CEC task offloading is developed to comprehensively evaluate the effectiveness and practicality of the proposed scheme under realistic network conditions. The testbed integrates real-world communication, computation, and blockchain environments, and includes implementations of baseline algorithms for fair comparison.

- A series of experiments is designed to evaluate the performance of the proposed scheme. Experimental results demonstrate that, compared to previous works, the BTRU algorithm achieves higher efficiency and accuracy in reputation evaluation, while the DTTO algorithm significantly optimizes the efficiency and reliability of task offloading. Notably, in the ultra-dense smart environment with malicious nodes, the DTTO algorithm improves the task success rate by at least 3.04%.

The rest of this paper is organized as follows. Section 2 reviews related works. Section 3 presents the system model and problem formulation. Section 4 provides detailed information on the proposed scheme. Section 5 introduces the Kubernetes and Ethereum-based testbed for edge computing task offloading. Section 6 presents the experimental results and analysis. Section 6 presents the experimental results and analysis in the simulation and testbed. Finally, Section 8 concludes the paper and suggests directions for future research.

2 | RELATED WORK

The task offloading problem for CEC has attracted widespread attention, and many approaches have been developed to optimize system performance. Sahni *et al.*[6] investigated partial offloading and flow scheduling in the CEC and proposed a heuristic algorithm aimed at minimizing the average makespan. He *et al.*[7] addressed the challenge of peer offloading in mobile edge computing (MEC), maximizing time-average throughput under the constraints of energy consumption and worst-case response time. Yamansavascilar *et al.*[24] introduced a task orchestrator based on double deep Q-network (DDQN) to balance workloads at the edge layer. However, all these studies are centralized algorithms, which incur high communication overhead and are not scalable to large-scale edge networks.

Some studies investigated the decentralized task offloading based on mathematical optimization or heuristic methods. Wang *et al.*[8] investigated the constrained task offloading problem in CEC and proposed a distributed algorithm based on gradient projection and virtual queue techniques, which effectively balances the workload between edge servers. Lin *et al.*[9] considered the computation offloading problem in CEC networks and proposed a distributed offloading algorithm based on Lyapunov optimization, achieving lower average energy consumption. Nujhat *et al.*[10] used the meta-heuristic algorithm based on binary multi-objective grey wolf optimization to find the optimal task allocation scheme that satisfies delay, energy consumption, and cost constraints within polynomial time. Sthapit *et al.*[11] investigated the use of collaboration among edge devices to achieve computational load balancing in the absence of cloud and fog support, and proposed four algorithms to minimize energy consumption and latency. However, these approaches either suffer from high computational complexity, making them difficult to adapt to dynamic environments, or struggle to find globally optimal solutions in high-dimensional scenarios.

Some works proposed game theory-based decentralized task offloading algorithms. Chen *et al.*[13] investigated the peer offloading problem among small-cell base stations. They formulated a peer offloading game between base stations to analyze and optimize offloading decisions. Jošilo *et al.*[14] employed a game theory model and variational inequality theory to develop a decentralized algorithm based on static mixed-strategy equilibrium, which addressed task offloading between devices and between devices and edge clouds. Game theory offers a unique perspective on task offloading by modeling node interactions. However, game theory-based algorithms struggle with convergence in dynamic environments, and their reliance on strict assumptions to compute static Nash equilibrium limits real-world applicability.

The development of DRL has opened new possibilities for overcoming the limitations of these traditional approaches. Sun *et al.*[5] proposed two intelligent computation offloading algorithms based on soft actor-critic (SAC) in an edge-cloud collaborative environment, achieving promising results in optimizing service latency. Lin *et al.*[4] introduced a DRL-based CEC task offloading algorithm that leverages heuristic task logs to pre-train the DRL model offline and fine-tunes it with online policy updates. Zhang *et al.*[25] proposed a digital twin-driven intelligent task offloading framework that uses DRL to optimize task offloading and resource allocation decisions in CEC systems. Compared to traditional approaches, DRL demonstrates greater adaptability in dynamic environments and the capability to optimize system performance over the long term. However, existing studies generally assume that computing resources in the system are trustworthy. In reality, some nodes, driven by selfish motivations, may disrupt collaboration among nodes, thereby degrading overall system performance.

To mitigate the impact of malicious nodes, reputation mechanisms have been proven effective in various scenarios[26,27,28]. Kong *et al.*[29] proposed a reliable and efficient task offloading strategy based on a multi-feedback trust mechanism to address reliability and rapid response issues in multi-demand task scenarios with a large number of heterogeneous resources. Ouyang *et al.*[30], the authors introduced a trust-based task offloading scheme in UAV-enhanced edge computing networks, aiming to achieve efficient

and reliable task offloading for IoT devices in UAV-assisted edge computing environments. However, the reputation data of these schemes relies on centralized storage or third-party trust authorities, posing risks of data tampering and single points of failure.

Due to the unique security and decentralized features, blockchain technology can be introduced into edge computing as a potential solution for enhancing trust and security[31,19]. Iqbal *et al.*[21] proposed a blockchain-based reputation management framework for task offloading, where task offloading decisions are made based on social reputation and queuing time. Wang *et al.*[22] investigated the task offloading problem in reputation-based consortium blockchain networks, proposing an improved delegated proof-of-stake consensus mechanism and optimizing user costs and blockchain node utilities through a three-tier Stackelberg game model. Hu *et al.*[23] introduced a trusted resource allocation scheme based on a proof-of-reputation consensus mechanism, combining reputation evaluation, incentive mechanisms, and smart contracts. Although incorporating reputation mechanisms into task offloading schemes can mitigate the impact of malicious nodes to some extent, these schemes often rely on several key factors (especially reputation values) to directly make task offloading decisions. This strong dependence on reputation data reduces the flexibility and adaptability of strategies in the complex and dynamic CEC environment. Moreover, due to the lack of effective optimization for resource competition and utilization, these approaches may lead to uneven resource allocation, increasing the risk of task timeouts and degrading overall system performance.

This paper proposes a trust-enabled decentralized task offloading scheme based on blockchain and DRL. Compared to traditional algorithms, the proposed scheme demonstrates greater adaptability to dynamic environments, effectively handling high-dimensional state spaces. Compared to DRL-based algorithms, the proposed scheme minimizes the possibility of offloading tasks to malicious nodes. Compared to other reputation-based approaches, the BTRU algorithm enables more accurate and efficient reputation evaluation. The DTTO algorithm avoids excessive reliance on reputation values by integrating reputation data into the training process of agents. Not only can it mitigate the impact of malicious nodes, but it can also enhance the flexibility of offloading decisions, efficiently coordinating resource competition and utilization.

3 | SYSTEM MODEL AND PROBLEM FORMULATION

In this section, we first outline the system model, followed by the definition of the task, transmission, computing, and delay models. Based on these foundations, we formulate the task offloading problem in CEC.

3.1 | System Models

A blockchain-based reputation mechanism is introduced into the CEC architecture, with the system model illustrated in Figure 1. The system consists of IoT devices and edge nodes, where each edge node is integrated with the blockchain node in physical deployment, and logically has dual capabilities of processing edge computing tasks and participating in blockchain consensus and data recording. This integrated design of edge-blockchain nodes reduces deployment overhead while achieving a tight integration between computing and the trust mechanism. As a blockchain-supported heterogeneous edge network, it is configured to run the proof of reputation (PoR) algorithm[32]. The PoR algorithm is a consensus mechanism based on the reputation value of nodes, in which a leader is randomly selected from nodes with higher reputation in each round, and the leader obtains the block production rights. And smart contracts on the blockchain are mainly used to achieve reputation management.

This network can be modeled as a connected graph $G = (V, E)$, where V represents the set of edge nodes, $V = \{e_i \mid 1 \leq i \leq N\}$, and E denotes the set of links connecting different edge nodes, $E = \{l_{i,j} \mid i, j \in V\}$. Time is discretized into multiple time slots of constant duration Δ, indexed by t, where $t \in \{1, 2, \ldots, T\}$. At the beginning of each time slot t, the reputation values of all edge nodes are represented as $H(t) = \{R_1, \ldots, R_N\}$. Each edge node $e_i \in V$ can receive a task $J_i(t)$ from local IoT devices at each time slot t and then decide where to process the task $J_i(t)$.

The main process of the proposed system is divided into the following five steps. (1) Smart contracts deployment. Smart contracts for reputation updating are deployed to the blockchain-supported CEC network. Each node maintains a copy of smart contracts, which includes logic for updating node reputation based on subsequent task processing results to achieve dynamic reputation updates on the chain. (2) Task offloading decision. When task $J_i(t)$ arrives at edge node e_i, the node e_i utilizes its DRL decision model, integrating local observations and on-chain reputation data to make an offloading decision. The task can be offloaded to a target node e_j for processing or processed locally without offloading. (3) Task offloading. If the task is offloading processing, edge node e_i will securely offload the task to edge node e_j using encryption techniques. If edge node e_j refuses to provide service, node e_i will initiate an on-chain transaction to call the smart contract, decreasing the reputation of node e_j. (4)

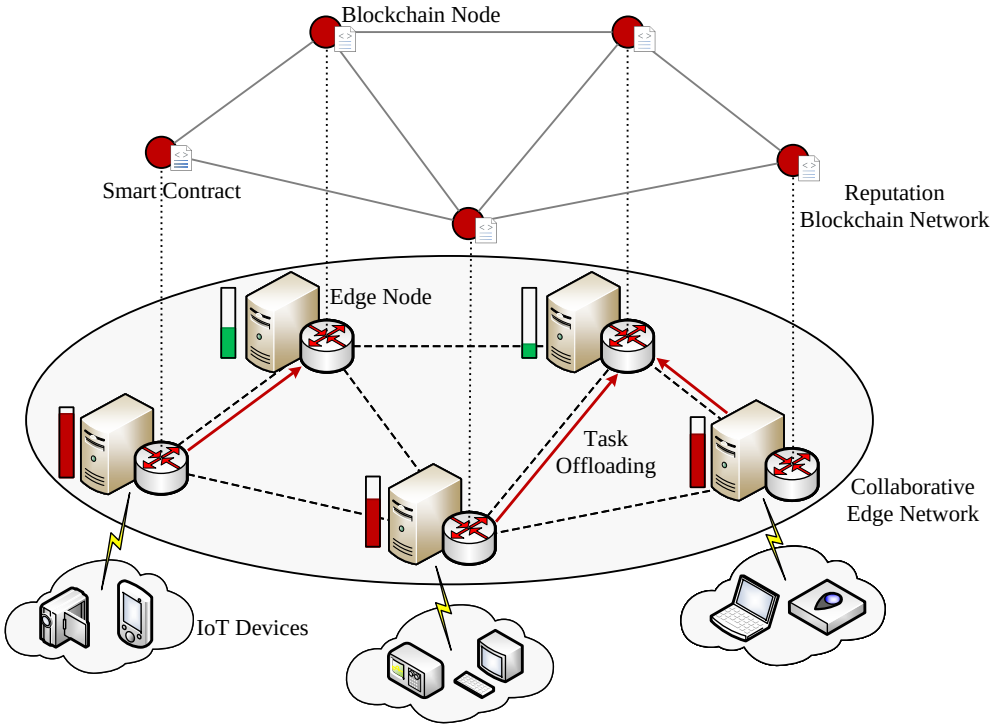


FIGURE 1 System model.

Result Handling. If node e_j completes the task within the deadline, node node e_i will initiate an on-chain transaction to call the smart contract, increasing the reputation of node e_j. If node e_j does not handle the task correctly or exceeds the task deadline, its reputation will decrease. (5) Transaction processing and reputation update. Edge node e_i initiates transactions invoking the related contract to update reputation and broadcasts them to the network. In the current round, according to the PoR algorithm, a leader node is randomly selected from nodes with higher reputation. The leader is responsible for collecting transactions and executing the corresponding smart contract logic to update the reputation. Then, the leader bundles the transactions into a new block, records the contract execution results in the block's state change list, and broadcasts the new block to the network. Upon receiving the new block, each node verifies its validity and updates its local blockchain state, achieving network-wide synchronization and consistency of reputation values.

In this paper, malicious nodes are defined as those that exhibit consistent and intentional misbehavior over time, such as refusing to process tasks, rejecting token staking, deliberately delaying processing, or incorrectly processing tasks. These behaviors are typically not caused by system resource limitations, but rather stem from a deliberate intent to evade cooperation or deceive the system. In contrast, nodes that occasionally fail to process tasks on time due to poor service quality or temporary overload are considered honest but underperforming.

3.1.1 | Task Model

Tasks can be of various types, such as computation-intensive, delay-sensitive, or a combination of both. We assume that at the beginning of time slot t, edge node e_i has new tasks arriving with a certain probability, denoted as $\lambda_i(t)$, which follows a Bernoulli distribution[4,33]. To more precisely describe the tasks, we model the task $J_i(t)$ arriving at edge node e_i at time slot t as a three-tuple $\langle s_i(t), \rho_i(t), \tau_i(t) \rangle$, where $s_i(t)$ represents the task size (in bits), $\rho_i(t)$ represents the processing density (in CPU cycles per bit), i.e., the number of CPU cycles required to process a unit of data, and $\tau_i(t)$ represents the delay constraint of the task, i.e., the deadline of the task.

3.1.2 | Transmission Model

Each edge node maintains $N-1$ transmission queues to transmit tasks to other edge nodes. At the same time, it also maintains N processing queues to receive and process tasks from the local IoT devices and other edge nodes. Tasks are transmitted in a first-in-first-out (FIFO) order. If there is a task currently being transmitted in the transmission queue, other tasks must wait. Once the transmission is completed, the transmitted task is automatically placed into the processing queue corresponding to its target computing resource for further processing, and will no longer be forwarded to other computing resources[14,4]. The transmission rate of each link $l_{i,j}$ varies over time and is a priori unknown. Let $r_{i,j}(t)$ represent the transmission rate between edge node e_i and e_j at time slot t. Assume that edge node e_i starts transmitting task $J_i(t)$ to edge node e_j at time slot x, where $x \in \{t, t+1, \ldots, t+\tau_i(t)-1\}$. The transmission delay $T_{i,j}(t)$ for this task can be calculated through the following steps: **(1) Initialization:** The initial remaining transmission amount is $S_{i,j} = s_i(t)$, i.e., the size of the task. **(2) Per slot update:** For each time slot $x+k$ (where $k \in \{0, 1, \ldots, \tau_i(t)-1\}$), the remaining transmission amount for the task is updated as:

$$S_{i,j} = S_{i,j} - \min\left(S_{i,j}, r_{i,j}(x+k) \cdot \Delta\right), \tag{1}$$

where Δ is the duration of a time slot, and $r_{i,j}(x+k) \cdot \Delta$ represents the maximum transmittable amount in this time slot. After updating through multiple time slots, until $S_{i,j} \leq 0$, the task transmission is completed, and the transmission delay is $T_{i,j}(t) = (k+1) \cdot \Delta$.

3.1.3 | Computing Model

Whether it is a task executed locally or received from other nodes, it will be automatically placed into the corresponding processing queue. Tasks in each processing queue are processed in a FIFO order. For any processing queue, if a task arrives or is being processed at a certain time slot t, this queue is considered an active queue. Let $B_j(t)$ represent the number of active queues at edge node e_j at time slot t. It is assumed that all edge nodes adopt the generalized processor sharing model, i.e., the processing capacity of an edge node is evenly distributed among its active queues[33,34]. The number of active queues $B_j(t)$ varies over time and is a priori unknown. Therefore, the processing capacity allocated to each active queue changes over time. Let the start time slot for edge node e_j processing task $J_i(t)$ received from edge node e_i be denoted by x, where $x \in \{t, t+1, \ldots, t+\tau_i(t)-1\}$. The processing delay $P_{i,j}(t)$ for task $J_i(t)$ can be calculated in two steps: **(1) Initialization:** The initial remaining processing amount is $R_{i,j} = s_i(t) \cdot \rho_i(t)$, which represents the number of CPU cycles required to complete the task. **(2) Per slot update:** For each time slot $x+k$ (where $k \in \{0, 1, \ldots, \tau_i(t)-1\}$), the remaining processing amount for task $J_i(t)$ is updated as:

$$R_{i,j} = R_{i,j} - \min\left(R_{i,j}, \frac{F_j \cdot \Delta}{B_j(x+k)}\right), \tag{2}$$

where F_j represents the computing capability of node e_j. After updating through multiple time slots, until $R_{i,j} \leq 0$, the task processing is completed, and the processing delay is $P_{i,j}(t) = (k+1) \cdot \Delta$.

3.1.4 | Delay Model

The total delay includes transmission delay, processing delay, and waiting delay, but excludes the delay of task offloading decisions, as this delay is negligible. The total delay $T_i(t)$ for completing task $J_i(t)$ can be defined according to the processing method of the task, as follows:

$$T_i(t) = \begin{cases} W_{i,j}(t) + P_{i,j}(t), & i = j, \\ \mathcal{W}_{i,j}(t) + T_{i,j}(t) + W_{i,j}(t) + P_{i,j}(t), & i \neq j, \end{cases} \tag{3}$$

where $W_{i,j}(t)$ represent the processing waiting times for task $J_i(t)$ at edge node e_j. $\mathcal{W}_{i,j}$ represent the transmission waiting times for task $J_i(t)$ from edge node e_i to e_j. $i = j$ indicates that the task is processed locally, $i \neq j$ indicates that the task is offloaded to other edge nodes. Furthermore, task $J_i(t)$ must be completed before its deadline, i.e., $T_i(t) \leq \tau_i(t)$. Otherwise, the task will expire and be dropped from the network.

3.2 | Problem Formulation

This paper focuses on maximizing the task success rate, by implementing continuous policy optimization, minimizing the possibility of offloading tasks to malicious nodes, and enhancing collaboration among edge nodes. The task success rate $\mathbb{S}$ is defined as follows:

$$\mathbb{S} = \frac{|J^{\text{succ}}|}{|J^{\text{succ}}| + |J^{\text{drop}}| + |J^{\text{fail}}|}, \tag{4}$$

where $|J^{\text{succ}}|$ represents the number of successfully processed tasks, $|J^{\text{drop}}|$ represents the number of tasks dropped due to not being completed within the deadline, and $|J^{\text{fail}}|$ represents the number of tasks that failed due to malicious behaviors.

4 | PROPOSED SCHEME

In this section, we first introduce the implementation of the BTRU algorithm. Then, we model the formulated optimization problem as a decentralized partially observable Markov decision process (DPOMDP). Finally, we present the implementation of the DTTO algorithm.

4.1 | Implementation of BTRU

This paper introduces a blockchain-based reputation mechanism in CEC to enhance the trustworthiness of collaboration, and the BTRU algorithm is proposed to improve the efficiency and accuracy of reputation updates. The BTRU algorithm is implemented in a smart contract and deployed in the network. Each node is initially assigned a moderate reputation value. Node initiating collaboration can provide positive or negative feedback based on task completion status and initiate a on-chain transaction. The leader collects transactions and executes the BTRU algorithm in the smart contract to calculate the new reputation values. Among them, if an edge node involved in collaboration completes the task within the deadline, it will receive positive feedback $\mu_j^u = 1$; malicious behaviors such as service refusal, delayed processing, or incorrect processing will receive negative feedback $\mu_j^u = 0$.

We give the basic structure of the BTRU algorithm. The positive feedback count α_j^u, negative feedback count β_j^u, and new reputation value R_j are computed as follows:

$$\begin{cases} \alpha_j^u = W_f \cdot \alpha_j^{u-1} + W_c \cdot W_d \cdot \mu_j^u, \\ \beta_j^u = W_f \cdot \beta_j^{u-1} + (1 - W_c) \cdot (1 - \mu_j^u), \\ R_j = E(x) = \frac{\alpha_j^u}{\alpha_j^u + \beta_j^u}, \end{cases} \tag{5}$$

where u and $u-1$ denote the steps of reputation updates. $W_f \in [0, 1]$ is the forgetting factor. In the dynamically changing CEC environment, the forgetting factor ensures that the system gradually diminishes the influence of historical behaviors, thereby enhancing the real-time adaptability of the reputation value. $W_c \in [0, 1]$ is the historical task completion rate factor, designed to prevent malicious nodes from rapidly recovering their reputation by occasionally processing tasks correctly. $W_d \in [0, 1]$ is the task difficulty factor, which reflects the computational difficulty of the task. If an edge server completes a complex task, its reputation increases more significantly; conversely, if an edge node fails to complete even a simple task, its reputation should be substantially reduced. The task difficulty factor W_d is defined as $W_d = (s_i(t) \cdot \rho_i(t))/(s_{\max} \cdot \rho_{\max})$. Where $s_{\max}$ and $\rho_{\max}$ represent the maximum task size and processing density in the system, respectively. The inclusion of factors W_c and W_d extends the multidimensionality of the reputation update algorithm, equivalent to adding trust indicators based on historical performance and task difficulty to nodes. This makes the BTRU algorithm more robust and able to tolerate occasional errors from honest but low-quality nodes, while still effectively identifying nodes with malicious behaviors.

Finally, we briefly analyzed the expected overhead and reliability impact of using blockchain. Although the introduction of blockchain in this system has brought some overhead, it remains overall controllable. In terms of computational and communication overhead, the smart contract is designed to perform lightweight numerical computations to obtain the new reputation value. This operation is triggered by transactions initiated by edge nodes and ultimately determined through a consensus algorithm. By avoiding complex on-chain calculations, the computational overhead of smart contract execution is minimal. Communication delay primarily occurs during the consensus phase, and by choosing a lightweight consensus mechanism, it can be kept controllable. In terms of reliability, blockchain provides an immutable record of critical state changes,

such as task completion status and reputation updates. Moreover, the consensus protocol and on-chain history further ensure trust consistency across the entire system.

4.2 | DPOMDP Modeling

The optimization problem formulated in Section 3.2 is formalized as a DPOMDP. Specifically, It can be represented by a tuple $\langle \mathcal{N}, \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \mathcal{O}, \rho, \gamma \rangle$. Here, $\mathcal{N} = \{e_1, \ldots, e_N\}$ represents the set of N agents, each corresponding to an edge node. $\mathcal{S}$ is the state space, where $s_t \in S$ denotes the environment state at time slot t. $\mathcal{A}$ is the action space, where each agent $e_i \in \mathcal{N}$ selects an action $a_t^i \in \mathcal{A}$ at time slot t, forming a joint action $a_t = \{a_t^1, \ldots, a_t^i, \ldots, a_t^N\}$. $\mathcal{P}$ is the state transition function, where $\mathcal{P}(s_{t+1}|s_t, a_t)$ represents the probability that the environment transitions from state s_t to s_{t+1} after all agents execute the joint action a_t. r is the reward function, where after a state transition, each agent $e_i \in \mathcal{N}$ receives a reward $r_t^i = r(s_t, a_t)$. $\mathcal{O}$ is the observation space, where each agent $e_i \in \mathcal{N}$ in the partially observable scenario can only observe its own local information $o_t^i \in \mathcal{O}$ and selects an action $a_t^i \sim \pi_{\theta_i}(\cdot|o_t^i)$ based on a decentralized policy. ρ is the initial state distribution, where the initial state s_0 is drawn from distribution ρ. $\gamma \in [0, 1]$ is the discount factor, which balances the importance of immediate and future rewards.

In the DPOMDP, the local observation of an agent reflects the local edge network information, and the action taken by the agent corresponds to the task offloading decision of the edge node. The trustworthiness of task offloading and whether the task is successfully processed determine the immediate reward. The detailed definition of the DPOMDP is as follows.

4.2.1 | Observation

At time slot t, each edge node e_i observes its local information, including task size, task processing density, task arrival probability, computation capability, transmission rate with other edge nodes, number of active queues, and historical reputation data. Therefore, the observation of edge node e_i at time slot t is defined as:

$$o_t^i = \{s_i(t), \rho_i(t), \lambda_i(t), F_i, r_{i,j}(t), B_i(t), (H(t-p+1), \ldots, H(t))\}. \tag{6}$$

Where $(H(t-p+1), \ldots, H(t))$ is a $p \times N$ matrix representing the historical reputation data over the last p steps. Additionally, each element in the observation o_t^i should be divided by its maximum value to ensure that all elements are within the range $[0, 1]$, thereby reducing the impact on the stability and convergence.

4.2.2 | Action

When task $J_i(t)$ arrives, edge node e_i selects the appropriate node to handle the task based on the current observation. Therefore, the action space should include all computing resources. The action space is defined as $\mathcal{A} = \{a_0, \ldots, a_i, \ldots, a_N\}$. Where action a_0 indicates that no task has been received by the edge node, and actions $a_1, \ldots, a_N$ represent the processing of tasks by different edge nodes. Note that if edge node e_i chooses action a_i, it means the task will be processed locally. When the edge node does not need to make an offloading decision because no task has been received, it can select action a_0 by masking other actions.

4.2.3 | Reward

The immediate reward should comprehensively consider the trustworthiness of the task offloading decision and whether the task processing was successful. Thus, if task $J_i(t)$ is dropped due to exceeding the deadline or fails due to malicious behavior, a negative reward of –1 is assigned. If task $J_i(t)$ is successfully processed and correctly submitted within the deadline, a positive reward of $1 + R_j$ is given, where R_j is the reputation value of the target edge node. This reward design encourages the agent to balance high trustworthiness with high success rates. If the agent overly focuses on highly trusted offloading decisions, it may lead to overloading high-reputation edge nodes, resulting in more tasks being dropped.

4.3 | Implementation of DTTO

To learn the optimal task offloading policy within the formulated DPOMDP, we propose the DTTO algorithm, whose framework is illustrated in Figure 2. Specifically, we let the actor network π_{θ_i} parameterized by θ_i to model the approximate offloading policy for each edge node $e_i \in \mathcal{N}$, let the critic network V_{ω_i} parameterized by ω_i to model the approximate value function for each edge node $e_i \in \mathcal{N}$. The observation information is fed into both neural networks π_{θ_i} and V_{ω_i} through an input layer. Next, a long short-term memory (LSTM) layer is used to predict the reputation values of nodes shortly based on historical reputation data. After that, two fully-connected (FC) layers are employed to learn the mapping from observations to actions and the mapping from observations to value functions, respectively. By incorporating historical reputation data as part of the observation input, each edge node $e_i \in \mathcal{N}$ can better understand the dynamic changes of the reputation mechanism, thereby making more optimal offloading decisions.

To optimize the network parameters and obtain the optimal task offloading policy, we adopt the proximal policy optimization (PPO) algorithm[35]. Let D_i represent the rollout buffer of edge node $e_i \in \mathcal{N}$, which stores the interaction data between the node and the environment in the form of $(o_t^i, a_t^i, r_t^i, o_{t+1}^i)$. To avoid a large update to the offloading policy, PPO uses a clipped surrogate objective function, defined as:

$$J_i^{\text{CLIP}}(\theta_i) = \mathbb{E}_{(o_t^i, a_t^i, r_t^i, o_{t+1}^i) \in D_i}[\sum_{t=1}^{T} \min(Pr_t^i \hat{A}_t^i, \mu(Pr_t^i)\hat{A}_t^i)], \tag{7}$$

where $Pr_t^i = \pi_{\theta_i}(a_t^i|o_t^i)/\pi_{\theta_i^{(\text{old})}}(a_t^i|o_t^i)$ is the probability ratio between the new policy and the old policy, and $\mu(x)$ is the clipping function, defined as:

$$\mu(x) = \begin{cases} 1+\epsilon, & x > 1+\epsilon, \\ x, & 1-\epsilon \le x \le 1+\epsilon, \\ 1-\epsilon, & x < 1-\epsilon, \end{cases} \tag{8}$$

where ϵ is a tunable clipping factor. $\hat{A}_t^i$ is the advantage function. To balance the bias of the value estimate and the variance of the returns, generalized advantage estimation (GAE) is used as the advantage function, defined as:

$$\hat{A}_t^i = \sum_{k=0}^{T-t} \left((\gamma\lambda)^k \left(r_{t+k} + \gamma V_{\omega_i^{(\text{old})}}(o_{t+k+1}^i) - V_{\omega_i^{(\text{old})}}(o_{t+k}^i) \right) \right), \tag{9}$$

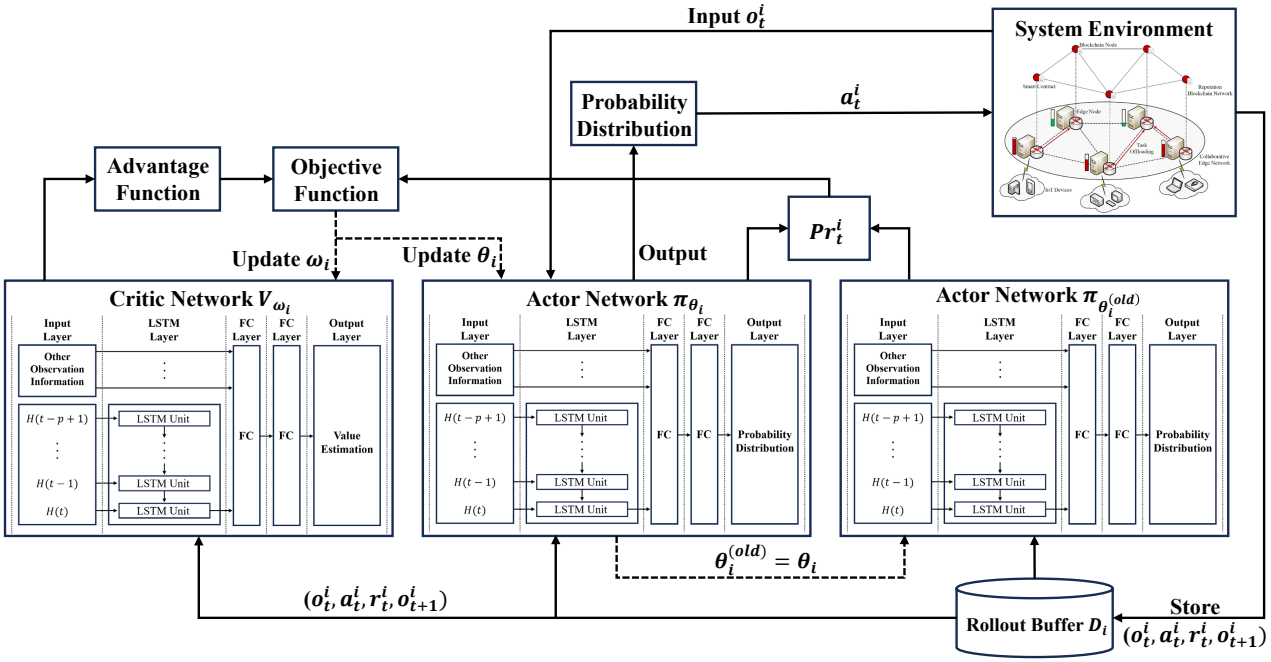


FIGURE 2 The framework of DTTO algorithm at edge node $e_i \in \mathcal{N}$.

Algorithm 1 Training of DTTO at Each Node $e_i \in \mathcal{N}$

Input: Heterogeneous blockchain-supported CEC system.
Initialize the rollout buffer D_i.
Initialize the actor network π_{θ_i} and critic network V_{ω_i}.
for iterations $k_1 \in \{1, \cdots, K\}$ **do**
  Reset environment.
  for time slot $t \in \{1, \ldots, T\}$ **do**
    Edge node e_i observes local environment information and traces back a total of p steps of historical reputation data from the past to the present, forming an observation o_t^i.
    Draws a task offloading decision a_t^i from the actor network π_{θ_i}.
    if $a_t^i = a_0$ **then**
      No task.
    else if $a_t^i = a_i$ **then**
      The task $J_i(t)$ is processed locally on node e_i.
    else
      The task $J_i(t)$ is offloaded to node e_j for processing.
    end if
    Executes action a_t^i, obtains observation o_{t+1}^i.
    Determine the completion status of the task $J_i(t)$. If the task was offloaded, the node initiating collaboration send a transaction based on this completion status to update the reputation of the target node.
    Receive reward r_t^i.
    Store $(o_t^i, a_t^i, r_t^i, o_{t+1}^i)$ in the rollout buffer D_i.
    if buffer size > training threshold **then**
      for iterations $k_2 \in \{1, \ldots, \mathcal{K}\}$ **do**
        Update the actor network π_{θ_i} and the critic network V_{ω_i} by mini-batch gradient update based on D_i via Eq. (12) and Eq. (13).
      end for
    end if
  end for
end for
Output: The trained actor network π_{θ_i} and critic network V_{ω_i}.

where $\lambda \in [0, 1]$ is the bias-variance trade-off factor. To optimize the value function, the mean squared error loss is introduced:

$$J_i^{\mathrm{VF}}(\omega_i) = \mathbb{E}_{(o_t^i, a_t^i, r_t^i, o_{t+1}^i) \in D_i} \left[\sum_{t=1}^{T} \left(V_{\omega_i}(o_t^i) - R_t^i \right)^2 \right], \tag{10}$$

where $R_t^i = \sum_{k=0}^{T-t} \gamma^k r_{t+k}$ represents the cumulative return. To increase the exploration capability of the policy, an entropy regularization term is introduced:

$$J_i^{\mathrm{ENT}}(\theta_i) = \mathbb{E}_{(o_t^i, a_t^i, r_t^i, o_{t+1}^i) \in D_i} \left[\mathcal{H} \left(\pi_{\theta_i}(\cdot | o_t^i) \right) \right], \tag{11}$$

where $\mathcal{H}$ is the entropy of the policy π_{θ_i}. Based on Eq. (7), Eq. (10), and Eq. (11), the update of the actor and critic network parameters are as follows:

$$\theta_i \leftarrow \theta_i + \alpha \left(\nabla_{\theta_i} J_i^{\mathrm{CLIP}}(\theta_i) + c \nabla_{\theta_i} J_i^{\mathrm{ENT}}(\theta_i) \right), \tag{12}$$

$$\omega_i \leftarrow \omega_i + \beta \nabla_{\omega_i} J_i^{\mathrm{VF}}(\omega_i), \tag{13}$$

where c is the weight coefficient for the entropy regularization. α and β represent the learning rates for the actor network and the critic network, respectively.

The process of training the DTTO algorithm on each edge node $\theta_i \in \mathcal{N}$ is summarized in Algorithm 1. First, initialize the parameters θ_i and ω_i of the actor and critic networks. Then, node e_i interacts with the environment, storing experience data in the

rollout buffer D_i. Once the data in the buffer is sufficient and exceeds the training threshold, the network models are updated. The complexity of the algorithm can be approximately represented as $O(KTM + LB\mathcal{K}M)$. Here, K is the number of episodes, T is the maximum time slot index, M is the total number of parameters in the neural network, L is the number of times training occurs once the buffer data reaches the threshold, B is the batch size, and $\mathcal{K}$ is the number of optimization steps (i.e., hyperparameter K_{epochs}). $O(KTM)$ represents the complexity of the interaction phase, as each episode contains T time slots and the complexity of each decision (forward propagation) in each time slot is $O(M)$. $O(LB\mathcal{K}M)$ represents the complexity of the training phase. After the buffer data reaches the threshold, training is conducted for $\mathcal{K}$ rounds, with each round using a batch size B for gradient updates.

5 | A KUBERNETES AND ETHEREUM-BASED TESTBED FOR CEC TASK OFFLOADING

Based on Kubernetes and Ethereum, this paper designs a testbed to test the performance of the framework and strategies in real-world applications.

5.1 | Motivation

Most related works conduct preliminary and rapid evaluations of their proposed methods using Python and related simulation frameworks. However, this evaluation approach suffers from several inherent limitations: 1) Each edge node is represented merely as a program instance, making it impossible to deploy and test real applications on edge nodes. 2) Communication between nodes is implemented via function calls within the simulation environment rather than real network protocols, making it difficult to thoroughly assess the stability and timeliness of task offloading under real network conditions. 3)There is a lack of a genuine blockchain environment to support the execution of the proposed algorithms.

To overcome these limitations and further verify the practicality and applicability of the proposed DTTO algorithm, we developed a Kubernetes and Ethereum-based testbed for CEC task offloading. In this testbed, we employed containerization technologies to deploy multiple simulated edge nodes on a single physical server. Each node has its own isolated computing and network space, and the nodes communicate with each other using real TCP/IP protocols, thereby replicating realistic network interactions in the CEC environment. In addition, smart contracts were implemented using the Solidity language and successfully deployed onto a private Ethereum blockchain. Each edge node owns an Ethereum account, enabling authentic interactions and trust management under a real blockchain environment, which provides strong support for the proposed trustworthy offloading strategy. The testbed is open-sourced on GitHub via the link: https://github.com/ygy45ns/Kubernetes-and-Ethereum-based-Testbed.

5.2 | Implementation of Testbed

Figure 3 illustrates the system architecture of the task offloading testbed. The overall architecture consists of four components: the Experiment layer, the Controller layer, the Processing layer, and the Middleware. All modules are uniformly deployed on the Kubernetes platform with Docker containers as the runtime environment. This architecture exhibits good modularity and scalability, supporting rapid integration and evaluation of different offloading algorithms. The testbed implementation is primarily written in Java, comprising approximately 8,500 lines of code.

5.2.1 | Experimental Layer

The Experimental layer serves as the top layer of the testbed, undertaking the management and scheduling responsibilities for the entire experiment lifecycle. This module runs as a Pod in Kubernetes and is responsible for initiating experimental processes, configuring experimental parameters, initializing the Controller and Processing layers, and performing persistence and visualization of experimental results. Users can specify task offloading algorithms at this layer and trigger processes such as model training, policy testing, and data visualization.

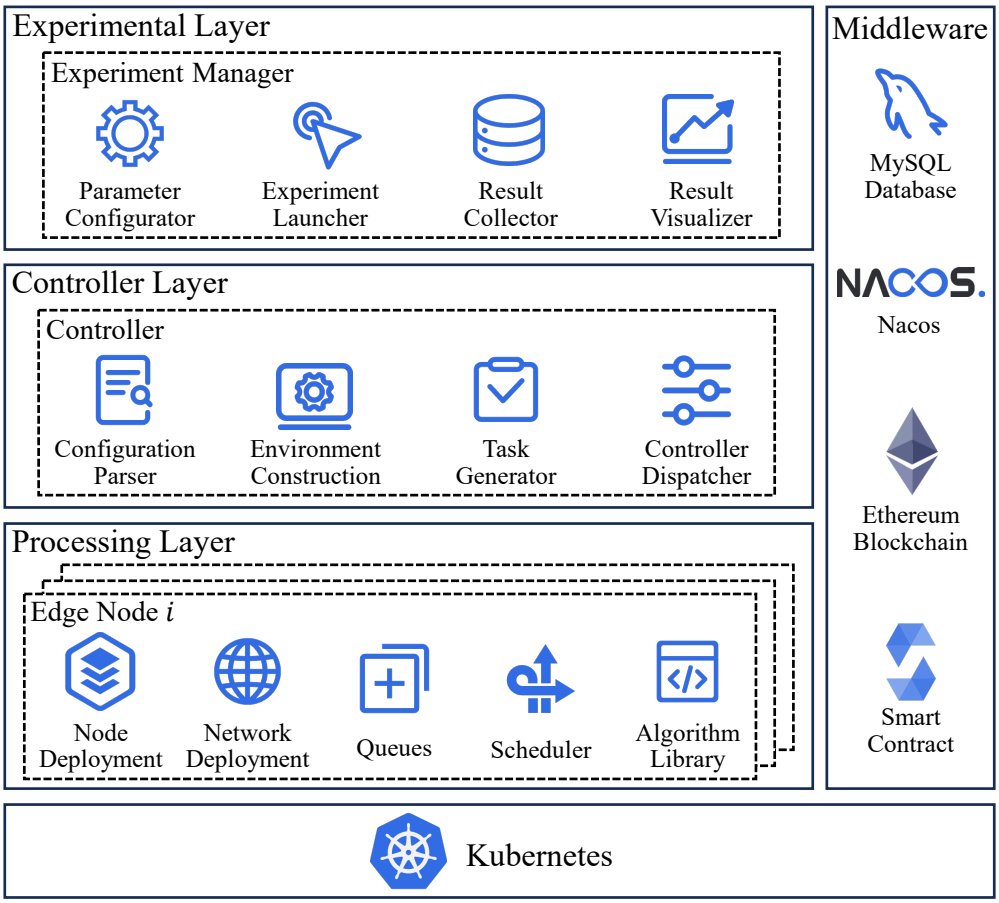


FIGURE 3 System architecture of the Kubernetes and Ethereum-based testbed for CEC task offloading.

5.2.2 | Controller Layer

The Controller layer is positioned below the Experimental layer and serves as a coordinator between numerous IoT devices and edge nodes. It also runs as a Pod and its primary functions include network environment construction, task generation simulation, configuration file parsing, and control signal distribution. The Controller layer supports users in flexibly configuring experimental parameters such as node information, node topology, and task scale through configuration files, with the system automatically parsing and completing node and network deployment. Its built-in task generator can dynamically generate task requests based on task configurations, simulating large-scale IoT device behavior, and send tasks to edge nodes, allowing nodes to autonomously decide whether to perform collaborative offloading.

5.2.3 | Processing Layer

The Processing layer is the core of the testbed, featuring multiple edge nodes. Each edge node runs as an independent Pod with dedicated computing and network spaces. When setting up edge nodes, the network deployment and node deployment modules are respectively responsible for reading node and network information from the database to complete the corresponding deployment tasks. Each edge node internally contains one processing queue and $N - 1$ transmission queues. The Scheduler module is responsible for making task offloading decisions and can flexibly switch between heuristic methods and DRL methods according to experimental settings. An algorithm library is configured within each node to provide internal algorithms for the Scheduler module. The algorithms in the library include DTTO, DDQN, SAC, etc., where the DRL algorithms are implemented based on the Deep Java Library.

5.2.4 | Middleware

The Middleware provides service management, data storage, and reputation management for the entire system. The testbed employs Nacos to implement a service registration and configuration center, supporting registration, discovery, and dynamic configuration updates for Experiment, Controller, and Edge Node services. The MySQL database is used to store node information, network topology, and task logs. To achieve trustworthy task offloading, we introduce an Ethereum-based blockchain component, deploying a private chain and implementing the BTRU algorithm through smart contracts written in Solidity. Each edge node possesses an independent blockchain account and can invoke smart contracts to complete reputation updates and queries.

6 | PERFORMANCE EVALUATIONS

In this section, we conduct a set of simulation-based experiments to rapidly verify and evaluate the effectiveness of the proposed solution. Specifically, we first introduce the simulation setup and multiple baseline algorithms. Then, a series of experiments is performed to compare the performance of our algorithm against these baselines under various settings. All experiments were carried out on a computer equipped with an Intel Core i7-12700KF CPU @3.60 GHz running a 64-bit operating system.

6.1 | Simulation Setup

We implemented a CEC simulator in Python. Specifically, we randomly select 20 nearby base station locations from the EUA dataset[36] as edge nodes and use a flexible measurement-based modeling generator[37] to obtain an edge node model for the target area that matches the distribution patterns of real datasets (including attributes such as disk, memory, and bandwidth). Moreover, the default parameters of the environment are as follows. The total number of time slots is 100, and each time slot is 0.5s[4]. The proportion of malicious edge nodes is 20%. Two typical trust attack models are introduced to evaluate the robustness of the system: on-off attack and bad-mouthing attack[38]. Malicious nodes launch combined attacks using both methods, making the behavior more deceptive and destructive.

The task arrival probability for each edge node in each time slot follows a uniform distribution on [0, 1], the task size for each edge node in each time slot follows a uniform distribution on [1000, 8000] KB, and the task processing density for each edge node in each time slot follows a uniform distribution on [800, 2400] cycles/bit[4,15]. The deadline for the task is 4s. The number of CPU cores for each edge node is randomly selected from $\{16, 20, 24, 28, 32\}$, and the CPU frequency of each core is 3 GHz[4,39].

PyTorch is used for building and training the networks. The dimension of the LSTM layer is 20, and the time step p of the input sequence is 20. The dimensions of the two FC layers are both 128. The learning rates α and β are 0.0003 and 0.0005, respectively. The discount factor γ is 0.9, the clipping factor ϵ is 0.2, the bias-variance trade-off factor λ is 0.9, and the entropy coefficient c is 0.01. The number of training iterations K_{epochs} is 4, and the batch size is 64.

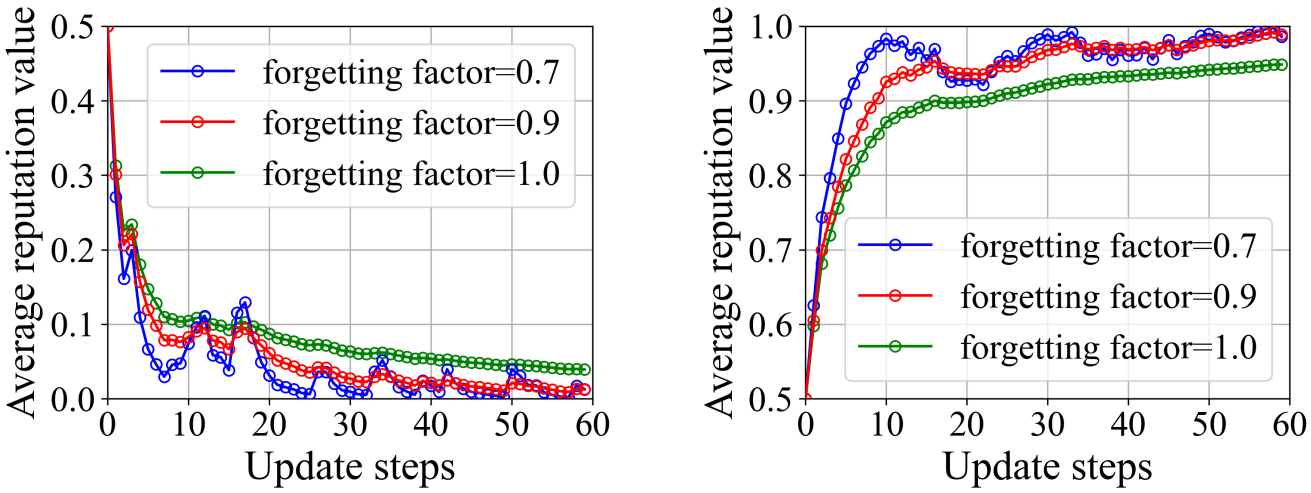
6.2 | Baseline Algorithms

To evaluate the performance of the proposed BTRU and DTTO algorithms, we select outstanding works in the related field as baseline algorithms for comparison.

6.2.1 | Baselines for Reputation Update

In terms of reputation value updates, we compare BTRU with the following two representative reputation mechanisms.

- BTRES[40]: A reputation evaluation system based on the beta distribution, which monitors node behavior and uses the beta distribution to describe node reputation distribution, thereby calculating trust values to guide interactions between nodes.
- MLRMD[41]: It combines machine learning and reputation mechanisms to achieve misbehavior detection in vehicular communication networks. Its reputation mechanism uses Dempster-Shafer theory for feedback aggregation and beta distribution for reputation value updates.



(a) Malicious node.

(b) Honest node.

FIGURE 4 Variation curves of average reputation value in the BTRU algorithm under different forgetting factors W_f.

For the reputation value update problem, we evaluate and compare the performance of different reputation update algorithms mainly through the reputation value changes of edge nodes. The average reputation values of honest nodes and malicious nodes serve as the two key performance metrics.

6.2.2 | Baselines for Task Offloading

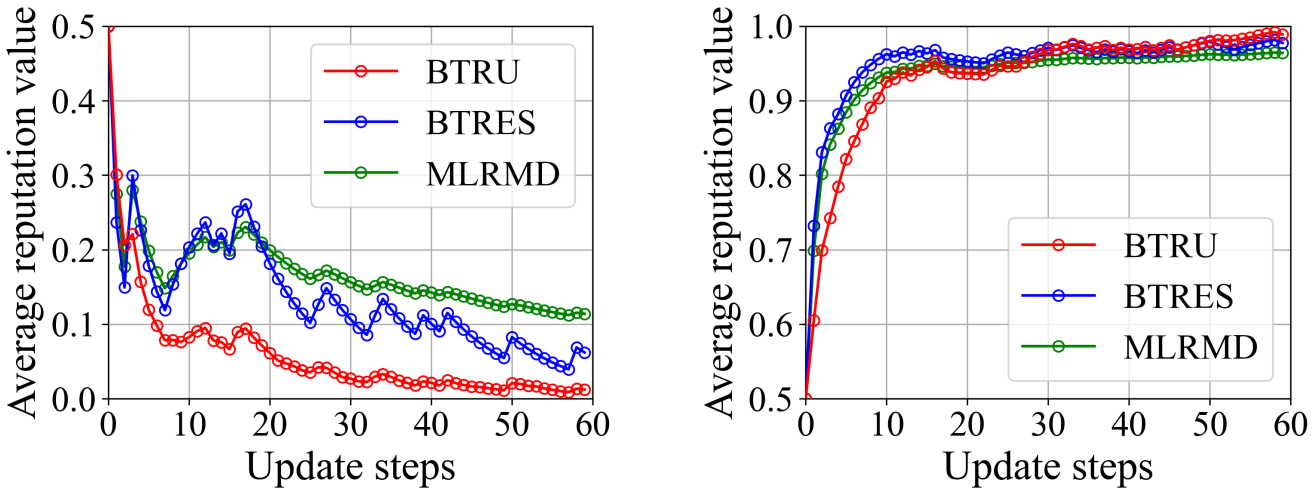
In terms of task offloading, we compare the DTTO algorithm with several excellent offloading algorithms.

- LOCAL[7]: A non-cooperative strategy where tasks are processed locally upon arrival at the edge node. This algorithm demonstrates system performance in the absence of cooperation or offloading mechanisms.
- BRMTO[21]: A blockchain-based task offloading algorithm that selects the top k candidate nodes based on reputation values maintained in a blockchain ledger and then chooses the node with the shortest queue time as the target computing node.
- DDQN[24]: This algorithm effectively reduces potential overestimation bias in the action-value function through a double-network structure. Combined with an experience replay mechanism, it continuously optimizes the task offloading policy by leveraging past experiences.
- SAC[4]: This algorithm is based on the maximum entropy framework, which encourages the policy to maintain a certain level of randomness while optimizing task offloading performance, thereby enhancing exploration capabilities and robustness in decision-making.

For the task offloading problem, we primarily consider the task success rate as the performance metric. We conducted ten trials for each algorithm and plotted the average performance metrics in the figures.

6.3 | Performance Comparison of BTRU Algorithm

Figure 4 illustrates the variations in the average reputation values of honest and malicious nodes under different forgetting factors W_f in the BTRU algorithm. Although the reputation value curves exhibit similar overall trends across different forgetting factors, their subtle variations differ. When the forgetting factor is set to 0.9, the algorithm achieves the best balance. Not only can it quickly distinguish between honest and malicious nodes, but it can also ensure stable reputation updates. This is because a larger forgetting factor makes reputation evaluation overly dependent on historical behaviors, preventing timely reflection of short-term behavioral changes, while a smaller forgetting factor results in significant fluctuations in average reputation values due to the occasional behavior of a few nodes.



(a) Malicious node.

(b) Honest node.

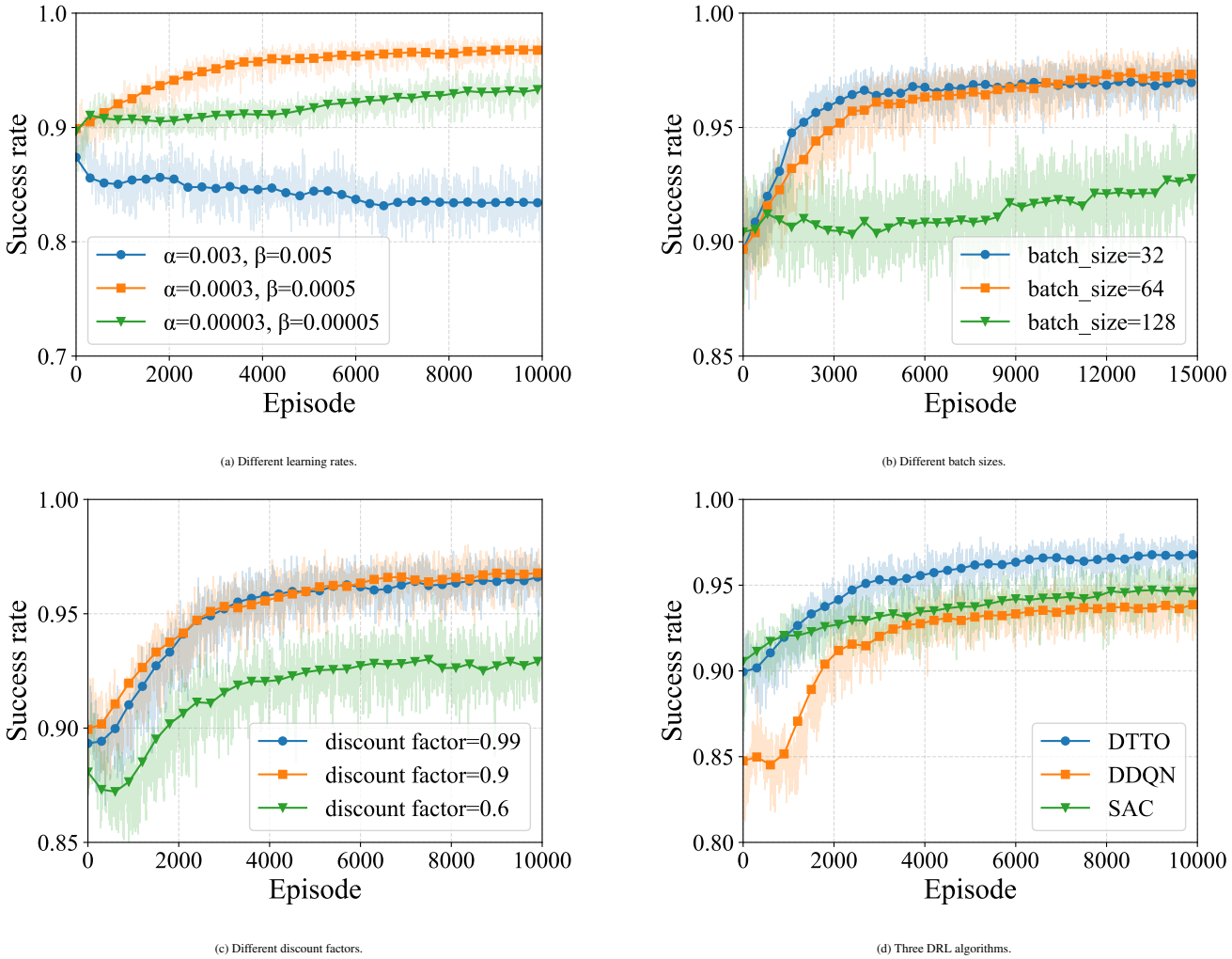
FIGURE 5 Variation curves of average reputation value in the BTRU, BTRES, and MLRMD algorithms.

Next, we compare the BTRU algorithm with two baseline algorithms, with the results shown in Figure 5. It can be observed that the BTRU algorithm exhibits significant advantages in both the efficiency and accuracy of reputation updates. In the reputation value changes of malicious nodes (Figure 5 (a)), the BTRU algorithm outperforms other algorithms by reducing the reputation values of malicious nodes more quickly and stably. The MLRMD algorithm shows a smaller decline in the average reputation value of malicious nodes, making it less effective in distinguishing malicious nodes. While the BTRES algorithm can significantly reduce the reputation values of malicious nodes, it is affected by occasional normal behaviors of these nodes, leading to substantial fluctuations in reputation values that compromise system stability. In the reputation value changes of honest nodes (Figure 5 (b)), the BTRU algorithm exhibits slower initial growth of reputation value due to its multi-factor consideration mechanism. However, it ultimately surpassed other algorithms, and the average reputation value of honest nodes increased by at least 1.15%.

6.4 | Convergence Analysis of DTTO Algorithm

We analyzed the impact of several hyperparameters on the convergence of the DTTO algorithm in the default CEC environment. In Figure 6 (a), we study the impact of different learning rates on the DTTO algorithm. The results indicate that setting the learning rates of the actor and critic networks to 0.0003 and 0.0005, respectively, can achieve faster convergence and a higher success rate. In Figure 6 (b), different batch sizes exhibit slight differences in performance. When the batch size is 64, the algorithm achieves a good balance between convergence speed and success rate, achieving the best success rate in the later stage. In Figure 6 (c), we investigate the impact of different discount factors on the DTTO algorithm. The discount factor quantifies the algorithm's temporal preference for future rewards relative to immediate returns. The results suggest that setting the discount factor to 0.9 is the optimal choice, balancing immediate rewards with long-term reliability in task offloading.

Moreover, we present the convergence results for three DRL-based algorithms in Figure 6 (d). The DTTO algorithm exhibits superior performance in both convergence speed and task success rate, with smaller fluctuations. This is attributed to the integration of the reputation mechanism to guide agents in making trustworthy task offloading decisions, the use of entropy regularization to encourage exploration, the adoption of GAE to balance the bias and variance, and the implementation of gradient clipping to strictly limit the range of policy updates. In contrast, the other two DRL-based baseline algorithms perform less effectively. On the one hand, the DDQN algorithm is based on a greedy strategy that struggles to explore a globally optimal task offloading policy in a complex and dynamic CEC environment. On the other hand, both DDQN and SAC algorithms depend on experience replay buffers, requiring extensive exploration and data collection in the early stages, which limits training efficiency and convergence speed. Moreover, they lack relevant mechanisms to drive models to make trustworthy decisions, resulting in many tasks being offloaded to malicious nodes.



(a) Different learning rates.

(b) Different batch sizes.

(c) Different discount factors.

(d) Three DRL algorithms.

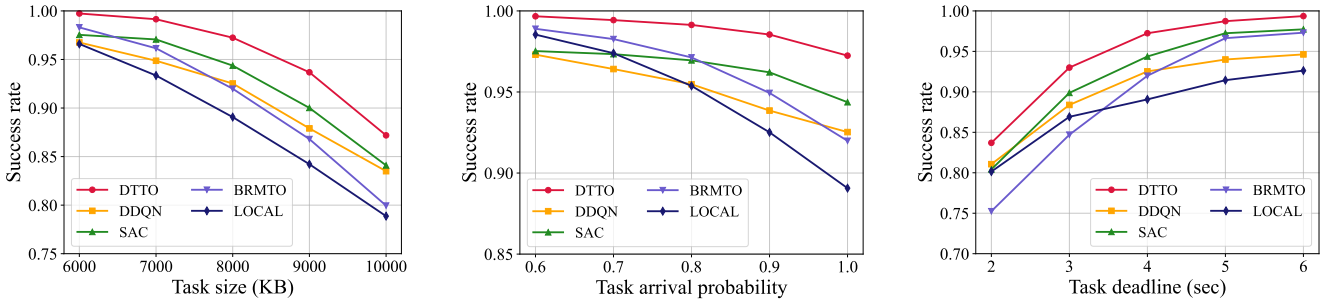
FIGURE 6 Convergence under different hyperparameters

6.5 | Performance Comparison of DTTO Algorithm

In this subsection, we compare the performance of all baseline algorithms under different system parameters in the default CEC environment.

First, we investigate the impact of different task sizes on the task success rate, where task size refers to the maximum size of generated tasks. As shown in Figure 7 (a), the task success rate of all algorithms decreases significantly as task size increases. Nevertheless, the DTTO algorithm consistently achieves the highest success rate for any task size. Compared with other baseline algorithms, DTTO improves the success rate by at least 3.68% for larger task sizes. This indicates that the DTTO algorithm is more adaptable to changes in task size.

Then, we examine the performance of the DTTO algorithm compared to other baseline algorithms under different task arrival rates. As shown in Figure 7 (b), the task success rate of all algorithms decreases as the task arrival rate increases, but the DTTO algorithm exhibits a significant advantage over the baseline algorithms. When the task arrival probability is low (i.e., below 0.8), all algorithms maintain relatively good performance, as the lower workload allows most tasks to be processed locally, reducing the need for collaboration among edge nodes. However, when the task arrival probability increases to 0.9 or 1.0, only the DTTO algorithm effectively mitigates the negative impact of the surge in workload. Compared with the four baseline algorithms, the DTTO algorithm improves the success rate by at least 3.04%. This is because the DTTO algorithm not only minimizes the offloading of tasks to malicious nodes but also better coordinates and optimizes the utilization of other computing



(a) Impact of task sizes.

(b) Impact of arrival probabilities.

(c) Impact of task deadlines.

FIGURE 7 Success rate of tasks under different system parameters in the simulation.

resources, thereby efficiently responding to the surge in task requests and ensuring the system's performance stability under high concurrency.

Next, we study the impact of different task deadlines on task success rates. As shown in Figure 7 (c), the success rate of all algorithms increases significantly as the task deadline extends. This is because longer deadlines allow tasks that require extended processing and transmission times to be completed. The DTTO algorithm consistently achieves the highest task success rate under all deadline settings. Particularly under strict time constraints, it outperforms other baseline algorithms, improving the success rate by at least 5.01%. This demonstrates that the DTTO algorithm can adapt to the ultra-dense environment with strict deadlines.

Finally, we analyze the impact of the proportion of malicious edge nodes on task success rates, as shown in Figure 8. As the proportion of malicious nodes increases, all algorithms experience a decline in task success rates. A higher proportion of malicious nodes means fewer trusted resources in the system, causing more tasks to exceed their deadlines due to resource shortages. Nevertheless, the DTTO algorithm consistently maintains a task success rate above 95% under different proportions of malicious nodes, demonstrating its robustness and adaptability in the dynamic adversarial environment. Furthermore, when the proportion of malicious nodes reaches 20% or higher, the success rate of the BRMTO algorithm surpasses that of the DDQN algorithm, and when the proportion exceeds 40%, the BTMTO algorithm also outperforms the SAC algorithm. This indicates that when malicious nodes are relatively few, optimizing the competition and collaboration of computing resources is more important, while as the number of malicious nodes increases, avoiding task offloading to these nodes is more critical.

In summary, different algorithms exhibit distinct performance variations under various system parameter settings. The LOCAL algorithm consistently performs the worst under most system parameters, because it is a non-collaborative strategy and is difficult to handle intensive workloads. Similarly, the performance of the BRMTO algorithm is also relatively weak. The main reason is that it relies on a fixed selection strategy and lacks the necessary adaptability and flexibility to coordinate resources in complex environments. The DDQN and SAC algorithms perform well in most cases, but their performance drops sharply when the proportion of malicious nodes exceeds 30%, because they lack the protection of a reputation mechanism. In contrast, the reputation-driven DTTO algorithm consistently ensures highly trustworthy task offloading and achieves the highest task success rates under various system parameters.

7 | EXPERIMENT IN TESTBED

We conducted additional experiments on the developed testbed to further validate the practicality and applicability of the proposed solution in a realistic environment. To enable rapid evaluation on the testbed, the number of edge nodes is 10. All other parameters remained consistent with the simulation settings. The testbed runs on a server. The CPU configuration of the server is Intel(R) Xeon(R) Gold 6133, which has 20 cores and 40 threads. The memory of the server is 256GB.

7.1 | Convergence analysis

Figure 9 depicts the task success rate learning curves of DTTO and four baseline algorithms (DDQN, SAC, BRMTO, and LOCAL) during training on our Kubernetes and Ethereum-based CEC task offloading testbed. It can be observed that DTTO

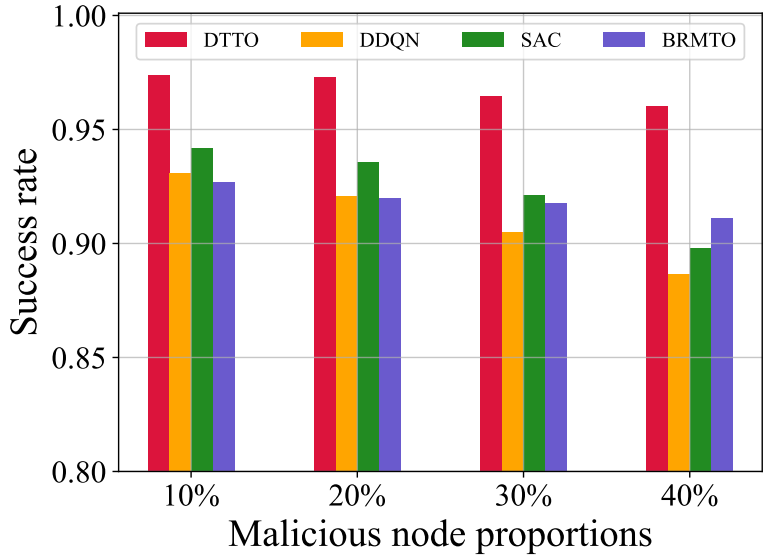


FIGURE 8 Success rate of tasks under different malicious edge node proportions.

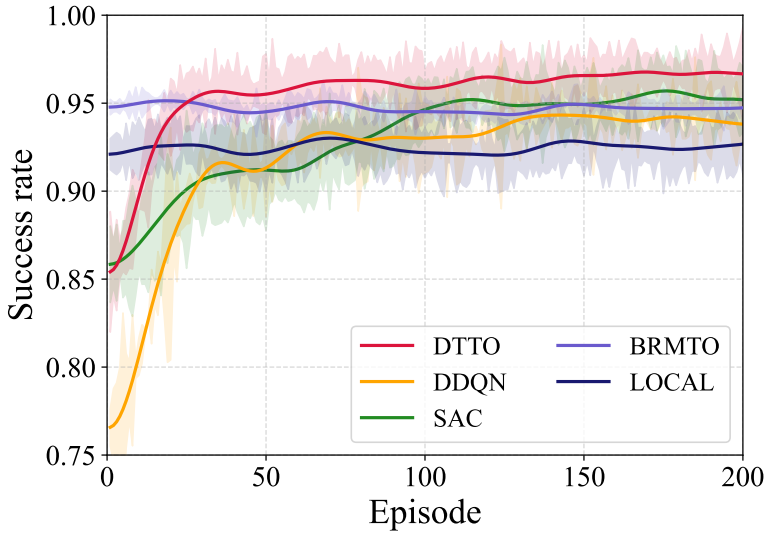


FIGURE 9 Learning curve of task success rate in the testbed.

rapidly improves its task success rate during early training and reaches a stable state after approximately 50 episodes, ultimately achieving a success rate of around 0.97. This significantly outperforms the baselines, demonstrating the effectiveness of DTTO in real network environments. Among the baselines, BRMTO and LOCAL exhibit nearly static performance due to their inability to learn from the environment. BRMTO, though trust-aware, lacks adaptability and fails to effectively coordinate and utilize system resources. Both DDQN and SAC demonstrate good convergence trends, yet their success rates and overall stability are inferior to DTTO. This is mainly attributed to the absence of a trust mechanism to guide learning toward reliable offloading decisions, leading them to occasionally offload tasks to untrustworthy nodes.

In summary, DTTO achieves trustworthy task offloading while effectively coordinating and utilizing system resources, both of which contribute to its high task success rate.

7.2 | Performance comparison

We further evaluated the performance of all algorithms under different system parameters on the developed testbed, as shown in Figure 10. The experimental results are consistent with the simulation outcomes, showing that the DTTO algorithm outperforms all baseline algorithms in terms of task success rate across different system conditions.

In Figure10 (a), the task success rate of all schemes decreases as task size increases, due to increased transmission and processing burden. Nevertheless, DTTO maintains a relatively stable performance compared with other baselines. In Figure10 (b), as the task arrival probability increases, the success rates gradually decline under heavier workloads, and DTTO demonstrates better adaptability. In Figure 10 (c), longer task deadlines improve the success rates for all schemes; DTTO performs competitively across the entire deadline range. To further quantify the performance gains, we calculate the relative improvements of DTTO over the best-performing baselines under each setting. DTTO achieves at least 1.33%, 1.57%, and 1.85% higher task success rates compared to the second-best algorithm under varying task deadlines, arrival probabilities, and task sizes, respectively. These improvements can be attributed to the adaptability of DTTO to dynamic environments and the integration of its reputation mechanism, effectively avoiding task offloading to unreliable or malicious nodes. These improvements are observed consistently across different parameter ranges, indicating the robustness of DTTO under diverse system conditions.

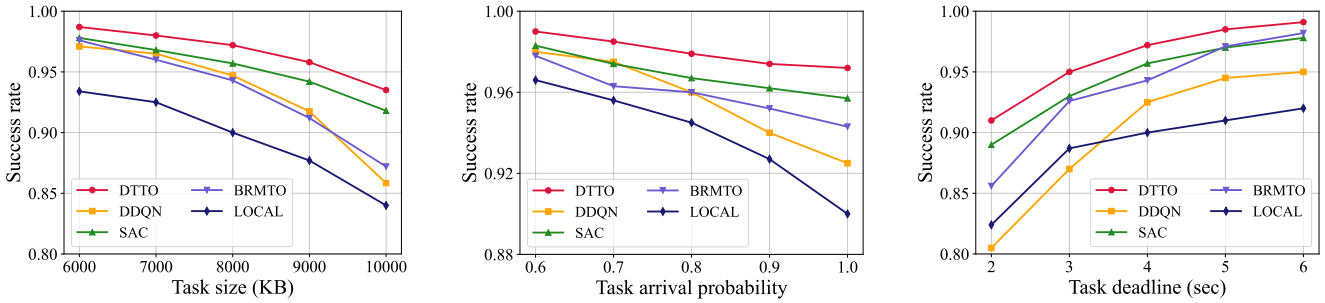
Unlike controlled simulation environments, the testbed experiments involve real-world uncertainties, such as dynamic network conditions, resource fluctuations, and blockchain transaction confirmation latency. The performance of DTTO under these realistic settings demonstrates its practicality and applicability in complex and dynamic CEC environments.

7.3 | Blockchain Overhead Evaluation

In the experiment spanning 200 episodes, approximately 57,000 tasks were generated, involving 200 reputation initialization transactions and about 30,800 reputation update transactions. The average confirmation latency for reputation initialization transactions was 68 ms, while that for reputation update transactions was 14 ms, both satisfying the real-time requirements at the task level.

8 | CONCLUSIONS AND FUTURE WORK

This paper combines a blockchain-based reputation mechanism with Deep Reinforcement Learning (DRL) to propose a trust-enabled decentralized task offloading scheme. The objective is to ensure secure and efficient task offloading in a dynamic and potentially adversarial CEC environment. Firstly, the paper introduces a blockchain-based reputation mechanism within the CEC architecture to facilitate trustworthy collaboration among edge nodes. To improve the efficiency and accuracy of reputation updates, the BTRU algorithm is proposed, which helps agents better differentiate between honest and malicious nodes. Finally, the DTTO algorithm is introduced, integrating reputation data into the observations and rewards for agents. This minimizes the likelihood of offloading tasks to malicious nodes while optimizing task success rates. To address the limitations of traditional simulation experiments, the paper develops a blockchain-based decentralized edge task offloading prototype system



(a) Impact of task sizes.

(b) Impact of arrival probabilities.

(c) Impact of task deadlines.

FIGURE 10 Success rate of tasks under different system parameters in the testbed.

using Kubernetes and Ethereum, and conducts experiments on the algorithm framework and strategies in a real environment. Extensive experimental results confirm the advantages of the proposed scheme. Compared to other works, the BTRU algorithm provides more efficient and accurate reputation updates, while the DTTO algorithm ensures reliable task offloading and shows significant improvements in task success rates, especially in ultra-dense environments with strict deadlines. Additionally, the DTTO algorithm demonstrates strong adaptability to dynamic changes in various system parameters, maintaining the stability of offloading policies across different environments.

Despite these positive results, several aspects warrant further optimization: (1) Extend the proposed approach to adapt to complex task structures, such as directed acyclic graph (DAG) tasks. (2) Extend the threat model to cover complex malicious behaviors, including collusion attacks and reputation bleaching. (3) Quantify the communication and computational overhead of blockchain operations, and explore low-latency and high-throughput consensus algorithms to further optimize costs and performance.

AUTHOR CONTRIBUTIONS

Genyuan Yang and Wenjuan Li wrote the main manuscript text; Qifei Zhang and Chengjie Pan collected the data and prepared the figures; Minxian Xu and Ben Wang completed the performance test. All authors reviewed the manuscript.

ACKNOWLEDGMENTS

This research was funded in part by the Joint Funds of the Zhejiang Provincial Natural Science Foundation of China under Grant LHZSZ24F020001, and in part by the National Natural Science Foundation of China under Grant 61702151.

FINANCIAL DISCLOSURE

None reported.

CONFLICT OF INTEREST

The authors declare no potential conflict of interest.

CODE AVAILABILITY

The complete testbed source code is open-sourced on GitHub for research usage: https://github.com/ygy45ns/Kubernetes-and-Ethereum-based-Testbed.

REFERENCES

1. Deng S, Zhao H, Fang W, Yin J, Dustdar S, Zomaya AY. Edge intelligence: The confluence of edge computing and artificial intelligence. *IEEE Internet of Things Journal.* 2020;7(8):7457–7469.
2. Buyya R, Srirama SN. *Fog and edge computing: principles and paradigms.* John Wiley & Sons, 2019.
3. Pu Y, Li Z, Yu J, Lu L, Guo B. An elastic framework construction method based on task migration in edge computing. *Software: Practice and Experience.* 2024;54(9):1811–1830.
4. Lin H, Yang L, Guo H, Cao J. Decentralized task offloading in edge computing: an offline-to-online reinforcement learning approach. *IEEE Transactions on Computers.* 2024.
5. Sun C, Wu X, Li X, Fan Q, Wen J, Leung VC. Cooperative computation offloading for multi-access edge computing in 6G mobile networks via soft actor critic. *IEEE Transactions on Network Science and Engineering.* 2021.
6. Sahni Y, Cao J, Yang L, Ji Y. Multi-hop multi-task partial computation offloading in collaborative edge computing. *IEEE Transactions on Parallel and Distributed Systems.* 2020;32(5):1133–1145.
7. He X, Wang S. Peer offloading in mobile-edge computing with worst case response time guarantees. *IEEE Internet of Things Journal.* 2020;8(4):2722–2735.
8. Wang D, Zhu H, Qiu C, Zhou Y, Lu J. Distributed task offloading in cooperative mobile edge computing networks. *IEEE Transactions on Vehicular Technology.* 2024;73(7):10487–10501.
9. Lin R, Xie T, Luo S, et al. Energy-efficient computation offloading in collaborative edge computing. *IEEE Internet of Things Journal.* 2022;9(21):21305–21322.
10. Nujhat N, Haque Shanta F, Sarker S, et al. Task offloading exploiting grey wolf optimization in collaborative edge computing. *Journal of Cloud Computing.* 2024;13(1):23.
11. Sthapit S, Thompson J, Robertson NM, Hopgood JR. Computational load balancing on the edge in absence of cloud and fog. *IEEE Transactions on Mobile Computing.* 2018;18(7):1499–1512.
12. Su M, Wang G, Chen J. Efficient task offloading with swarm intelligence evolution for edge-cloud collaboration in vehicular edge computing. *Software: Practice and Experience.* 2024;54(10):1888–1915.
13. Chen L, Zhou S, Xu J. Computation peer offloading for energy-constrained mobile edge computing in small-cell networks. *IEEE/ACM transactions on networking.* 2018;26(4):1619–1632.
14. Jošilo S, Dán G. Decentralized algorithm for randomized task allocation in fog computing systems. *Ieee/Acm Transactions on Networking.* 2018;27(1):85–97.
15. Liu C, Tang F, Hu Y, Li K, Tang Z, Li K. Distributed task migration optimization in MEC by extending multi-agent deep reinforcement learning approach. *IEEE Transactions on Parallel and Distributed Systems.* 2020;32(7):1603–1614.

16. Hao H, Xu C, Zhang W, Yang S, Muntean GM. Joint task offloading, resource allocation, and trajectory design for multi-uav cooperative edge computing with task priority. *IEEE Transactions on Mobile Computing.* 2024;23(9):8649–8663.
17. Li W, Yang G, Wang B, et al. Adaptive two-stage task offloading based on meta reinforcement learning for mobile edge computing. *The Journal of Supercomputing.* 2025;81(6):1–33.
18. Guha Roy D, Srirama SN. A blockchain-based cyber attack detection scheme for decentralized Internet of Things using software-defined network. *Software: practice and experience.* 2021;51(7):1540–1556.
19. Xu C, Zhang P, Xia X, Kong L, Zeng P, Yu H. Digital twin-assisted intelligent secure task offloading and caching in blockchain-based vehicular edge computing networks. *IEEE Internet of Things Journal.* 2024.
20. Yang G, Li W, Wang B, Pan C, Cao J. An Efficient Privacy Preservation Scheme with Blockchain-Based Reputation Management for IoV-Based Mobile Crowdsensing. In: IEEE. 2024:1–7.
21. Iqbal S, Malik AW, Rahman AU, Noor RM. Blockchain-based reputation management for task offloading in micro-level vehicular fog network. *IEEE Access.* 2020;8:52968–52980.
22. Wang D, Jia Y, Liang L, Dong M, Ota K. A game for task offloading in reputation-based consortium blockchain networks. *IEEE Wireless Communications Letters.* 2022;11(7):1508–1512.
23. Hu Q, Cheng H, Zhang X, Lin C. Trusted resource allocation based on proof-of-reputation consensus mechanism for edge computing. *Peer-to-Peer Networking and Applications.* 2022:1–17.
24. Yamansavascilar B, Baktir AC, Sonmez C, Ozgovde A, Ersoy C. Deepedge: A deep reinforcement learning based task orchestrator for edge computing. *IEEE Transactions on Network Science and Engineering.* 2022;10(1):538–552.
25. Zhang Y, Hu J, Min G. Digital twin-driven intelligent task offloading for collaborative mobile edge computing. *IEEE Journal on Selected Areas in Communications.* 2023;41(10):3034–3045.
26. Mun H, Han K, Yeun HK, et al. Emerging blockchain and reputation management in federated learning: Enhanced security and reliability for internet of vehicles (IoV). *IEEE Transactions on Vehicular Technology.* 2024.
27. Deng Q, Zuo Q, Li Z, Liu H, Xie Y. Blockchain-based reputation privacy preserving for quality-aware worker recruitment scheme in MCS. *IEEE/ACM Transactions on Networking.* 2024.
28. Yin J, Xiao Y, Feng J, Yang M, Pei Q, Yi X. Didtrust: Privacy-preserving trust management for decentralized identity. *IEEE Transactions on Dependable and Secure Computing.* 2025.
29. Kong W, Li X, Hou L, Yuan J, Gao Y, Yu S. A reliable and efficient task offloading strategy based on multifeedback trust mechanism for IoT edge computing. *IEEE Internet of Things Journal.* 2022;9(15):13927–13941.
30. Ouyang Y, Liu W, Yang Q, Mao X, Li F. Trust based task offloading scheme in UAV-enhanced edge computing network. *Peer-to-Peer Networking and Applications.* 2021;14:3268–3290.
31. Nguyen T, Nguyen H, Gia TN. Exploring the integration of edge computing and blockchain IoT: Principles, architectures, security, and applications. *Journal of Network and Computer Applications.* 2024;226:103884.
32. Yu J, Kozhaya D, Decouchant J, Esteves-Verissimo P. Repucoin: Your reputation is your power. *IEEE Transactions on Computers.* 2019;68(8):1225–1237.
33. Tang M, Wong VW. Deep reinforcement learning for task offloading in mobile edge computing systems. *IEEE Transactions on Mobile Computing.* 2020;21(6):1985–1997.
34. Jiang H, Dai X, Xiao Z, Iyengar A. Joint task offloading and resource allocation for energy-constrained mobile edge computing. *IEEE Transactions on Mobile Computing.* 2022;22(7):4000–4015.
35. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347.* 2017.
36. He Q, Cui G, Zhang X, et al. A game-theoretical approach for user allocation in edge computing environment. *IEEE Transactions on Parallel and Distributed Systems.* 2019;31(3):515–529.
37. Shen S, Feng Y, Xu M, et al. A holistic QoS view of crowdsourced edge cloud platform. In: IEEE. 2023:01–10.
38. Marche C, Nitti M. Trust-related attacks and their detection: A trust management model for the social IoT. *IEEE Transactions on Network and Service Management.* 2020;18(3):3297–3308.
39. Qiu X, Zhang W, Chen W, Zheng Z. Distributed and collective deep reinforcement learning for computation offloading: A practical perspective. *IEEE Transactions on Parallel and Distributed Systems.* 2020;32(5):1085–1101.
40. Fang W, Zhang C, Shi Z, Zhao Q, Shan L. BTRES: Beta-based trust and reputation evaluation system for wireless sensor networks. *Journal of Network and Computer Applications.* 2016;59:88–94.
41. Gyawali S, Qian Y, Hu RQ. Machine learning and reputation based misbehavior detection in vehicular communication networks. *IEEE Transactions on Vehicular Technology.* 2020;69(8):8871–8885.

SUPPORTING INFORMATION

Additional supporting information may be found in the online version of the article at the publisher's website.