Trust-aware Caching-constrained Tasks Offloading in Multi-Access Edge Computing

Xinyuan Zhu[a], Fei Hao[a,*], Ming Lei[a], Aziz Nasridinov[b], Jiaxing Shang[c], Zhengxin Yu[d], Longjiang Guo[a,*]

[a]School of Computer Science, Shaanxi Normal University, Xi'an 71009, China
[b]Department of Computer Science, Chungbuk National University, Cheongju 28644, South Korea
[c]College of Computer Science, Chongqing University, Chongqing 400030, China.
[d]School of Computing and Communications, Lancaster University, United Kingdom

Abstract

Multi-access edge computing (MEC) networks face significant challenges in managing congestion and safeguarding personal privacy data on a massive scale. Integrating trust awareness into MEC networks presents an opportunity to enhance security and privacy by correlating human relationships with connected devices. Moreover, leveraging trust-aware task caching and offloading holds promise in mitigating latency and reducing energy consumption. Despite existing research efforts to address these challenges, they often overlook either trust awareness or caching optimization in task offloading, potentially compromising security or leading to task failures. To address this gap, this paper proposes a novel approach: a trust-aware task offloading strategy with cache constraints (TCTO) in MEC networks, which considers social relationships, task offloading, and caching. Drawing on the characteristics of bipartite graphs and bipartite perfect matching, we develop a trust-aware caching-constrained task offloading algorithm based on bipartite graphs. This algorithm aims to select task offloading strategies that minimize delay, energy consumption in task transmission and execution, while maximizing security among devices in MEC networks. Extensive simulations demonstrate that our proposed method has a better performance than other task offloading strategies for reducing delay and energy consumption in the process of task transmission and execution. Compared with the other baselines, the overhead of our proposed method is reduced 55.65% ~ 96.20% compared with other baselines.

Keywords: MEC, Task offloading, Trust awareness, Caching, Bipartite matching graph

1. Introduction

Multi-access Edge Computing (MEC), positioned as a pivotal technology and facilitator of the Internet of Things (IoT), seamlessly merges telecommunications and IT services to furnish cloud computing functionalities at the network periphery, thereby meeting stringent network performance demands concerning latency and bandwidth [1, 2]. In the MEC network, the majority of user operations are executed at the network edge, leading to significant reductions in communication latency and energy consumption within the MEC networks [1, 3, 4].

In MEC networks, task offloading serves as an effective means to mitigate resource constraints and enhance service quality [5]. Device-to-Device (D2D) communication facilitates resource sharing and equilibrium among diverse devices by enabling direct data transmission to nearby counterparts [5, 6, 7]. Moreover, frequent task requests within short time intervals can strain network links, leading to resource wastage or poor user experiences [8, 9]. Therefore, those frequently requested tasks can be cached on the devices in advance. If these tasks to be offloaded are already cached on the MEC device, the delay and energy consumption can be markedly reduced [8, 10]. However, assuming universal willingness among user devices to provide services and resources in D2D-assisted MEC networks is unrealistic. Social relationships influence user's trust, fostering efficient and reliable cooperation in various aspects, including data transmission, task offloading, and resource sharing [11].

In certain scenarios of Device-to-Device (D2D) auxiliary networks, trust awareness is considered a key factor in improving the performance of MEC systems [12, 13, 14]. Social relationships, such as proximity and user intimacy, significantly influence how devices collaborate for task offloading and resource sharing. Among the various dimensions of social relationships, trust awareness plays a particularly critical role. Trust can be regarded as a fundamental component of social awareness, providing a more concrete and actionable indicator for evaluating the reliability of collaborative partners [15]. While general social ties describe the existence and strength of social relationships, trust awareness focuses specifically on the perceived reliability, honesty, and willingness of a device (or its owner) to participate in cooperative activities. For example, in the context of the Social Internet of Things (SIoT) [16], devices belonging to the same social group such as smartphones used by family members,

*Corresponding author
Email address: feehao@gmail.com (Fei Hao)

close friends, or colleagues tend to exhibit higher levels of mutual trust. In dynamic and potentially adversarial environments, trust becomes a decisive factor for task offloading and secure resource sharing [17, 14]. By incorporating trust awareness into offloading strategies, MEC systems can prioritize collaboration with trusted devices, thereby improving both the security and efficiency of the offloading process. Therefor, trust awareness not only extends but also strengthens the capabilities of social awareness by focusing on reliability and safety in device cooperation.

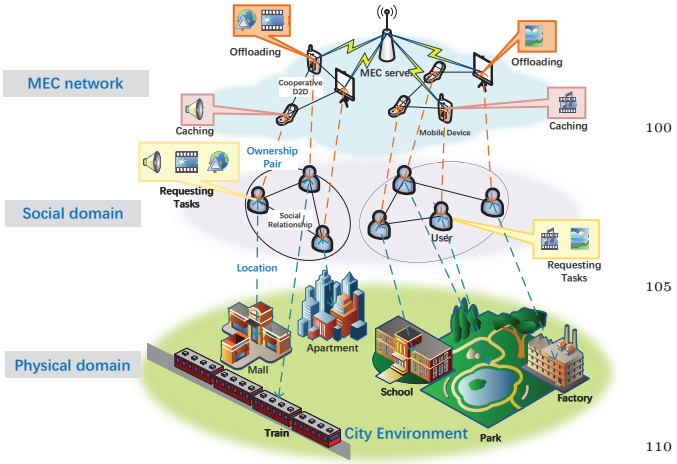


Figure 1: An illustrate of motivating scenario

In Figure 1, we illustrate a trust-aware caching constrained D2D-assisted MEC network in an urban environment as our motivating scenario. Within this environment, individuals in the city possess varying degrees of trust, which can be mapped onto the devices within the MEC network based on ownership. This integration facilitates safer and more efficient communication links between devices. Consequently, tasks requiring high levels of privacy can be offloaded to devices with high levels of trust, thereby incorporating trust awareness into the decision-making process. Simultaneously, devices in the MEC network possess caching capabilities capable of storing various types of tasks, such as videos, audio, and images requests. This caching functionality significantly reduces both delay and energy consumption associated with executing tasks repeatedly by pre-storing task results, which is particularly advantageous for delay-sensitive and energy-efficient tasks. In this trust-aware caching-constrained D2D MEC network, we can adopt task offloading strategies that prioritize both trustworthiness and efficiency, resulting in an enhanced user experience and ensuring the security of task transmission between devices.

However, there are also many challenges in the research of MEC network. Firstly, ensuring secure and efficient D2D connections is complex yet crucial for mitigating latency and energy consumption [18]. Additionally, an ill-conceived task offloading scheme can lead to prolonged response times and resource wastage [19]. Thus, there's a pressing need to develop optimal task offloading strategies that integrate edge caching and consider the trust degrees among devices in MEC networks. While some researches [8, 20, 21] have tackled task offloading alongside edge caching, these approaches often fall short in accurately mapping real-world trust relationships to device nodes in MEC networks. Furthermore, certain studies such as [22], have incorporated trust relationships into task offloading but failed to account for how device caching impacts offloading decisions. While certain researches [23, 24] have delved extensively into cooperative caching with trust awareness, they have yet to thoroughly explore the implications of trust-aware cooperative caching on the selection of task offloading strategies.

To tackle the challenges inherent in MEC, this paper introduces a approach: the trust-aware caching-constrained task offloading (TCTO) strategy in MEC networks. With the TCTO approach, devices in the MEC network are equipped to cache computation results and resources, thus catering to the low delay requirements of tasks sensitive to delays [25]. Moreover, we incorporate the influence of trust relationships among users on device trust in MEC networks. Building upon these considerations, we propose a comprehensive task offloading scheme encompassing local offloading, direct-to-cloud offloading, D2D offloading, and D2D-assisted to-cloud offloading strategies [7, 26]. Our primary objective is to identify the optimal task offloading strategy characterized by high trust degrees, minimal delay and energy consumption in both transmission and execution phases. In summary, the contributions of this paper can be outlined as follows:

- The trust-aware caching-constrained task offloading strategy is proposed to reduce the delay and energy consumption during task execution and improving the security of the transmission process in MEC network. We subsequently define a comprehensive overhead metric that integrates considerations of delay, energy consumption, and trust awareness for the evaluation of task offloading strategies. We further implement the optimal problem by minimizing the overhead based on the trust-aware caching constrained task offloading strategy.

- In order to solve the above optimization problem, inspired by the characteristics of the bipartite graph, we adapt the optimization problem to the bipartite graph. Initially, we construct a weighted bipartite graph to represent the relationship between tasks and devices. Subsequently, we employ the Kuhn-Munkres (K-M) algorithm to facilitate optimal matching between tasks and devices, thereby enabling the selection of an appropriate task offloading strategy.

- In order to verify that our proposed method can effectively reduce delay and energy consumption, we conducted extensive simulation experiments. Sim-

ulation results show that the overhead of the proposed method is significantly lower than other baselines. According to the experimental results, we find that the overhead is 55.65% ~ 96.20% compared with other baselines, representing the lower delay and energy consumption in task transmission and execution. Furthermore, our simulation results illustrates that the caching also takes an significant effect on reducing consumption of delay and energy.

The organization of this paper is as follows. Section 2 presents the related work. Section 3 explains the system model of the proposed TCTO. The optimization problem of this paper is formulated in section 4. The trust-aware caching constrained task offloading based on bipartite graph matching method which solves the above optimization problem is then elaborated in Section 5. The performances and evaluations of the proposed TCTO simulation are provided in 6. Finally, section 7 concludes this work with future directions.

2. Related Work

There are many works that focus on the optimization problem in MEC networks with the consideration of socially-aware task offloading or collaborative caching. Details are as follows.

2.1. Socially-aware Task Offloading in MEC

Long et al. [14] carried out research on the energy-efficient task offloading in socially-aware D2D-assisted MEC networks, where user devices can offload tasks to nearby devices or further forward them to MEC servers based on social relationships. Chen et al. [22] proposed a socially-aware system model for collaborative MEC, where the social links between devices is represent on device social graph. In [27], Xu et al. proposed the Mobile Device Selection Algorithm (MDSA), which provides device-to-device offloading by considering social relationships, location dependencies, and activities of mobile devices in the selection of target mobile devices. Liu et al. [28] incorporated the social relationship of energy harvesting (EH) mobile devices (MDs) into the computation offloading schemes in fog computing and developed a dynamic computing offloading scheme using the game theory. Li et al. [29] considered the effects of social attributes on task offloading and resource allocation in real offloading system, and the response computing resources are allocated according to social attributes. Ibrar et al. [30] implemented an adaptive capacity task offloading solution considering equipment utilization and social relationship strength in order to improve resource utilization, improve QoS and obtain better task completion rate based on D2D social Industrial Internet of Things (ToSIIoT). In [15], Alioua et al proposed a trust management mechanism based on blockchain, which improves the security and reliability of task offloading in MEC through reputation model and Stackelberg game incentive mechanism, and built trust relationship between system entities. Wang et al. [16] presented a new Social Internet of Things (SIoT collaborative group and device selection problem (SCGDSP) and an approximation algorithm to optimize load sharing between SIoT devices, MEC servers, and remote servers, reducing communication and computation costs while supporting large-scale distributed deployment. Gao et al. [31] investigated dynamic computation offloading mode selection in MEC-aided low-latency IoT, leveraging social ties in human networks to minimize execution latency and energy consumption, and proposed a Lyapunov-based solution (DPP algorithm) for efficient resource assignment.

However, most of these above researches [14, 22, 32] focused on the impact of social awareness or trust awareness on the task offloading and resource allocation in MEC networks, but they ignored that the cache of user devices and MEC servers plays an important role in reducing the delay and energy consumption in the process of task offloading by caching the results of tasks to the edge devices in advance. Therefore, in this paper, we will combine the cache and social awareness to optimize the task offloading scheme in MEC network.

2.2. Cooperative Caching in MEC

Fan et al. [23] investigated the cooperative caching problem in Fog Radio Access Networks (F-RANs), and presented a clustering method based on Hedonic Coalition Game (HCG) by exploiting the social relationships among Fog Access Points (F-APs) to optimize transmission delay and energy consumption. Zohreh et al. [33] proposed an encoded/non-encoded content placement, where content can be propagated among nearby cache nodes according to its popularity. Meybodi et al. [34] studied the caching of multimedia content under MEC, and proposed the transformer-based edge caching (TEDGE) framework which can efficiently and dynamically predict the popularity of content, the popularity of content in active caching schemes, and store the content in advance. In [35], a similarity-aware popularity-based caching (SAPoC) algorithm was proposed to exploit the similarity between contents to improve the performance of wireless edge caching in dynamic scenarios. In [36], Bai et al. implemented a heterogeneous network collaborative caching and video transcoding architecture based on MEC, and proposed a new knowledge graph-based video caching scheme to optimize the cache hit rate and latency performance. In [37], Bai et al. proposed a social-aware D2D caching scheme that integrates the concept of social incentive and recommendation with D2D caching decision making.

However, the works mentioned above paid attention on the caching scheme in MEC networks. Researches [33, 34, 35] were aiming at solving the problem of content placement, and [23, 36, 37] settled with cooperative caching problem in MEC networks. But in this paper, we comprehensively concentrate that caching of MEC devices will

have a significant effect on the task offloading scheme for reducing delay and energy consumption. At the same time, we associate the trust awareness with caching to optimize the problem of task offloading in D2D MEC networks, which differs from most of the existing works.

Finally, in view of the above researches, we comprehensively consider important factors of cache and trust awareness on task offloading scheme, and propose a trust-aware caching-constrained task offloading strategy in MEC networks, thus avoiding inaccurate execution results or failure of task offloading due to a lack of consideration of one of these factors. The comparison of the aforementioned research works and our work is presented in Table 1.

Table 1: comparison of existing research works

Research	Social and Trust	Caching	Task offloading
[8, 38, 39]	×	✓	✓
[14, 22, 32]	✓	×	✓
[23, 35, 37]	✓	✓	×
Our Scheme	✓	✓	✓

3. System Model

In this section, we introduce the system model including the communication model, cache and queue model, computational model and social relationship model. The specific descriptions for these models are elaborated as follows.

3.1. System Description

A trust-aware D2D-assisted MEC system is considered which is as shown in Figure 2. It consists of several base stations (BSs) equipped with MEC servers and user equipments in physical domain. All user devices are divided into two types, one is DT (distant transmitter), and the other is NR (nearby device) which assists DT for task offloading. DTs and NRs are carried by humans in the social domain.

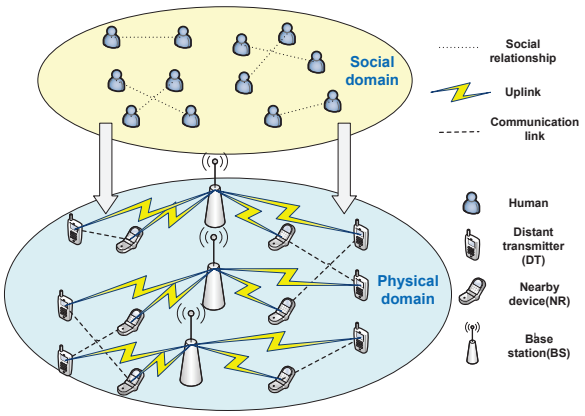


Figure 2: An illustration of system model

For the MEC system depicted in Figure 2, we assume that tasks exist in the task set $C=\{c_1, c_2, \cdots, c_i, \cdots, c_I\}$, and the task size is represented as $D=\{d_1, d_2, \cdots, d_i, \cdots, d_I\}$, where I is the total number of tasks; it is considered that the BS storage size is Ω, the computing power is λ_B, and the BS set is $B=\{b_1, b_2, \cdots, b_K\}$; DT and NR have the same storage size Λ; the computing power is λ_U, the DT set is $S=\{s_1, s_2, \cdots, s_M\}$, and the NR set is $R=\{r_1, r_2, \cdots, r_N\}$. In addition, four offloading strategies are considered: local offloading, directly to cloud offloading, D2D offloading, and D2D-assisted to cloud offloading, shown in Figure 3. In this paper, we consider the cloud to refer to the BS equipped with MEC servers. We assume that computational tasks are generated by DT, and the arrival of tasks in the system follows a Poisson process based on the research [40]. And we consider the binary offloading of tasks, tasks are constrained to complete offloading and execution within a limited amount of time τ.

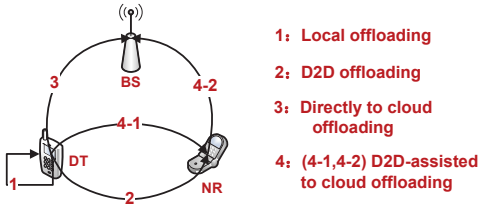


Figure 3: An illustration of task offloading strategies in MEC network

In next subsections, we elaborate the communication model and the computational model in physical domain, cache and queue model, and the social relationship model in social domain, respectively.

3.2. Communication Model

Considering that tasks are transmitted through wireless channels when offloading to nearby auxiliary devices and offloading to BSs, we assume that the wireless channels between users and BSs are stable. Therefore, according to the Shannon formula [41, 42], the content transmission rate of the two communication links under this model can be expressed as follows. The transmission rate of the communication link between user m and user n is expressed as Formula 1.

$$R^U_{m,n} = W_1 log_2(1 + \frac{P_{U,S} l^{-\alpha}_{m,n,U}}{\sigma^2 + V}) \quad (1)$$

where W_1 is the bandwidth of the communication link between users, $P_{U,S}$ represents the transmission power of the users, $l_{m,n,U}$ denotes the distance of user m and n, α is the path loss exponent, the noise is σ^2, and V is the sum of signal interference generated by neighboring devices [42].

The transmission rate of the communication link between user m and BS k is expressed as Formula 2.

$$R^B_{m,k} = W_2 log_2(1 + \frac{P_{U,S} l^{-\alpha}_{m,k,B}}{\sigma^2 + V}) \quad (2)$$

where W_2 is the bandwidth of the communication link between user and BS, $l_{m,k,B}$ denotes the distance of user m and BS k.

3.3. Cache and Queue Model

In this model, the device' memory is divided into two parts. A part of the memory space is used to cache the historical execution results of the tasks (the memory sizes of the user and BS are H_U and H_B respectively), and the remaining part is used to store the task execution queue Q.

In this cache and queue model, user devices and BSs are able to store task execution results. Leveraging this feature, we hypothesize that when a new task requires execution, we prioritize the possibility of retrieving the task's execution results from device caches. The process involves the following steps. Initially, our proposed method conducts a local search to ascertain if the execution result is available locally. If the result is found on the device itself, it is directly returned. In cases where the result is not found locally, we extend the search to nearby auxiliary devices. If the result is located on any of these devices, it is retrieved. If the execution result is still unavailable, we proceed to search BSs. If the result exists on any of the BSs, it is retrieved without the need for task re-execution.

We set $X_m^U=\{x_{m,1}^U, x_{m,2}^U, \cdots, x_{m,I}^U\}$ for each user device and $X_k^B=\{x_{k,1}^B, x_{k,2}^B, \cdots, x_{k,I}^B\}$ for BS to record the execution results of tasks. For any task c_j, the $x_{m,j}^U$ in the vector X_m^U stored on user m is interpreted as follows: if the result of c_j has stored on the user m, then, $x_{m,j}^U$=1; otherwise, $x_{m,j}^U$=0. Similarly, if the result of c_j has stored on the BS k, $x_{k,j}^B$=1; otherwise, $x_{k,j}^B$=0.

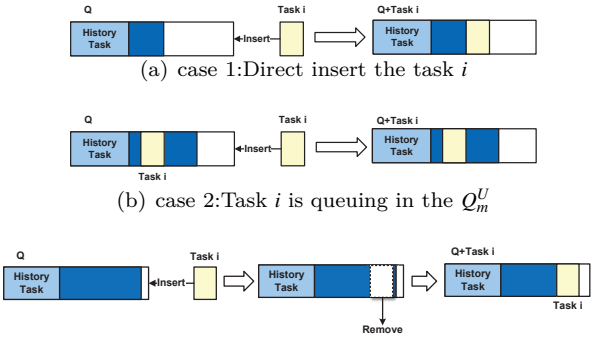
Due to the limited computing resources and cache space of edge devices, it is necessary to consider whether the remaining computing resources and cache space of the devices are more than the newly generated task size d_j We set the waiting queue $Q_m^U=\{q_{m,1}^U, q_{m,2}^U, \cdots, q_{m,r}^U, \cdots\}$ on each user and $Q_k^B=\{q_{k,1}^B, q_{k,2}^B, \cdots, q_{k,r}^B, \cdots\}$ on each BS respectively, which are used to store the waiting task queue on the current device. The queue is initially empty, and the task scheduler selects the one with the shortest completion time. Among them, $q_{m,r}^U$ and $q_{k,r}^B$ represent the size of the r_{th} task in the waiting queue of the current device(user m or BS k). When there is a new task that needs to be offloaded, we insert the task at the end of the waiting queue.

Example 1. Figure 4 depicts three cases of the waiting queue for the user m when the result of task i is not cached on user m.

(1) If the remaining cache space and computing resources are sufficient, and the task does not exist in the waiting task queue Q_m^U, then the task will be directly inserted at the end of the queue.

(2) If the task i already exists in the task waiting queue , then it is unnecessary to insert the task at the end of the queue, but wait for execution at the position of the current task already stored in the queue.

(3) If the remaining local computing resources and cache space are insufficient when inserting a task into the end of the queue, that is, $\Lambda - H_U - \sum Q_m^U < d_i$, then the cached tasks will be removed from memory based on memory replacement policies, such as FIFO(First Input First Output), LRU(Least Recently Used), or LFU(Least Frenquent Used), until there is enough space to store the new tasks. In this paper, the least recently used task will be deleted iteratively according to the LRU algorithm (The specific reasons can be found in 6.3.2) until the remaining memory space is enough to accommodate the task; which is formulated as $\Lambda - H_U - \sum Q_m^U \geq d_i$, then the task will be inserted into the task waiting queue.



(a) case 1:Direct insert the task i

(b) case 2:Task i is queuing in the Q_m^U

(c) case 3:The caching space of Q_m^U is insufficient for task i

Figure 4: An illustration of waiting queue of Q_m^U when inserting task i

3.4. Computational Model

In this model, the delay and energy consumed under the four task offloading strategies are calculated respectively.

3.4.1. Local Execution

If there is a execution result locally, the result will be returned directly, and the transmission delay and energy consumption are set to 0. If not hit, the local execution time is represented as follows:

$$t_e^L = \frac{d_j}{\lambda_u} \tag{3}$$

where λ_u is the computational ability of user device. The local waiting queuing delay of user m is expressed as follows:

$$t_m^L = \frac{\sum_{p=1}^{p=r} q_{m,p}^U}{\lambda_u} \tag{4}$$

in which r is the position of the task in the task waiting queue, the $\sum_{p=1}^{p=r} q_{m,p}^U$ is the sum of task r and the task before

it. Therefore, the delay under the local offloading strategy of user m is as follows:

$$T_m^L = (1 - x_{m,j}^U)(t_e^L + t_m^L) \tag{5}$$

The energy consumption of user m can be calculated by execution power and time of user's device:

$$E_m^L = P_U T_m^L \tag{6}$$

where P_U is the execution power of user's device.

3.4.2. Directly to Cloud Offloading Strategy

If hits on BS, only the delay and energy consumption in the transmission process are taken into account. In this process, the transmission delay from user m to BS k is described as follows:

$$t_{m,k}^B = \frac{d_j}{R_{m,k}^B} \tag{7}$$

where d_j is size of task j, and $R_{m,k}^B$ is the rate of transmission between user device m and base station k. The energy consumption for the task transmission from user m to BS k is calculated as follows:

$$e_{m,k}^B = P_{U,S} t_{m,k}^B \tag{8}$$

If it is not hit, and the memory of the BS satisfies $\Omega - H_U - \sum Q_k^B \geq d_j$, the delay and energy consumption of the offloading strategy are divided into three parts: consumed in transmission process, queuing process and execution process. The execution time of BS k can be represented as follows:

$$t_e^B = \frac{d_j}{\lambda_B} \tag{9}$$

λ_B is the computational ability of base station. The time delay consumed by the queuing process is:

$$t_k^B = \frac{\sum_{p=1}^{p=r} q_{k,p}^B}{\lambda_B} \tag{10}$$

Therefore, based on above, under the directly to cloud offloading strategy, the delay can be calculated as:

$$T_{m,k}^B = (1 - x_{k,j}^B)(t_e^B + t_k^B) + t_{m,k}^B \tag{11}$$

The energy consumption is as follows:

$$E_{m,k}^B = P_B(1 - x_{k,j}^B)(t_e^B + t_k^B) + e_{m,k}^B \tag{12}$$

where P_B is the execution power of BS k.

3.4.3. D2D Offloading Strategy

If it hits on a nearby auxiliary node, consider the transmission delay between devices is:

$$t_{m,n}^D = \frac{d_j}{R_{m,n}^U} \tag{13}$$

The energy consumption under this process is expressed as follows:

$$e_{m,n}^D = P_{U,S} t_{m,n}^D \tag{14}$$

If there is no hit, and the nearby auxiliary device memory satisfies $\Lambda - H_U - \sum Q_m^U \geq d_j$, then the delay and energy consumption of the offloading strategy are divided three parts: transmission process, queuing process and execution process. Similarly, the execution time of user m is:

$$t_e^D = \frac{d_j}{\lambda_U} \tag{15}$$

The time delay consumed by the queuing process on the user n is described as follows:

$$t_n^D = \frac{\sum_{p=1}^{p=r} q_{n,p}^U}{\lambda_U} \tag{16}$$

Therefore, under the condition that the memory is satisfied, the delay from user m to user n of the D2D-assisted offloading execution strategy is calculated as:

$$T_{m,n}^D = (1 - x_{m,j}^U)(t_e^D + t_n^D) + t_{m,n}^D \tag{17}$$

The energy from user m to user n of D2D-assisted offloading execution strategy is as follows:

$$E_{m,n}^D = P_U(1 - x_{m,j}^U)(t_e^D + t_n^D) + e_{m,n}^D \tag{18}$$

3.4.4. D2D-Assisted to Cloud Offloading Strategy

Under the D2D-assisted to cloud offloading strategy, if it hits at BS k, the transmission delay from user m to user n and to BS k can be calculated as follows:

$$t_{m,n,k}^{DA} = \frac{d_j}{R_{m,n}^U} + \frac{d_j}{R_{n,k}^B} \tag{19}$$

The energy consumption of the transmission process is:

$$e_{m,n,k}^{DA} = P_{U,S} t_{m,n,k}^{DA} \tag{20}$$

If it not hit, we can obtain the execution time under this execution strategy is the execution time of the task on BS k.

$$t_e^{DA} = \frac{d_j}{\lambda_B} \tag{21}$$

The queuing delay on the BS k is:

$$t_k^{DA} = \frac{\sum_{p=1}^{p=r} q_{k,p}^U}{\lambda_B} \tag{22}$$

Therefore, for task j, the delay from user m to user n and from user n to BS k under this strategy is represented as:

$$T_{m,n,k}^{DA} = (1 - x_{k,j}^B)(t_e^{DA} + t_k^{DA}) + t_{m,n,k}^{DA} \tag{23}$$

The energy consumption is as follows:

$$E_{m,n,k}^{DA} = P_B(1 - x_{k,j}^B)(t_e^{DA} + t_k^{DA}) + e_{m,n,k}^{DA} \tag{24}$$

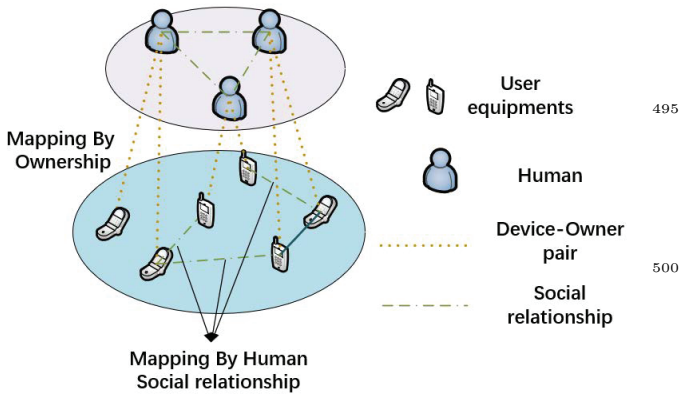


Figure 5: An illustration of social relationship

3.5. Social Relationship and Trust Model

In MEC networks, especially in device cooperative offloading scenarios such as D2D communication, successful collaboration between devices depends not only on physical network conditions (e.g., channel quality), but also closely on their social relationships. We construct a social graph to model and quantify these relationships, using trust levels to measure whether a device is reliable for processing or forwarding tasks. When selecting the offloading strategy or target device, the scheduler does not simply choose the one with the lowest delay and energy from all available devices. Instead, it makes decisions based on a comprehensive system cost metric that integrates weighted trust degree, delay and energy consumption, ultimately selecting the optimal offloading strategy. Moreover, incorporating trust awareness can enhance the stability and sustainability of collaboration by avoiding frequent task failures. In dense networks, ignoring trust between devices may lead to offloading tasks to untrustworthy devices, potentially causing security or privacy risks.

Considering the privacy and security of the user's social interaction, the trust between devices is also important. We believe that devices are more inclined to offload their tasks to those with closer social relationships and high trust for offloading and execution. We establish a correspondence between the social relationships among devices and those among users in the real world [43, 44]. As shown in the network model in Figure 5, we use users' ownership of devices to represent both social connections in the human domain and the physical interactions between the devices. Specifically, the trust between devices are modeled based on the social strengths of the users who own them.

Additionally, the social relationships between users are characterized by similarities in their behaviors. This behavioral similarity is quantified by analyzing the likelihood of users selecting similar content and engaging in comparable activities. The probability of users consuming similar content is used to describe the similarity of user behavior, which determines the strength of their social relationship, which is mapped to trust degree between devices.

The trust degree, denoted as ω, quantifies the level of trust between two devices based on the similarity of tasks they execute. This similarity is captured through the intersection of tasks stored in the task queues of the devices, which can be represented as vectors. For instance, let X_m^U and X_k^B denote the task vectors of user m and BS k, respectively, while $d=\{d_1, d_2, \cdots, d_i\}$ represents the task sizes. The trust degree is then defined by the similarity in the task vectors, normalized by the task size, as follows:

$$\omega_{m,n} = \frac{X_m^U X_n^U d^T}{X_m^U d^T} \text{ or } \omega_{m,k} = \frac{X_m^U X_k^B d^T}{X_m^U d^T} \tag{25}$$

Here, X_m^U and X_n^U represents the dot product of the task vectors for users m and n, and the normalization by $X_m^U d^T$ ensures that the trust degree reflects the relative importance of the common tasks being offloaded.

This formulation provides a way to calculate the trust degree between devices based on both the tasks they share and the behavioral similarities of their respective users. By incorporating user behavior into the trust model, we ensure that the system reflects real-world social dynamics and privacy concerns, where devices with stronger trust (as determined by user behavior) are more likely to collaborate and share resources in a secure and privacy-preserving manner.

4. Problem Formulation

The purpose of this paper is to find a strategy with high trust between device nodes, low delay and energy consumption during the transmission and execution of tasks in MEC network. Thus, based on the references [32, 14], we define the overhead metrics Z_m^L, $Z_{m,k}^C$, $Z_{m,n}^{D2D}$, $Z_{m,n,k}^{DA}$ corresponding to the four strategies of local, directly to cloud, D2D, and D2D-assisted to cloud, respectively and quantify the four metrics as follows. Under the local task offloading strategy, the overhead Z_m^L can be calculated as:

$$Z_m^L = e^{-\omega_{m,m}}[\theta\ T_m^L + (1-\theta)E_m^L] \tag{26}$$

where θ is the weight of delay and energy consumption and $\theta \in [0,1]$. T_m^L and E_m^L represent the delay and energy consumption for the local offloading strategy, respectively, while $\omega_{m,m}$ represents the trust level between the device m and itself.

The overhead $Z_{m,k}^C$ of directly to cloud task offloading strategy can be described as:

$$Z_{m,k}^C = e^{-\omega_{m,k}}[\theta\ T_{m,k}^B + (1-\theta)E_{m,k}^B] \tag{27}$$

$T_{m,k}^B$ and $E_{m,k}^B$ are the delay and energy consumption for offloading from device m to base station k and then to the cloud. The trust between device m and base station k is denoted by $\omega_{m,k}$.

Under the D2D offloading strategy, the overhead $Z_{m,n}^{D2D}$ is expressed as:

$$Z_{m,n}^{D2D} = e^{-\omega_{m,n}}[\theta\ T_{m,n}^{D} + (1-\theta)E_{m,n}^{D}] \tag{28}$$

$T_{m,n}^{D}$ and $E_{m,n}^{D}$ represent the delay and energy consumption for direct device-to-device offloading, and $\omega_{m,n}$ is the trust level between devices m and n.

And the overhead $Z_{m,n,k}^{DA}$ of which tasks offload by D2D-assisted to cloud can be calculated as follows:

$$Z_{m,n,k}^{DA} = e^{-\omega_{m,n}\omega_{n,k}}[\theta\ T_{m,n,k}^{DA} + (1-\theta)E_{m,n,k}^{DA}] \tag{29}$$

In this case, $T_{m,n,k}^{DA}$ and $E_{m,n,k}^{DA}$ represent the delay and energy consumption for D2D-assisted offloading to the cloud, while $\omega_{m,n}$ and $\omega_{n,k}$ denote the trust levels between devices m and n, and between device n and base station k, respectively.

In this paper, we choose a strategy with a large trust degree and low delay as well as energy consumption as the optimal offloading strategy. Then, the problem can be formulated as follows.

$$\begin{aligned} & min\{Z_m^L, Z_{m,k}^C, Z_{m,n}^{D2D}, Z_{m,n,k}^{DA}\} \\ s.t.\ & C1: \theta \in [0,1]; \\ & C2: \omega_{m,m}, \omega_{m,n}, \omega_{m,k}, \omega_{n,k} \in [0,1]; \\ & C3: x_{m,j}^{U} \in \{0,1\}, x_{k,j}^{B} \in \{0,1\}; \\ & C4: \sum Q_m + H_U \le \Lambda, \sum Q_k + H_B \le \Omega \end{aligned} \tag{30}$$

where $C1$ denotes the weight of delay and energy consumption constraint, and θ denotes the weight of delay, $(1-\theta)$ is the weight of energy consumption. Especially, if we balance the delay and energy, θ is set to 0.5. $C2$ is the trust constraint which represents the trust between device nodes, and the more two nodes trust each other, the closer the value of ω will be to 1. $C3$ represents wether the history result of task j has been cached on the user device m or BS k or not, and the $x_{m,j}^{U}$ and $x_{k,j}^{B}$ are dispersed which can only be 0 or 1. And $C4$ is the constraint of caching size of user devices and BSs, which guarantees that the size of all the tasks caching on the user device or the BS is less than the memory of user device or BS, respectively.

Theorem 1. The optimization problem of Eq.(30) is NP-hard.

Proof. In optimization problem, it is obvious that both discrete and continuous variables are involved, so the above problem is a mixed integer nonlinear programming problem (MINLP) [45]. And it has been proved that the MINLP problem is NP-hard [46, 47]. Therefore, the optimization problem Eq.(30) is NP-hard as well. □

5. Trust-aware Caching Constrained Task Offloading Strategy Selection Based on Bipartite Graph Matching

To overcome the NP-hardness of the above problem, this section is devoted to addressing the optimization problem by exploiting the underlying social relationships among device users. We will consider the bipartite graph matching based on social awareness to minimize the overhead consumption of delay and energy.

Bipartite graph is a graph structure where the vertex set can be divided into two disjoint subsets, and edges are allowed only between vertices from different subsets. In the context of this work, we model the offloading environment as a bipartite graph, where one subset represents the tasks and the other represents devices capable of executing tasks, including local devices, nearby devices and BSs. This modeling choice is motivated by the natural suitability of bipartite graphs for representing assignment-type problems, such as pairing tasks with available computing devices. Furthermore, the bipartite matching framework enables the application of efficient optimization algorithms (e.g., the Hungarian algorithm), which significantly reduce the computational complexity compared to exhaustive search methods.

5.1. Task Cache and Placement

Before selecting the task strategy, we first need to solve the problem of content placement on the users. The main problem to be solved is to choose which tasks should be cached on which user devices. The main steps to solve this problem are as follows: Firstly, according to the unit popularity of the content, that is, the ratio of the frequency of a task being requested to the size of the task, the content is sorted in descending order. Then, according to the size of the memory, the tasks with a higher frequency of requests are sequentially selected for caching to improve the hit rate of the tasks on the nodes as much as possible.

Obviously, the main idea of solving this content placement problem is the same as solving the $0-1$ Knapsack problem. To solve the problem, we use the greedy strategy of maximum unit popularity. Here, the unit popularity of a task is the ratio of the frequency at which a task is requested to the size of the task. First, the unit popularity of each task is calculated, and tasks are sorted by unit popularity from largest to smallest. Then, according to the greedy strategy of unit popularity priority, the tasks are cached in the node memory in sequence until all tasks are cached or the remaining memory capacity is empty. If the size of the current task does not exceed the remaining memory capacity, the task is cached to the node, otherwise, the task is not cached to the node. This strategy is executed in a loop until the remaining memory is 0.

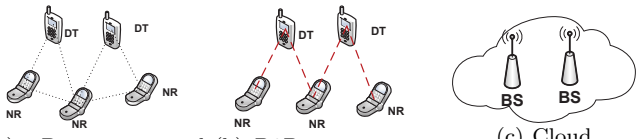
5.2. Bipartite Graph Construction

To solve the task offloading problem, we need to construct a suitable bipartite graph based on trust awareness according to the trust between devices. In a bipartite graph, all the vertices can be divided into two sets, there are no edges in the two sets and the edges in the graph only exist between the two sets. Based on the characteristics of the bipartite graph, it is effective and convenient to use the bipartite graph to solve our optimization problem. And we try to solve the problem of task offloading

based on the maximum weight bipartite perfect matching method.

5.2.1. Device Social Graph

A device social graph in trust-aware MEC network is introduced. It can be considered to achieve cooperation in MEC by using human social relationships. The trust relationships between BSs are represented in the device social graph. The nodes in the device social graph are the same as the devices in the MEC networks, and the edges in the device social graph represent the trust relationships between devices. We believe that when two devices trust each other, there will be a certain social relationship, then there will be an edge connecting two devices. The device social relationship graph, D2D connection graph and cloud [32] are respectively represented as shown in Figure 6.



(a) Device social graph (b) D2D connectivity graph (c) Cloud

Figure 6: Preparation for bipartite matching graph

According to the above preparations, we can easily obtain a network model based on the trust and the D2D links. The D2D-assisted MEC network is shown in Figure 7.

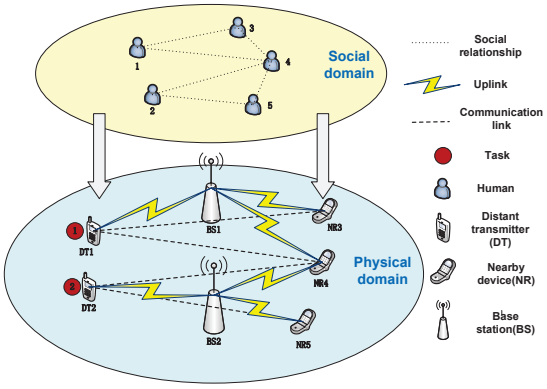


Figure 7: An illustration of D2D-assisted MEC network

5.2.2. Bipartite Matching Graph

As shown in Figure 6, by considering the device social graph and the D2D connection graph, a weighted bipartite graph $G(V, E)$ can be conducted, where V is the vertex set and E is the edge set of the bipartite matching graph. And the vertex set consists of two parts, one is the set of tasks where vertex m represents the task of device m; the other contains vertices of executors including the user devices and BSs for task execution. Then, the cooperative task offloading strategy under trust awareness can be modeled as a minimum weight bipartite perfect matching problem [22] and the bipartite matching graph is shown in Figure 8.

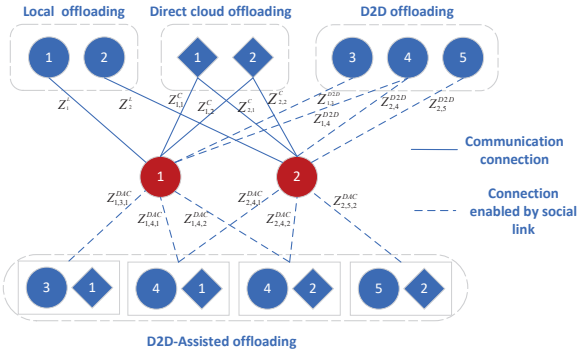


Figure 8: Bipartite Matching Graph of Trust-aware Caching Constrained Task Offloading

Then we elaborate on the edge set of bipartite graph. The edges of the bipartite graph are divided into the following four different cases: the first is that the task vertex m is connected with its corresponding device vertex, which is corresponding to local task offloading strategy, then the weight value of this edge is under the local offloading strategy Z_m^L. The second case is under the D2D offloading strategy, there is an edge between the task vertex m and the device vertex n, which is connected with the task vertex m on the device social graph and the D2D connectivity graph, and the weight value of this edge is under the D2D offloading strategy $Z_{m,n}^{D2D}$. The weight of edge between the task vertex m and its corresponding BS vertex k in the cloud, which represents the directly to cloud offloading, is set as $Z_{m,k}^C$. As for D2D-assisted to cloud offloading, if there exists a social connection between the task vertex m and the device vertex n in device social graph and a D2D link in D2D connection graph, then there is an edge between the task vertex and the binding vertex consisting of the auxiliary device vertex n and the corresponding BS vertex k. Correspondingly, the weight of this edge is $Z_{m,n,k}^{DA}$. After clarifying the edge set and vertex set of the bipartite matching graph, we can select by the weight value of the edge in the bipartite matching graph, and then solve the above optimization problem.

5.3. Trust-aware Caching Constrained Task Offloading Algorithm

According to our analysis mentioned in Section 5.2, the trust-aware caching constrained task offloading algorithm based on bipartite graph is proposed. The proposed algorithm is presented in Algorithm 1.

Algorithm 1 works as follows: variables and characterization formula are initialized based on the task set T, the device set D, the DT set S and the collection of all tasks A. Then, the task placement and caching process include the following steps: calculating the unit popularity of each

Algorithm 1 Trust-aware Caching Constrained Task Offloading Algorithm

Input: The task set T, the device set D, the DT set S, the collection of all tasks A

Output: The optimized task offloading strategy scheme W

Initialization: Define variables and characterization formula, and generate the corresponding bipartite graph $G(V, E)$;

1: for $s = 1 : S$ do
2: Calculate the unit popularity η_a of task a, and the task set A in descending order according to the unit popularity η_a;
3: for $a = 1 : A$ do
4: if $X_s d^T < H_U$ then
5: Cache the result of task a on DT s ;
6: else
7: Break;
8: end if
9: end for
10: end for
11: for $t = 1 : T$ do
12: Calculate the Z_t^L, $Z_{t,k}^C$, $Z_{t,n}^{D2D}$, $Z_{t,n,k}^{DA}$, according to the formula (26), (27), (28), (29);
13: Make the weight ω of the edge in the bipartite graph $G(V, E)$ equal to the Z value under the corresponding task offloading strategy;
14: end for
15: for $t = 1 : T$ do
16: for $e = 1 : E$ do
17: if task t was not the matched or conflict with device vertices then
18: Use binary perfect matching algorithm (Algorithm 2) [27] to find matching vertices;
19: end if
20: end for
21: end for
22: for $e = 1 : E$ do
23: Add to strategy set W;
24: end for

task, and sorting each task according to the unit popularity from large to small. Then, according to the greedy strategy of unit popularity priority, the tasks are cached in the node memory in sequence until all tasks are cached or the remaining memory capacity is empty (Line 1-10). Then, the bipartite graph $G(V, E)$ is constructed according to the device social graph and the D2D connectivity graph. After that, we traverse the tasks in the task set, and for every task, calculate the time delay and energy consumption of executing the task under each strategy, and finally finding the overhead of executing the task under each strategy, including local offloading, directly to cloud offloading, D2D offloading and D2D-assisted to cloud offloading. After obtaining the result, the weights of the edges of the bipartite graph are assigned, and the weight value of the corresponding edge is the overhead metric calculated by the execution of the task under the corresponding strategy (Line 11-14). The next procedure is the selection part of the task offloading strategy. The edges of each task vertex in the bipartite graph are traversed with binary perfect matching algorithm (Algorithm 2) [27] to find matching vertices in the bipartite graph, and delete other edges. The above process is cyclically executed until all vertices in the task set have found their matching strategies. At this time, the corresponding strategy selection is also the set of edges with the smallest weight value (Line 15-21).

Algorithm 2 Trust-aware Bipartite Perfect Matching Based on K-M Algorithm

Input: Bipartite graph $G(V, E)$.

Output: The matched bipartite graph $G'(V, E)$.

Initialization: Negate the weights of edges in a bipartite graph $G(V, E)$. And set V can divide into two subset (T, N)

1: for $t = 1 : T$ do
2: Make the value c_t of vertex in the starting set T the maximum value of weight among all the edges connected with it
3: end for
4: for $n = 1 : N$ do
5: Assign $c_n = 0$ to the vertex value in the matching set N, which meets the criterion $c_t + c_n = \omega_{t,n}$
6: end for
7: for $t = 1 : T$ do
8: if Task t was conflict with other nodes then
9: Modify the value of vertices and re-match.
10: else
11: Match the task vertices in the starting set.
12: end if
13: end for

The process of selecting a task offloading strategy based on bipartite graph perfect matching is shown in Algorithm 2. To simplify the matching process, we convert the problem of minimum weight matching into maximum weight matching. So we take the inverse of the weight of the edges. In this way, the above problems can be solved by bipartite graph maximum weight perfect matching. The main steps to solve maximum perfect matching problem are as follows: the first step is to initialize the value of the vertices, the value c_n of the device vertex (the matching set) is set as 0, and the value of the task vertex c_t (the starting set) is set as the maximum value after taking the inverse of the weight of the edge connected to it, thus the criterion $c_n + c_t = \omega_{n,t}$ is satisfied (Line 1-6). The second step is to select the task vertices in the starting vertex set for matching. If a conflict occurs, we modify the vertex value and re-match until all vertices in the starting set are matched (Line 7-13). Then we assume that matched edge is the optimal offloading strategy for the task.

5.4. Analysis of the Proposed Algorithm

Computational complex analysis: Obviously, the time complexity of the trust-aware caching constrained task offloading algorithm is related to the number of edges and vertices of the bipartite graph when solving the problem of bipartite graph maximum weight perfect matching. So the time complexity of the bipartite matching algorithm is $O(T|E|)$, where T represents the quantity of vertices in the starting set, $|E|$ is the number of edges. However, the number of edges is proportional to the number of vertices in our problem, so the proposed algorithm's time complexity in the worst case is $O(T^3)$ [22].

Scalability analysis: The scalability of bipartite graph matching algorithm mainly depends on the efficiency of the algorithm when processing large-scale data. The Hungarian algorithm is implemented by depth-first search (DFS) and has a time complexity of $O(T^3)$, where T is the number of vertices. For sparse graphs, this algorithm performs well. For dense graphs, however, performance may degrade. In order to improve the efficiency of the algorithm, some heuristics optimization techniques, such as preprocessing and pruning, are used when the network scale is large. For example, in this paper when the network scale is large, we cluster the devices according to the trust relationship and physical distance between the devices to achieve network preprocessing. By dividing the MEC network into several small-scale networks and tasks are offloaded within the cluster, the amount of computation can be decreased by reducing unnecessary searches. Finally, we carry out simulation experiments for large-scale networks comparing the execution time of algorithm to prove the effectiveness of network preprocessing in dealing with the problem of increasing computational complexity in large-scale networks. We compare the algorithm execution time of a large-scale network with and without the use of a network preprocessing algorithm under different numbers of devices whish is shown as Table 2, focusing on scenarios involving numerous devices and tasks, and we set the number of tasks is 1000.

Table 2: Convergence time comparison

	225	350	450
Preprocessed network	283.4051s	628.6768s	996.0353s
Unpreprocessed network	357.4183s	1285.3927s	1309.6373s

Sensitivity analysis: To evaluate the robustness of the proposed model, we conduct a sensitivity analysis on the weighting factor θ that balances delay and energy consumption in the objective function. By varying θ from 0.1 to 1, we observe the corresponding changes in system overhead function, which is shown in the simulation experienment Exp-3 Figure 10(c). The result demonstrates the impact of different θ values on system performance and provide insights into the model's behavior under varying optimization preferences.

6. Simulation and Performance Evaluation

In this section, the results and performance of the proposed trust-aware caching constrained task offloading scheme is verified and evaluated through simulation experiments.

6.1. Simulation Settings

The simulation experiments are carried out on a computer equipped with Intel Core i5-1135G7 CPU 2.40GHz 2.42GHz, and Windows 11 system. We implement our proposed algorithm using Python in the Pycharm 2021.3 environment. We assume a circular area with a radius of 100m, and the BS is located in the center of the circular area. Correspondingly, NRs and DTs are distributed in annular areas with a radius of $50-100m$ and $10-50m$, respectively [23]. The initial value settings of related variables and parameters are shown in Table 3 based on the researches [23, 48, 14, 22]. We calculated the overhead metric Z values of different task offloading strategies in various situations representing the trust between devices and the delay and energy consumption.

Table 3: The Parameters Initialization

	Parameters	Values
1	$P_{U,S}$	28 dBm
2	P_B	46 dBm
3	P_U	46 dBm
4	W_1	8 MHz
5	W_2	10 MHz
6	λ_u	1 GHz
7	λ_B	40 GHz
8	σ^2	-100 dBm
9	α	4
10	Λ	3000 GB
11	Ω	5000 GB
12	V	-174 dBm/Hz
13	θ	0.5
14	d_j	$[1, 50]$ MB
15	$l_{m,n,U}$	$[1, 50]$ m
16	$l_{m,k,B}$	$[50, 100]$ m

6.2. Comparison Schemes

In order to fairly compare the results of simulation experiments and conduct analysis and evaluation, we choose the following approaches and strategies for comparative experiments.

- GA: Genetic Algorithm (GA) is an optimization method inspired by natural evolution [49]. It searches for the optimal solution through selection, crossover, and mutation. Each chromosome represents task offloading decisions, and the best one is selected as the final solution [50].

- DRL: DRL-based algorithm (DRL) solves the problem of various task offloading strategies, which is aiming at generating an offloading decision in MEC network [51]. It learns a policy network (Actor) and a value function network (Critic) at the same time, and improves the decision-making ability of the agent by optimizing both networks [52].
- W/O-Soc: The task offloading strategy without the consideration of social (W/O-Soc). Under this approach, the social relationship between nodes is not considered, that is, nodes do not trust each other.
- Local: The approach that all task execute on local device nodes (Local). In this way, all the tasks will be executed on the nodes that they generate.
- Cloud: The method that all tasks execute on BSs (Cloud). All the tasks executed by this way will offload to the nearest BS to execute.
- Random: Under this scheme (Random), multiple task offloading strategies are randomly generated based on the number of tasks.
- Proximity: Under the Proximity scheme, task offloading strategies are designed based on the physical or network closeness of devices. Tasks are offloaded to nearby devices with available resources to improve efficiency and reduce latency [53].
- TCTO (Ours): A trust-aware caching constrained task offloading strategy (Ours) proposed in this paper comprehensively considers the latency, energy consumption and trust between nodes representing content privacy and security for each offloading strategy. And the offloading strategies are selected using binary perfect matching algorithm. We compare the Z values and select the strategy corresponding to the minimal Z value as the task offloading strategy under this scheme.

6.3. Simulation Results and Performance Evaluation

In this chapter, we conduct a comparative simulation experiment and performance evaluation of the algorithm.

6.3.1. Algorithm convergence time comparison

Firstly, we discuss and evaluate the convergence speed of our proposed algorithm and different algorithms. The average convergence speed and time of each algorithm are compared as Table 4. Since the proposed bipartite matching algorithm is non-iterative and can obtain the optimal solution in a single pass, we evaluate its execution time. For comparison, we also present the convergence times of iterative algorithms such as DRL, TD3 and GA. Among the baseline algorithms, TD3 (Twin Delayed Deep Deterministic Policy Gradient) is a reinforcement learning method known for its stability in continuous control tasks. It uses twin critic networks and delayed updates to reduce overestimation and improve learning [54].

Table 4: Convergence time comparison

	Ours	DRL	TD3	GA
50 *Tasks*	0.2253	1.7587	1.2174	1.2887
100 *Tasks*	0.4242	4.9415	3.9456	4.4382
200 *Tasks*	0.7785	16.0491	17.7317	14.5797
300 *Tasks*	1.1729	33.8548	37.5643	36.2252
400 *Tasks*	1.4761	71.9385	61.5111	74.6866
500 *Tasks*	1.9501	112.5215	86.6081	101.1736

6.3.2. Memory replacement algorithm comparison

Moreover, considering that different memory replacement policies can affect task scheduling in memory, we compare the performance of our algorithm under various replacement strategies including FIFO (First-In-First-Out), LRU (Least Recently Used), and LFU (Least Frequently Used) with different numbers of tasks. As shown in the Figure 9, although the performance differences among the three strategies are not significant, the LRU replacement policy demonstrates slightly better performance in optimizing the overhead function of the algorithm. Therefore, the LRU strategy is adopted in this work to evict tasks from memory and make room for newly arriving tasks.

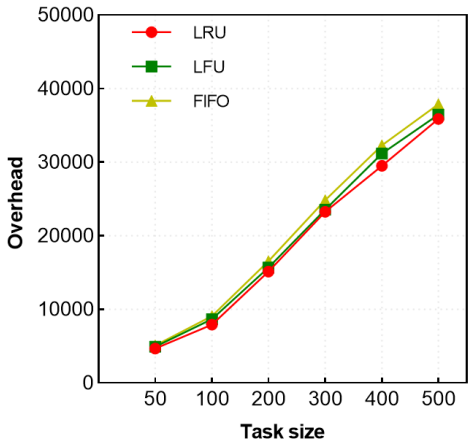
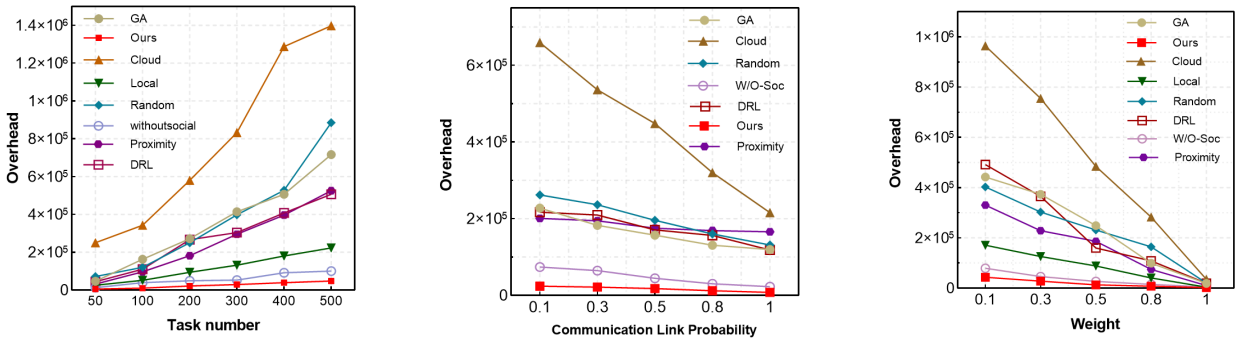


Figure 9: Memory replacement algorithm comparison

Nextly, we compare different task offloading strategies through simulation experiments, and then evaluate and analyze the experimental results.

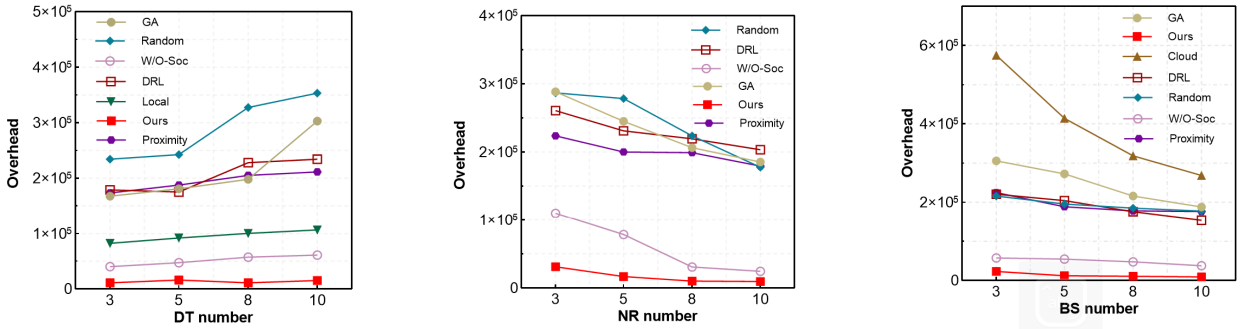
6.3.3. Exp-1: Overhead *vs.* Task number

In Figure 10(a), we show the overhead of different task offloading strategies under different task numbers, with 5 DTs, 5 NRs and 3 base stations. And our proposed approach decreases the overhead compared with others baselines which represents our proposed approach reduces the delay and energy consumption in the task offloading procedure under different task quantities. Especially, from Figure 10(a), it can be seen that the overhead of our proposed



(a) Task number (b) Communication link probability (c) Weight

Figure 10: Overhead under different parameters



(a) DT number (b) NR number (c) BS number

Figure 11: Overhead under different number of devices

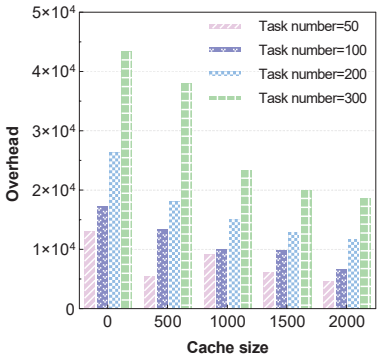
method is smaller than that of the method W/O-Soc, representing that trust awareness plays an important role in reducing delay and energy consumption of task transmission and execution. And we can find that in Figure 10(a) the overhead of our proposed method increases slowly compared with other methods with the increase of task numbers which shows that our proposed method performs stably facing enormous number of tasks.

6.3.4. Exp-2: Overhead *vs.* Link probability

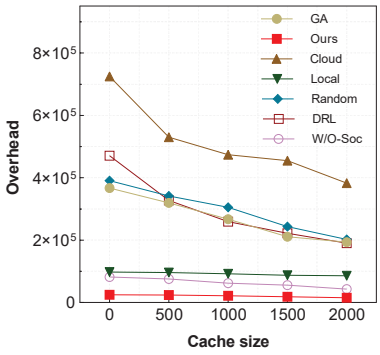
In Figure 10(b), we compare the overhead of different task offloading strategies under different link probabilities. The link probability represents the likelihood of a communication channel existing between two devices, and it can be analogized to social relationship link probability. In this analogy, the link probability of social relationships refers to the probability of maintaining effective interactions between individuals, reflecting the likelihood of establishing and sustaining relationships. As social interactions increase, each device benefits more from cooperative task offloading, as more devices with established social ties become available to assist, leading to more efficient task offloading. Owing to the task executed by the local offloading strategy is not influenced by the other nodes, we thus exclude it from consideration. The results of this simulation experiment are shown in Figure 10(b). And the overhead decreases while the link probability increases which means that more links between devices can effectively alleviate network congestion as well as reduce delay and energy consumption. The magnitude of the drop under the strategy of directly to cloud offloading and random offloading are more pronounced as the link probability increases. The overhead of our method changes more stably as the link probability changes, which means that our proposed method is less affected by link changes. And the curve of our proposed method is overall below the curves of other strategies which indicates that our method has the smallest delay and energy consumption compared with other strategies with the same link probability.

6.3.5. Exp-3: Overhead *vs.* Weight of delay and energy

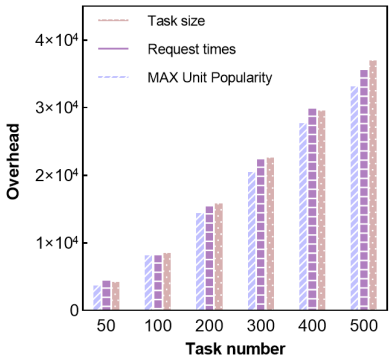
In this simulation experiment, the overhead of different approaches is compared under different value of weight of delay and energy consumption, which is shown as Figure 10(c). The weight value represents the impact of the delay on the task offloading strategy selection. In other words, the weight value corresponding to delay-sensitive tasks is larger, and the weight value corresponding to energy-sensitive tasks is smaller. The experiment result indicates that our method has advantages in reducing the delay and energy consumption compared with the other baselines under different values of weight.



(a) Overhead under different cache size with different task number



(b) Overhead under different cache size under different baseline



(c) Overhead under different caching strategy and different task numbers

Figure 12: Overhead under different cache size and caching strategy

6.3.6. Exp-4: Overhead *vs.* DT number

Figure 11(a) illustrates the overhead of these methods as the number of DTs [14] changes when the number of NRs is 5 and the number of tasks is 200. Because in the DC offloading strategy, the change of overhead is not affected by the change of the number of DTs, so it is not considered. The result shows that the overall trend is a downward trend with the increase of the number of DTs. And the overhead of our proposed method is below other curves of other algorithms, representing that the tasks executed by our proposed method has the least cost of delay and energy with the same number of DT in the MEC network. Compared to the *Local* strategy, it can be seen that the Z value of our proposed approach is lower than the strategy of executing on local device nodes, which indicates that the D2D and D2D-assisted to cloud offloading strategy can reduce delay and energy consumption in the process of task execution and retrieve the pressure of network.

6.3.7. Exp-5: Overhead *vs.* NR number

Correspondingly, Figure 11(b) shows the result of overhead under different task offloading strategies when the number of NRs [14] changes, where the number of DTs is 5 and the task number is 200. Since under the *Local* and *Cloud* offloading strategies, the NR devices do not participate in task offloading. Therefore, we only compare the overhead of the other methods under different NR numbers. And our proposed method performs the least overhead compared with other strategies under different NR numbers. The curve of random task offloading strategy is higher than others which means that our proposed approach can reduce the delay and energy consumption in the process of task transmission and execution under different quantity of NRs.

6.3.8. Exp-6: Overhead *vs.* BS number

Then we compare the overhead of different baselines under different number of BSs. And we do not consider the *Local* method because the changing of the number of BSs will not impact the overhead of local method. The result is depicted as Figure 11(c). We can find that our approach has least overhead under different MEC numbers. And the overhead of *Cloud* approach has a significant decline while the number of BSs increases, which means the increase of MEC severs will reduce the pressure of MEC networks.

6.3.9. Exp-7: Overhead *vs.* Cache size (Different task number)

Figure 12(a) simulates the variation of the overhead when varying the cache size of user device under different task numbers and the MEC scenario with 5 DTs, 5 NRs and 3 BSs. The simulation results illustrate that the value of overhead decreases with the increase of the cache capacity of device nodes under different task numbers. This demonstrates that the cache of devices can effectively avoid the delay and energy consumption caused by task transmission. In addition, with the increase of cache capacity, the number of tasks cached by devices increases, which represents that the hit ratio of tasks on devices increases during task offloading. In particular, compared with the circumstance that cache size is 0 which implies the cache is not considered, we find that our proposed method with the consideration of caching can significantly reduce the consumption of delay and energy in the process of transmission under different task number.

6.3.10. Exp-8: Overhead *vs.* Cache size (Under different approach)

Then, we conduct the experiment of the comparison of overhead under different cache size of different approaches. The simulation result is delineated as Figure 12(b). We can find that with the increase of cache size, the overall value of overhead shows a decreasing trend. Our method performs stably, which indicates that our method is more conducive to dealing with different cache changes compared with other baselines, especially the *Cloud* and *DRL*.

6.3.11. Exp-9: Overhead *vs.* Task numbers (Under different caching approach)

We compare the system overhead under different numbers of tasks and various task caching strategies, shown in Figure 12(c). In our approach, tasks are cached on devices based on their maximum unit popularity. For comparison, we also consider two alternative strategies: caching tasks in order of task size and in order of request times. Experimental results show that the maximum unit popularity-based strategy consistently achieves better performance in reducing system overhead across different task numbers.

6.3.12. Exp-10: Overhead *vs.* Task Repeat Probability

We analyze the impact of the probability of repeated task requests on the overall system overhead. Experiments shown in Figure 13(a) demonstrate that as the probability of repeated task requests increases, the system overhead decreases. A higher probability of repeated task requests indicates that a task may be requested multiple times within the network. Leveraging caching mechanisms, repeated task requests can directly return the task results, thereby reducing task execution time and energy consumption, which in turn lowers system overhead. We compare the system overhead under different offloading schemes, and the experimental results show that our proposed approach still maintains an advantage in algorithmically reducing system overhead.

6.3.13. Exp-11: Task Fail Rate *vs.* Task Deadline

In this experiment, shown in Figure 13(b), the failure rate of the whole system task offloading under different task deadline is compared. When the task offloading delay exceeds the deadline, the task offloading fails. In this experiment, the number of tasks is 200. When the deadline of a task is limited to $50ms - 110ms$, the task failure rate decreases with the increase of the deadline. At the same time, our proposed scheme has the lowest probability of task offloading fail rate within the same deadline, indicating that under our proposed scheme, the task offloading success rate is the highest and the delay is the smallest.

6.4. Discussion

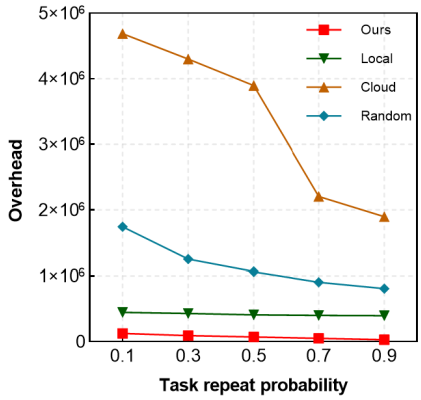
The above research findings, based on extensive simulations, highlight several key advantages and limitations of the proposed trust-aware caching-constrained task offloading strategy in MEC networks. Firstly, the proposed task offloading strategy in MEC networks effectively reduces latency and energy consumption, particularly in high connectivity and large-cache scenarios. It consistently outperforms baseline strategies across various task sizes, link probabilities, and energy-delay trade-offs, demonstrating adaptability to different task requirements. However, as it has not yet been tested on physical edge devices, real-world constraints may present implementation challenges. Additionally, the strategy dependence on social awareness data and cache capacity may limit performance in settings with limited social data or low cache. Future work will focus on real-device testing and further optimization to enhance practical applicability in MEC network.

However, since the TCTO strategy has not yet been tested on real devices such as the application in smart cities, UAV networks and so on, real-world constraints may pose significant challenges, such as limited availability of trust awareness data, which could affect the accuracy of predictions, especially in environments with sparse or fragmented data. Additionally, the strategy's reliance on large cache capacities may be constrained by the memory limitations of real devices, necessitating efficient cache management techniques. Network variability, including fluctuations in bandwidth and connectivity, could impact task offloading performance, requiring adaptive mechanisms to cope with dynamic network conditions. Moreover, while the TCTO strategy reduces energy consumption in simulations, real-world devices with limited battery life will need solutions to balance energy usage and task offloading, possibly through energy harvesting. Finally, scalability concerns in large-scale MEC deployments, such as those in smart cities mentioned in the introduction, must be addressed to manage computational overhead and network load. Future work will focus on TCTO in real-world scenarios, refining social data collection methods, optimizing cache management, and developing adaptive solutions for network and energy constraints.

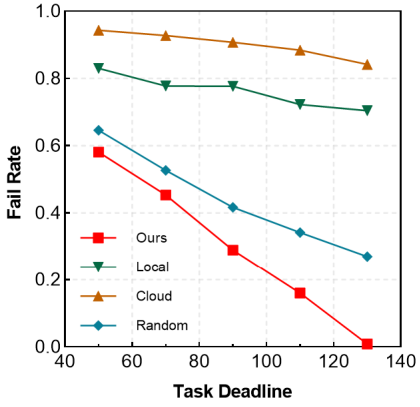
7. Conclusion

In this paper, we comprehensively consider the role of trust awareness, caching, and task offloading to optimize the reduction of latency and energy consumption in the MEC networks. A trust-aware task offloading strategy with cache constraints is proposed, which avoids the possibility of repeated execution of tasks requested multiple times and provides a more flexible task offloading strategy. Then, a trust-aware caching-constrained bipartite graph matching algorithm is devised to select the optimal task offloading strategy. Therefore, compared with schemes that partially consider some of these factors, our proposed scheme can more effectively reduce the latency and energy consumption of task execution and transmission. Extensive simulations show that our approach has a significant effect on reducing both delay and energy consumption. The proposed approach reduces the overhead 55.65% ~ 96.20% compared with other task offloading strategies. And we find that the caching plays an important role in reducing overhead based on extensive simulation results.

As a future direction, we consider introducing an adaptive adjustment mechanism for the weight of delay and energy, allowing it to change dynamically based on real-time network conditions to improve system flexibility and performance. In the future, we will use deep learning, reinforcement learning and other related theories to improve



(a) Overhead of tasks with different repeat request probabilities



(b) The fail rate of tasks under different deadlines

Figure 13: Comparison under different task parameters

efficiency for the selection of task offloading and caching based on trust awareness.

CRediT authorship contribution statement

Xinyuan Zhu: Conceptualization, Investigation, Resources, Writing-Original Draft, Writing-review and Editing. Fei Hao: Investigation, Supervision, Writing-Review and Editing; Ming Lei: Formal Analysis, Writing-Review and Editing. Aziz Nasridinov: Writing-Review and Editing. Jiaxing Shang: Writing-Review and Editing; Zhengxin Yu: Writing-Review and Editing; Longjiang Guo: Supervision, Writing-Review and Editing.

Acknowledgment

This work was funded in part by the National Natural Science Foundation of China (Grant Nos. 62477029, 61702317), the Ministry of Education Humanities and Social Sciences Research Youth Fund Project (Grant No. 22YJCZH046).

References

[1] V. A. S, N. E, V. G. R, Performance evaluation of mobile edge computing using 5g networks, in: 2021 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT), 2021, pp. 1–6. doi:10.1109/CONECCT52877.2021.9622523.

[2] Y. Liu, M. Peng, G. Shou, Y. Chen, S. Chen, Toward edge intelligence: Multiaccess edge computing for 5g and internet of things, IEEE Internet of Things Journal 7 (8) (2020) 6722–6747. doi:10.1109/JIOT.2020.3004500.

[3] J. Liu, X. Zhao, P. Qin, S. Geng, S. Meng, Joint dynamic task offloading and resource scheduling for wpt enabled space-air-ground power internet of things, IEEE Transactions on Network Science and Engineering 9 (2) (2022) 660–677. doi:10.1109/TNSE.2021.3130251.

[4] X. Yang, Z. Fei, J. Zheng, N. Zhang, A. Anpalagan, Joint multi-user computation offloading and data caching for hybrid mobile cloud/edge computing, IEEE Transactions on Vehicular Technology 68 (11) (2019) 11018–11030. doi:10.1109/TVT.2019.2942334.

[5] Z. Liu, J. Fan, S. Geng, P. Qin, X. Zhao, Joint optimization of task offloading and computing resource allocation in mec-d2d network, in: 2022 IEEE 5th International Conference on Computer and Communication Engineering Technology (CCET), 2022, pp. 256–260. doi:10.1109/CCET55412.2022.9906362.

[6] N. Fan, X. Wang, D. Wang, Y. Lan, J. Hou, A collaborative task offloading scheme in d2d-assisted fog computing networks, in: 2020 IEEE Wireless Communications and Networking Conference (WCNC), 2020, pp. 1–6. doi:10.1109/WCNC45663.2020.9120662.

[7] M. Tong, X. Wang, Y. Wang, Y. Lan, Computation offloading scheme with d2d for mec-enabled cellular networks, in: 2020 IEEE/CIC International Conference on Communications in China (ICCC Workshops), 2020, pp. 111–116. doi:10.1109/ICCCWorkshops49972.2020.9209940.

[8] C. Tang, C. Zhu, X. Wei, H. Wu, Q. Li, J. J. P. C. Rodrigues, Task offloading and caching for mobile edge computing, in: 2021 International Wireless Communications and Mobile Computing (IWCMC), 2021, pp. 698–702. doi:10.1109/IWCMC51323.2021.9498725.

[9] Y. Sun, W. Chen, Y. Yuan, Research of caching methods in mobile edge networks, in: 2021 3rd International Academic Exchange Conference on Science and Technology Innovation (IAECST), 2021, pp. 521–524. doi:10.1109/IAECST54258.2021.9695888.

[10] M. R. Azeem, S. M. Muzammal, N. Zaman, M. A. Khan, Edge caching for mobile devices, in: 2022 14th International Conference on Mathematics, Actuarial Science, Computer Science and Statistics (MACS), 2022, pp. 1–6. doi:10.1109/MACS56771.2022.10022729.

[11] X. Zhang, Y. Liu, J. Liu, A. Argyriou, Y. Han, D2d-assisted federated learning in mobile edge computing networks, in: 2021 IEEE Wireless Communications and Networking Conference (WCNC), 2021, pp. 1–7. doi:10.1109/WCNC49053.2021.9417459.

[12] M. Ahmed, Y. Li, M. Waqas, M. Sheraz, D. Jin, Z. Han, A survey on socially aware device-to-device communications, IEEE Communications Surveys and Tutorials 20 (3) (2018) 2169–2197. doi:10.1109/COMST.2018.2820069.

[13] J. Xu, L. Chen, K. Liu, C. Shen, Designing security-aware incentives for computation offloading via device-to-device communication, IEEE Transactions on Wireless Communications 17 (9) (2018) 6053–6066. doi:10.1109/TWC.2018.2854579.

[14] H. Long, C. Xu, G. Zheng, Y. Sheng, Socially-aware energy-efficient task partial offloading in mec networks with d2d collaboration, IEEE Transactions on Green Communications and Networking 6 (3) (2022) 1889–1902. doi:10.1109/TGCN.2022.3153956.

[15] A. Alioua, N. Bouchemal, R. Mati, M.-L. Messai, Blockchain-inspired incentive mechanism for trust-aware offloading in mobile edge computing, in: 2024 IEEE 49th Conference on Local Computer Networks (LCN), 2024, pp. 1–8. doi:10.1109/LCN60385.2024.10639661.

[16] C.-H. Wang, J.-J. Kuo, D.-N. Yang, W.-T. Chen, Collaborative social internet of things in mobile edge networks, IEEE Internet of Things Journal 7 (12) (2020) 11473–11491. doi:10.1109/JIOT.2020.3018304.

[17] Y. Gao, W. Tang, M. Wu, P. Yang, L. Dan, Dynamic social-aware computation offloading for low-latency communications in iot, IEEE Internet of Things Journal 6 (5) (2019) 7864–7877. doi:10.1109/JIOT.2019.2909299.

[18] P. Dong, J. Ge, X. Wang, S. Guo, Collaborative edge computing for social internet of things: Applications, solutions, and challenges, IEEE Transactions on Computational Social Systems 9 (1) (2022) 291–301. doi:10.1109/TCSS.2021.3072693.

[19] Q. V. Pham, B. L. Long, S. H. Chung, W. J. Hwang, Mobile edge computing with wireless backhaul: Joint task offloading and resource allocation, IEEE Access 7 (99) (2019) 16444–16459.

[20] L. Wang, G. Zhang, Joint service caching, resource allocation and computation offloading in three-tier cooperative mobile edge computing system, IEEE Transactions on Network Science and Engineering (2023) 1–11doi:10.1109/TNSE.2023.3259030.

[21] L. Kang, B. Tang, L. Zhang, L. Tang, Mobility-aware and data caching-based task scheduling strategy in mobile edge computing, in: 2019 IEEE Intl Conf on Parallel and Distributed Processing with Applications, Big Data Cloud Computing, Sustainable Computing and Communications, Social Computing and Networking, 2019, pp. 1071–1077. doi:10.1109/ISPA-BDCloud-SustainCom-SocialCom48970.2019.00153.

[22] X. Chen, Z. Zhou, W. Wu, D. Wu, J. Zhang, Socially-motivated cooperative mobile edge computing, IEEE Network 32 (6) (2018) 177–183. doi:10.1109/MNET.2018.1700354.

[23] B. Fan, Y. Jiang, F.-C. Zheng, M. Bennis, X. You, Social-aware cooperative caching in fog radio access networks, in: ICC 2022 - IEEE International Conference on Communications, 2022, pp. 1672–1677. doi:10.1109/ICC45855.2022.9839236.

[24] X. Liu, C. Sun, X. Zhang, Context-aware caching with social behavior in mec-enabled wireless cellular networks, in: 2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), 2019, pp. 1004–1008. doi:10.1109/PERCOMW.2019.8730879.

[25] V. Deka, A. Islam, M. Ghose, Cloud-assisted dynamic and cooperative content caching in mobile edge computing, in: 2022 IEEE 19th India Council International Conference (INDICON), 2022, pp. 1–6. doi:10.1109/INDICON56171.2022.10039991.

[26] X. Zhou, D. Pan, H. Song, X. Huang, Socially-aware d2d pair strategy: A stable matching approach, in: 2020 IEEE 39th International Performance Computing and Communications Conference (IPCCC), 2020, pp. 1–4. doi:10.1109/IPCCC50635.2020.9391547.

[27] C. Xu, C. Xu, B. Li, S. Li, T. Li, Joint social-aware and mobility-aware computation offloading in heterogeneous mobile edge computing, IEEE Access 10 (2022) 28600–28613. doi:10.1109/ACCESS.2022.3158319.

[28] L. Liu, Z. Chang, X. Guo, Socially aware dynamic computation offloading scheme for fog computing system with energy harvesting devices, IEEE Internet of Things Journal 5 (3) (2018) 1869–1879. doi:10.1109/JIOT.2018.2816682.

[29] M. Li, L. Yang, K. Lu, S. B. H. Shah, X. Zheng, Device-to-device task offloading in a stochastic invalid-device scenario with social awareness, in: 2022 IEEE Conference on Dependable and Secure Computing (DSC), 2022, pp. 1–7. doi:10.1109/DSC54232.2022.9888905.

[30] M. Ibrar, L. Wang, A. Akbar, M. A. Jan, V. Balasubramanian, G.-M. Muntean, N. Shah, Adaptive capacity task offloading in multi-hop d2d-based social industrial iot, IEEE Transactions on Network Science and Engineering (2022) 1–9doi:10.1109/TNSE.2022.3192478.

[31] Y. Gao, W. Tang, M. Wu, P. Yang, L. Dan, Dynamic social-aware computation offloading for low-latency communications in iot, IEEE Internet of Things Journal 6 (5) (2019) 7864–7877. doi:10.1109/JIOT.2019.2909299.

[32] Y. Gong, F. Hao, L. Wang, L. Zhao, G. Min, A socially-aware dependent tasks offloading strategy in mobile edge computing, IEEE Transactions on Sustainable Computing (2023) 1–14doi:10.1109/TSUSC.2023.3240457.

[33] Z. Hajiakhondi-Meybodi, M. Hou, A. Mohammadi, Content placement in a cluster-centric mobile edge caching network, in: 2022 IEEE International Conference on Networking, Sensing and Control (ICNSC), 2022, pp. 1–5. doi:10.1109/ICNSC55942.2022.10004115.

[34] Z. H. Meybodi, A. Mohammadi, E. Rahimian, S. Heidarian, J. Abouei, K. N. Plataniotis, Tedge-caching: Transformer-based edge caching towards 6g networks, in: ICC 2022 - IEEE International Conference on Communications, 2022, pp. 613–618. doi:10.1109/ICC45855.2022.9838981.

[35] X. Wei, J. Liu, Y. Wang, C. Tang, Y. Hu, Wireless edge caching based on content similarity in dynamic environments, J. Syst. Archit. 115 (2021) 102000. doi:10.1016/j.sysarc.2021.102000. URL https://doi.org/10.1016/j.sysarc.2021.102000

[36] Y. Bai, D. Wang, B. Song, A knowledge graph-based cooperative caching scheme in mec-enabled heterogeneous networks, in: GLOBECOM 2022 - 2022 IEEE Global Communications Conference, 2022, pp. 5959–5964. doi:10.1109/GLOBECOM48099.2022.10001575.

[37] Y. Bai, D. Wang, G. Huang, B. Song, A deep reinforcement learning-based social-aware cooperative caching scheme in d2d communication networks, IEEE Internet of Things Journal (2023) 1–1doi:10.1109/JIOT.2023.3234705.

[38] Y. Liao, X. Qiao, L. Shou, X. Zhai, Q. Ai, Q. Yu, Q. Liu, Caching-aided task offloading scheme for wireless body area networks with mec, in: 2019 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), 2019, pp. 49–54. doi:10.1109/AHS.2019.000-1.

[39] H. Feng, S. Guo, L. Yang, Y. Yang, Collaborative data caching and computation offloading for multi-service mobile edge computing, IEEE Transactions on Vehicular Technology 70 (9) (2021) 9408–9422. doi:10.1109/TVT.2021.3099303.

[40] J. Li, X. You, J. Zheng, Performance modeling and analysis of an mec system with task priority and expiring time constraint, IEEE Communications Letters 27 (7) (2023) 1754–1758. doi:10.1109/LCOMM.2023.3270338.

[41] X. Jiao, H. Ou, S. Chen, S. Guo, Y. Qu, C. Xiang, J. Shang, Deep reinforcement learning for time-energy tradeoff online offloading in mec-enabled industrial internet of things, IEEE Transactions on Network Science and Engineering 10 (6) (2023) 3465–3479. doi:10.1109/TNSE.2023.3263169.

[42] L. Huang, S. Bi, Y.-J. A. Zhang, Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks, IEEE Transactions on Mobile Computing 19 (11) (2020) 2581–2593. doi:10.1109/TMC.2019.2928811.

[43] X. Pei, W. Duan, M. Wen, Y.-C. Wu, H. Yu, V. Monteiro, Socially aware joint resource allocation and computation offloading in noma-aided energy-harvesting massive iot, IEEE Internet of Things Journal 8 (7) (2021) 5240–5249. doi:10.1109/JIOT.2020.3034380.

[44] S. Yu, B. Dab, Z. Movahedi, R. Langar, L. Wang, A socially-aware hybrid computation offloading framework for multi-access edge computing, IEEE Transactions on Mobile Computing 19 (6) (2020) 1247–1259. doi:10.1109/TMC.2019.2908154.

[45] Q.-V. Pham, L. B. Le, S.-H. Chung, W.-J. Hwang, Mobile edge computing with wireless backhaul: Joint task offloading and resource allocation, IEEE Access 7 (2019) 16444–16459. doi:10.1109/ACCESS.2018.2883692.

[46] J. Zhao, Q. Li, Y. Gong, K. Zhang, Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks, IEEE Transactions on Vehicular Technology 68 (8) (2019) 7944–7956. doi:10.1109/TVT.2019.2917890.

[47] Y. Sun, L. Xu, Y. Tang, W. Zhuang, Traffic offloading for online video service in vehicular networks: A cooperative approach, IEEE Transactions on Vehicular Technology 67 (8) (2018) 7630–7642. doi:10.1109/TVT.2018.2837024.

[48] D. Wu, Q. Liu, H. Wang, D. Wu, R. Wang, Socially aware energy-efficient mobile edge collaboration for video distribution, IEEE Transactions on Multimedia 19 (10) (2017) 2197–2209. doi:10.1109/TMM.2017.2733300.

[49] S. C. Sinthiya, N. Imtiaz Shuvo, R. R. Mahmud, J. Ahmed, Low-cost task offloading scheme for mobile edge cloud and internet cloud using genetic algorithm, in: 2022 4th International Conference on Sustainable Technologies for Industry 4.0 (STI), 2022, pp. 1–6. doi:10.1109/STI56238.2022.10103247.

[50] J. Tian, D. Wang, H. Zhang, D. Wu, Service satisfaction-oriented task offloading and uav scheduling in uav-enabled mec networks, IEEE Transactions on Wireless Communications 22 (12) (2023) 8949–8964. doi:10.1109/TWC.2023.3267330.

[51] S. Nath, J. Wu, Deep reinforcement learning for dynamic computation offloading and resource allocation in cache-assisted mobile edge computing systems, Intelligent and Converged Networks 1 (2) (2020) 181–198. doi:10.23919/ICN.2020.0014.

[52] J. Fang, D. Qu, H. Chen, Y. Liu, Dependency-aware dynamic task offloading based on deep reinforcement learning in mobile edge computing, IEEE Transactions on Network and Service Management (2023) 1–1doi:10.1109/TNSM.2023.3319294.

[53] K. Sharmila, V. Mohan, C. Ramesh, S. P. Munda, Proximity services based device-to-device framework design for direct discovery, in: 2016 2nd International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEICB), 2016, pp. 499–502. doi:10.1109/AEEICB.2016.7538340.

[54] A. Mohajer, J. Hajipour, V. C. M. Leung, Dynamic offloading in mobile edge computing with traffic-aware network slicing and adaptive td3 strategy, IEEE Communications Letters 29 (1) (2025) 95–99. doi:10.1109/LCOMM.2024.3501956.