A Study on Task Offloading and Service Caching Integration in Edge-Cloud Collaborative Systems

Yuhua Li [1,*] and Yancong Deng [2]
[1] Ming Hsieh Department of Electrical and Computer Engineering,
University of Southern California, Los Angeles, United States
[2] Electrical and Computer Engineering, University of California, San Diego, La Jolla, United States
E-mail: yuhuali@usc.edu (Y.L.); yancongdeng@gmail.com (Y.D.)
*Corresponding author

***Abstract*—The advancement of 5G technology and the Internet of Things (IoT) has enabled the development of applications such as augmented reality and autonomous vehicles, which demand low latency and high computational power. These applications generate computationally intensive and delay-sensitive workloads, necessitating an efficient edge-cloud collaborative framework for optimal resource management. This study introduces a task offloading and service caching framework designed to enhance computational efficiency by dynamically distributing tasks between local processing and cloud-based execution at the edge, thereby reducing system overhead in terms of latency and energy consumption. To achieve this, a non-cooperative game theory-based distributed task offloading mechanism is employed to optimize offloading decisions. Additionally, a dynamic service caching strategy, formulated as a 0–1 knapsack optimization problem, is implemented to store frequently requested services at edge servers, improving task execution efficiency. As a result, task offloading decisions dynamically adapt to service caching updates, ensuring an optimal balance between performance, resource utilization, and system efficiency.**

***Keywords*—game theory, service caching, task offloading**

I. Introduction

The rapid development of high-bandwidth, low-latency communication technologies such as Fifth Generation (5G) networks and the Internet of Things (IoT) has accelerated the adoption of computationally intensive applications like augmented reality and autonomous driving. These emerging applications demand both substantial processing power and stringent latency requirements, posing significant challenges for traditional centralized cloud computing frameworks. In particular, the transfer of large volumes of data to distant cloud servers can lead to increased latency and higher energy consumption, thereby limiting real-time responsiveness.

To address these challenges, edge-cloud collaborative computing has gained attention as a viable solution [1]. By distributing computing resources closer to end users, edge servers can process a portion of tasks locally, thereby reducing communication delays. At the same time, more complex or resource-intensive tasks can be offloaded to cloud servers with greater processing capacity. This hierarchical offloading approach helps balance latency and energy consumption—collectively referred to in this work as system overhead.

In this paper, we propose an integrated architecture that optimizes computation through a combination of task offloading and service caching. We begin by introducing a distributed task offloading method derived from non-cooperative game theory, enabling tasks to autonomously decide whether to process data locally or offload it to an edge server. This decision-making process aims to minimize system overhead by considering the trade-off between latency and energy consumption.

However, because edge servers have limited storage capacity, storing frequently accessed services at the edge becomes crucial for maintaining offloading efficiency. To address this, we employ a 0–1 knapsack algorithm that dynamically caches popular services at edge servers, thereby increasing the likelihood that an offloaded task will find the required service locally. If a requested service is not cached at the selected edge server, the task may either revert to local processing or offload to the cloud, depending on which option offers lower overhead.

By combining a distributed offloading strategy with a dynamic caching mechanism, our approach enhances resource utilization and reduces overall latency and energy consumption. In the following sections, we detail the system model, formulate the problem of task offloading under edge-cloud collaboration, and present our proposed caching strategy. We then demonstrate the effectiveness of our framework through experimental evaluations and conclude by discussing potential future directions.

II. Related Works

Several studies have explored strategies for optimizing task offloading in cloud-assisted edge computing environments. For instance, Wu *et al.* [2] proposes an optimal task offloading framework aimed at minimizing local energy consumption, while Yi *et al.* [3] focuses on maximizing network management profit through refined offloading strategies. Additionally, Ma *et al.* [4] addresses the delay-sensitive requirements of mobile users by designing an offloading scheme that maximizes cumulative network utility and throughput.

However, the approaches in [2–4] generally overlook the critical issue of service caching at fog servers, instead assuming that these servers possess unlimited storage capacity

Manuscript received January 10, 2025; revised February 5, 2025; accepted March 4, 2025; published June 5, 2025.

to support an infinite range of computational services. In contrast, Bi *et al.* [5] through [6, 7] investigate both task offloading and service caching at fog servers. Yet, their analyses are restricted to environments that consist solely of fog servers. In such scenarios, if a fog server lacks the required computational service, the task must be processed locally, often leading to significant delays due to limited local processing power. In integrated fog-cloud collaborative systems, tasks that cannot be serviced by fog servers are instead offloaded to the cloud, which offers greater storage capacity and a broader range of computational services.

Furthermore, recent work has begun to integrate security considerations with offloading and caching strategies. For example, Hanumantharaju *et al.* [8] presents a fog-driven approach for a distributed intrusion detection system in blockchain-cloud environments, leveraging the decentralization of blockchain to enhance real-time data auditing. Similarly, Gupta *et al.* [9] introduces a secure virtual machine live migration technique in cloud computing that employs Blowfish encryption alongside blockchain technology. Although both studies contribute valuable security mechanisms, they do not address the simultaneous challenges of service caching and task offloading within a fog-cloud collaborative framework.

While game-theoretic models provide a powerful framework for decision-making in edge-cloud collaborative systems, their computational complexity in large-scale implementations presents a significant challenge. Non-cooperative game-theoretic approaches, in particular, require solving equilibrium conditions, which can become intractable as the number of players and strategy spaces grow. Daskalakis *et al.* [10] demonstrated that computing a Nash equilibrium in even a bimatrix game is PPAD-complete, implying that exact solutions may be infeasible for large-scale, real-time systems. This computational burden limits the direct application of traditional equilibrium-based strategies in dynamic, resource-constrained environments such as fog-cloud architectures. To address this challenge, our approach leverages efficient approximation techniques and learning-based heuristics to reduce computational overhead while maintaining decision-making accuracy. By integrating decentralized optimization strategies and distributed resource allocation methods, we ensure scalability and feasibility, making our proposed model more suitable for practical implementations.

III. Methodologies

The rapid advancement of emerging technologies such as 5G and the Internet of Things (IoT) has significantly accelerated the development of applications like Augmented Reality (AR) and autonomous driving. These applications are characterized by strict low-latency requirements and high computational demands, making them both energy-intensive and delay-sensitive. Ensuring optimal performance for such applications presents a major challenge, particularly in resource-constrained environments.

To address this, we propose a collaborative edge-cloud computing architecture that integrates task offloading and service caching to enhance Quality of Service (QoS). Our approach employs a distributed task offloading mechanism based on non-cooperative game theory, enabling AIoT devices to make autonomous offloading decisions. This method evaluates whether computational tasks should be processed locally or offloaded to edge servers, with the primary goal of reducing system overhead and minimizing latency. Given the limited storage capacity of edge (fog) servers, efficient service caching is essential to maintain performance. To optimize resource utilization, we implement a 0–1 knapsack algorithm, which dynamically selects services for caching based on their popularity and request frequency. This ensures that frequently requested services are stored at the edge, improving offloading efficiency and reducing cloud dependency. Furthermore, offloading decisions are dynamically adjusted based on real-time service caching outcomes. For example, if an initial offloading decision assigns a task to an edge server but the required service is not cached, the system will adapt by either processing the task locally or offloading it to the cloud. This adaptive mechanism enhances reliability, reduces latency, and optimizes computational resource distribution across the edge-cloud infrastructure, making it particularly suitable for latency-sensitive AIoT applications.

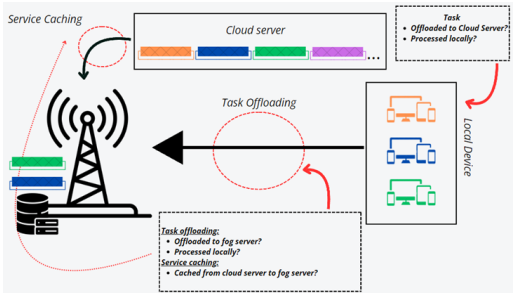

Fig. 1. System model.

A. System Models and Problem Formulation

The Task Offloading and Service Caching for Cloud-Assisted Fog Computing (TO&SC-CF) model, as illustrated in Fig. 1, consists of a cloud server, a fog server, and multiple Augmented Intelligence of Things (AIoT) devices. Each AIoT device is assigned a specific task, collectively represented as the job set $N = \{1, \ldots, N\}$. Within Emergency Management Systems (EMS), AIoT devices typically offload tasks to the fog server to take advantage of low transmission latency and reduced computational delays. During this process, tasks can either be executed locally or offloaded to the fog server for processing.

The fog server, equipped with computing capabilities, facilitates task execution but operates with limited storage capacity, restricting the number of computing services it can cache. In contrast, cloud computing provides a vast range of computational resources. To optimize resource allocation, dynamic service caching is implemented in the fog server, prioritizing frequently requested services to enhance task processing efficiency. However, despite these caching optimizations, certain high-demand computing services may still be unavailable at the fog layer. In such cases, AIoT devices must reevaluate their offloading strategies, deciding whether to execute tasks locally or offload them to the cloud server to ensure efficient processing. To enhance clarity, the key notations used in this model are summarized in Table I.

TABLE I
SUMMARY OF KEY NOTATIONS

Notation	Definition
$\mathcal{N}$	Collection of computational tasks in the system
$\mathcal{K}$	Set of available computational services
$\mathbf{x}_n$	Decision variable indicating whether task n^{th} is processed locally or offloaded
T_n^l	Time required for local execution of task n^{th}
c_n	Computational complexity of task n, measured in required CPU cycles
f_n	Available computational power for local execution of task n
E_n^l	Time delay incurred during task n's transmission for fog-based processing
$T_{trans}^{f,n}$	Transmission delay for fog processing
b_n	Amount of data associated with task n
r_n^f	Transmission rate between task n and the fog computing server
$E_{trans}^{f,n}$	Energy consumed during the transmission of task n to the fog server
p_n^f	Power consumption of task n during data transmission
f_n^f	Computational resources allocated to task n in the fog server
r_n^c	Transmission rate between task n and the cloud server
α_n^f	Binary variable indicating whether the fog server caches the required service for task n

B. Task Offloading Models

The decision to offload job n is represented by the vector $X_n = \{x_n^l, x_n^f, x_n^c\}$ where each component is a binary variable indicating the offloading decision.

1) Local Processing: Task n is processed locally when $x_n^l = 1$ and $x_n^f = x_n^c = 0$ The local computation delay is given by:

$$T_n^l = \frac{c_n}{f_n} \tag{1}$$

wheref_n represents the local CPU cycle frequency, and denotes the number of CPU cycles required for job n. Local processing also consumes energy, calculated as:

$$E_n^l = \varpi c_n f_n^2 \tag{2}$$

where ϖ is the energy coefficient associated with the AIoT device's chip design.

2) Fog Processing: Task n is offloaded to the fog server for processing if $x_n^f = 1$ and $x_n^l = x_n^c = 0$ [11]. The job offloading process involves three stages: 1) data transfer, 2) task completion, and 3) outcome feedback. The transmission delay, when task n is sent to the fog server, is:

$$T_{trans}^{f,n} = \frac{b_n}{r_n^f} \tag{3}$$

where r_n^f is the transmission rate from the AIoT device to the fog server for task n and b_n represents the data bits of task n [12]. The transmission energy usage is determined using the AIoT's transmission power p_n^f [13, 14]:

$$E_{trans}^{f,n} = p_n^f T_{trans}^{f,n} \tag{4}$$

Upon receiving the offloaded tasks, the fog server processes them using the assigned fog computing resources f_n^f, and the computation latency for fog processing is computed as:

$$T_{comp}^{f,n} = \frac{c_n}{f_n^f} \tag{5}$$

It is important to highlight that this study primarily examines local energy consumption rather than the energy usage of the fog server [15]. Furthermore, once tasks are processed at the fog layer, the results must be transmitted back to the AIoT devices to finalize the offloading process. Since the feedback transmission involves a significantly smaller data volume compared to the initial offloaded tasks, the associated latency and energy consumption are considered negligible [16].

3) Cloud Processing: Task n is offloaded to the cloud server for processing if $x_n^c = 1$ and $x_n^l = x_n^f = 0$ [17]. Given the superior computational capabilities of cloud servers, cloud processing latency is typically negligible. The transmission latency to the cloud is:

$$T_{trans}^{c,n} = \frac{b_n}{r_n^c} \tag{6}$$

wherer_n^f is the transmission rate between the AIoT device and the cloud server. The energy usage for this transmission is:

$$E_{trans}^{c,n} = p_n^c T_{trans}^{c,n} \tag{7}$$

C. Service Caching Model

To facilitate task processing in Emergency Management Systems (EMS), a specialized collection of computing services, along with associated databases and libraries, is stored on the fog server. Due to its limited storage capacity, the fog server can cache only a restricted set of computing services. However, as AIoT devices undertake diverse activities, the demand for specific computational services fluctuates. Employing a fixed caching strategy ensures that certain services are always available, but this approach may limit fog offloading options for particular tasks, especially if the necessary computing services are not initially cached. This limitation results in decreased offloading efficiency, particularly when AIoT devices frequently request services that are not available. To address this issue, our study develops a dynamic service caching technique, inspired by [18], that allows the fog server to dynamically request computational services from the cloud based on varying task requirements, thereby enhancing task offloading efficiency. As showed in Fig. 2, it illustrates the dynamic service caching principle via a flowchart.

(a) Requested Service Information: AIoT devices first communicate their required service details to the fog server before initiating task transmission. This enables the fog server to anticipate potential offloading needs.

(b) Determining Service Popularity: Upon receiving service requests, the fog server evaluates the popularity of each service by analyzing request frequency. Services that are frequently requested gain higher priority for caching.

(c) Service Caching: Due to limited storage, the fog server selectively caches essential services by retrieving them from

the cloud. A binary indicator ($\alpha_n^f = 0/1$)determines whether a requested service for a given task is available in the fog server cache. Cached services enable direct task execution at the fog server.

(d) Offloading Adjustments: If a required service is not available in the fog cache, AIoT devices must reconsider their offloading strategy—either executing tasks locally or offloading them to the cloud. Meanwhile, service downloads from the cloud to the fog server occur concurrently with task transmissions from AIoT devices. Given the high-speed connection between fog and cloud servers, the download delay is mainly attributed to transmission time, ensuring that the dynamic caching mechanism does not introduce additional latency.

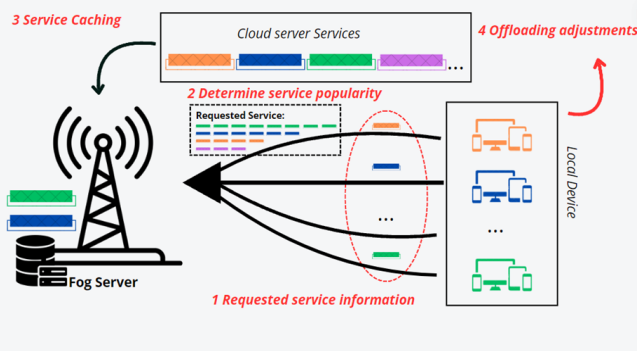

Fig. 2. Flowchart of execution dynamic service caching.

D. Problem Formulation

Given the energy and time sensitivity of tasks requiring offloading, our objective is to minimize the system cost, defined as the weighted sum of local energy consumption and task delay. Each AIoT device evaluates offloading decisions based on available power, data transfer rates, and assigned CPU cycles, determining whether to complete its task locally or offload it to a fog server. The limited storage capacity of the fog server restricts its ability to cache all required computing services. Consequently, tasks whose services are not cached must reassess their offloading decisions, choosing either local completion or cloud offloading to minimize system costs. When integrating service caching with offloading decisions, we derive the delay for task n, expressed as:

$$T_n = x_n^l T_n^l + \alpha_n^f x_n^f (T_{trans}^{f,n} + T_{comp}^{f,n}) + x_n^c T_{trans}^{c,n} \quad (8)$$

The local energy consumption is represented by:

$$E_n = x_n^l E_n^l + \alpha_n^f x_n^f E_{trans}^{f,n} + x_n^c E_{trans}^{c,n} \quad (9)$$

Mathematically, the problem is stated as:

$$min_{x_n} \sum_{n \in N} \beta T_n + (1-\beta) E_n \quad (10)$$

With constraints:

$$\alpha_n^f \in \{0,1\} \quad (11a)$$

$$s_n^f \leq s^{max} \quad (11b)$$

$$x_n^c, x_n^l, x_n^f \in \{0,1\} \quad (11c)$$

$$f_n \leq f_n^{max}, f_n^f \leq f^{max} \quad (11d)$$

Constraint 11(a) indicates that a service is either cached or not stored in the fog server. Constraints 11(b), 11(c), and 11(d) ensure that the maximum computing resources of the AIoT device and fog server, as well as the storage space for cached services, are not exceeded, and that each task is offloaded to only one processor. Finding the optimal solution to this problem is challenging. Firstly, the offloading and service caching decisions form a nonconvex problem due to their discrete nature. Furthermore, the integration of service caching with task offloading complicates the direct resolution of the issue. Secondly, the typical absence of centralized controllers in real-world scenarios makes it difficult for AIoT devices to access global information.

E. Task Offloading and Service Caching for Cloud-Assisted Fog Computing

In this section, we introduce the Task Offloading and Service Caching for Cloud-Assisted Fog Computing (TO&SC-CF) framework to effectively address the identified challenges. The framework is structured around two key components: task offloading and service caching. To minimize system costs, we first employ a non-cooperative game-theoretic approach to determine optimal task offloading decisions. Next, we leverage the 0–1 knapsack algorithm to maximize the popularity of cached services, ensuring efficient service caching. Finally, the task offloading strategy is further refined by incorporating the caching decisions, leading to an adaptive and optimized offloading mechanism that enhances overall system performance.

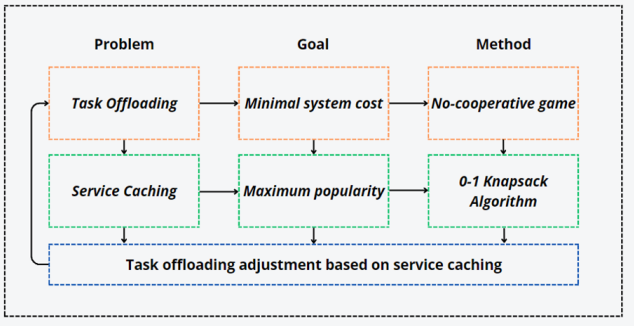

Fig. 3. Workflow of TO&SC-CF algorithm.

1) Noncooperative Game for Task Offloading: In Emergency Management Systems (EMS), AIoT devices often offload tasks to fog servers to take advantage of low transmission latency and reduced computational delay. During the offloading process, these devices can either execute tasks locally or transfer them to the fog server for processing. Given the distributed nature of AIoT devices, we design a non-cooperative game model to optimize task offloading decisions. Each device independently selects its offloading strategy without relying on a centralized controller, ensuring individual flexibility while enhancing overall system efficiency [19, 20]. The game is formally defined as follows:

$$G = (\mathcal{N}, \mathcal{A}, \mathcal{N}(\mathcal{A})) \quad (12)$$

where $\mathcal{A}$ represents the complete set of feasible task offloading choices, $\mathcal{N}$ denotes the finite set of tasks, and $\mathcal{U}(\mathcal{A})$ measures the utility derived from the offloading

decisions in set$\mathcal{A}$. The utility function for any allocation method $\mathcal{A}_i$ is defined by:

$$U(\mathcal{A}_i) = F - f(\mathcal{A}_i), i \in \mathcal{A} \tag{13}$$

where F is a sufficiently large number to ensure positive utility values, and as$f(\mathcal{A}_i)$ focuses on the optimized outcomes, stated as:

$$f(\mathcal{A}_i) = \beta T(\mathcal{A}_i) + (1-\beta)E(\mathcal{A}_i) \tag{14}$$

Here,$T(\mathcal{A}_i)$and $E(\mathcal{A}_i)$ represent the overall task delay and local energy consumption, respectively, executed by policy$\mathcal{A}_i$. We aim to maximize the utility$U(\mathcal{A}_*)$ to determine the optimal offloading strategy $\mathcal{A}_*$, thereby minimizing system costs, utilizing the "regret-matching" approach within the noncooperative game framework. The regret for preferring strategy$\mathcal{A}_j$ over$\mathcal{A}_i$ is computed as:

$$R(\mathcal{A}_i, \mathcal{A}_j) = max\{U(\mathcal{A}_i, \mathcal{A}_{-i}) - U(\mathcal{A}_j, \mathcal{A}_{-j}), 0\} \tag{15}$$

It defines the regret function used in the noncooperative game framework to evaluate task offloading strategies. The regret function, $R(\mathcal{A}_i, \mathcal{A}_j)$, measures the difference between the utility of an alternative strategy $\mathcal{A}_i$ and the currently chosen strategy $\mathcal{A}_j$ It is computed as the maximum between this utility difference and zero, ensuring that regret values remain non-negative. If the utility of $\mathcal{A}_i$ is higher than $\mathcal{A}_j$, the regret value quantifies the missed improvement in system performance had the alternative strategy been chosen. This formulation allows the system to continuously evaluate offloading decisions and provides the basis for adjusting strategies dynamically to optimize overall task execution in fog-cloud environments.

$$P(\mathcal{A}_i) = \frac{1}{\tau} R(\mathcal{A}_i, \mathcal{A}_j), i \neq j \tag{16}$$

Building on the regret function in Eq. (15), Eq. (16) determines the probability of selecting a different strategy based on its computed regret. The probability of choosing strategy $\mathcal{A}_i$ denoted as $P(\mathcal{A}_i)$, is directly proportional to its regret value $R(\mathcal{A}_i, \mathcal{A}_j)$ and is scaled by a factor τ, which controls the adaptation rate of the system. A higher regret value indicates that the alternative strategy $\mathcal{A}_i$ would have resulted in better performance, thereby increasing the likelihood of selecting it in future iterations. This probabilistic update mechanism ensures that task offloading decisions are adjusted adaptively, allowing the system to gradually shift toward optimal strategies while still maintaining flexibility to explore other potential solutions. Over time, strategies with consistently lower regret values become dominant, leading to an efficient and stable task offloading process.

Algorithm 1 summarizes the proposed task offloading method. After I iterations, the time complexity of Algorithm 1 is $\mathcal{O}(AI)$, where $A = |\mathcal{A}|$ signifies the total number of viable offloading strategies. For fog computing resources $\rho_\xi(\bullet)$, denotes the empirical distribution of all potential allocation strategies$\mathcal{A}$. The frequency of strategy $\mathcal{A}_i$ up to ξ iterations is represented by $S_\xi(\mathcal{A}_i)$. Thus, we assess how frequently strategy $\mathcal{A}_i$ was employed over time:

$$\rho_\xi(\mathcal{A}_i) = \frac{S_\xi(\mathcal{A}_i)}{\xi} \tag{17}$$

We postulate that a ξ approaches infinity,$\rho_\xi(\mathcal{A}_i)$ converges to a correlated equilibrium, as evidenced by the analyses in [19, 21].

Algorithm 1 Noncooperative game based offloading algorithm

Input: $b_n, c_n, f_n, r_n^c, p_n^f, p_n^c$
Output: The optimal task offloading strategy $\mathcal{A}_*$

Evaluate the utility value of each available offloading strategy, based on Equation (14)
Iterate through the previous step until the utility values for all possible strategies are obtained
Randomly select an initial offloading strategy $\mathcal{A}_i$
while $\mathcal{A}_i \in \mathcal{A}$ **do**
    Compute the regret degree associated with strategy $\mathcal{A}_i$ following Equation (15)
    if $P(\mathcal{A}_i) > 0$ **then**
        Include $\mathcal{A}_i$ in the set of candidate strategies
    end if
    Update the current strategy by selecting a new strategy from the set of candidate strategies
    Retrain the optimal strategy $\mathcal{A}_*$ by maximizing its utility function $U(\mathcal{A}_*)$ based on the accumulated knowledge.
end while

2) 0–1 Knapsack-based Service Caching: To avoid infeasible task offloading decisions, service caching plays a crucial role [17, 22]. Given the limited storage capacity of fog servers in Emergency Management Systems (EMS), they can only accommodate a restricted number of specialized services. The absence of frequently requested services at the fog layer can significantly degrade processing efficiency. To mitigate this issue, we introduce a dynamic service caching strategy that optimally selects which services should be cached to enhance task offloading efficiency.

Our approach enables the fog server to flexibly request computing services from the cloud, denoted as $k \in \mathcal{K}$, based on varying workload demands. We define a binary indicator $\lambda_n^k \in \{0, 1\}$ to represent whether task$n$ requires service$k$, where $\lambda_n^k = 1$ indicates a request, and$\lambda_n^k = 0$ means the service is unavailable at the fog server. The popularity of a service$k$ is determined based on its request frequency and is used as a criterion for caching decisions.

$$L_k = \sum_{n \in \mathcal{N}} \lambda_n^k \tag{18}$$

Each service requires s_k units of storage space, and the total allocated cache storage must not exceed the maximum storage capacity of the fog server, s_{fog}^{max}.The primary goal of the dynamic service caching mechanism is to prioritize the caching of frequently requested services while staying within storage constraints. This problem is formulated as a 0–1 knapsack optimization, where services that demand excessive storage will not be cached, whereas highly popular services that fit within the available storage will be retained. To implement this efficiently, we introduce a 0–1 knapsack-based dynamic caching strategy, detailed in Algorithm 2. The computational complexity of this algorithm is expressed as:

$$\mathcal{O}((S+1)(K+1)) \tag{19}$$

where $S = \left| s_{fog}^{max} \right|$ represents the maximum available storage in the fog server. and $K = |\mathcal{K}|$ denotes total number of services that need to be evaluated for caching. This

complexity suggests that the algorithm scales linearly with both storage capacity and the number of service requests, making it computationally efficient for real-time workload adaptation. By dynamically adjusting caching decisions based on service popularity and storage constraints, this approach ensures optimal resource utilization and reduces latency in fog computing environments.

Algorithm 2 0–1 Knapsack-based service caching algorithm

Input: s_k, L_k, s_{fog}^{max}
Output: The optimal fog service caching policies, $b_n^k, k \in \mathcal{K}, n \in \mathcal{K}$

Determine the available storage space s_r in the fog server
for k^{th} service **do**
  if the storage requirement s_k exceeds the available space s_r **then**
    k^{th} service will be cached.
  else if service k prodeces a larger popularity **then**
    k^{th} service will not be cached
  else
    k^{th} service will be cached
    Update the remaining fog storage capacity accordingly.
  end if
end for

IV. Performance Evaluation

A. Setup

In our experimental setup, we evaluate cloud-assisted fog computing using five AIoT devices, where each device generates a computational task that requires processing. Table II outlines the parameter settings used in our trials. We compare the performance of our proposed algorithm against several benchmark algorithms as follows:

TABLE II
Parameter Settings

Paramater Description	Value
Cloud server quantity	1
Fog server quantity	1
AIoT devices quantity	5
Services quantity	3
Data bit for each task	[0.1,2] MB
Required CPU cycles for each task	[0.2,4] GHZ
Fog server computational capabilities	[5,15] GHZ
One AIoT device computational capabilities	[1,2] GHZ
Data transmission rate to the fog server	[2,3] Mb/s
Transmission power	0.1 Watt
Energy coefficient ϖ	10^{-26}

B. Experiments

(a) Task Offloading with Adaptive Service Caching (TO&SC) [18]: This approach enables task offloading within a fog computing environment, where service caching is dynamically adjusted based on demand. Unlike cloud-assisted strategies, it operates exclusively within the fog layer without relying on cloud resources for caching decisions.

(b) Cloud-Integrated Fog Computing Task Offloading (TO&CF) [23]: This method follows a predefined service caching policy, meaning that the set of cached services remains unchanged. When the required service is unavailable at the fog layer, tasks are redirected to the cloud server, leveraging its resources to reduce processing latency and energy consumption.

(c) Fog-Based Task Offloading (TO) [24]: This strategy utilizes a static caching framework where services are fixed within the fog server. If a requested service is not locally available, tasks must be executed directly on the AIoT devices, as there is no option to offload them to the cloud.

(d) Standalone Local Processing (LP): In this method, all tasks are handled entirely on local devices, without any offloading capabilities, making it dependent on the device's own computational resources.

C. Comparison Analysis on Varied Required CPU Cycles

Fig. 4 presents a comparison of system costs across different task CPU cycle requirements for various offloading algorithms. As CPU cycles increase, system costs generally rise due to higher computational energy consumption and longer task execution times. The cost trends for LP, TO, and TO&SC exhibit a similar growth pattern, which can be attributed to the fact that in this experimental setup, tasks requiring higher CPU cycles do not necessitate caching in the fog server. Instead, both TO and TO&SC handle these computationally demanding tasks locally, leading to a steady increase in system costs.

In contrast, the proposed algorithm and TO&CF enable offloading of resource-intensive tasks to the cloud server, utilizing cloud-assisted processing capabilities. This approach maintains a more stable system cost across varying CPU cycles, as the primary factor influencing cost becomes cloud transmission energy rather than local computational complexity. Furthermore, in scenarios where fog storage capacity is constrained, the proposed method consistently achieves the lowest system cost, as further illustrated in Fig. 5.

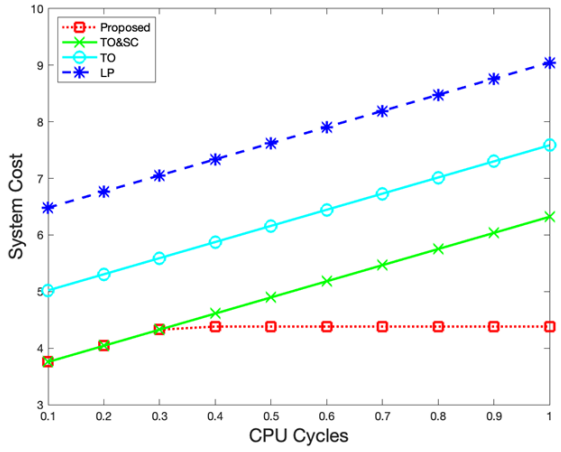

Fig. 4. Performance of different algorithms under various CPU requirements.

D. Comparison Analysis on varied Device Numbers

As illustrated in Fig. 6, the system cost increases as the number of devices grows, primarily due to higher task execution delays and greater energy consumption. The LP approach, which does not support fog-based processing,

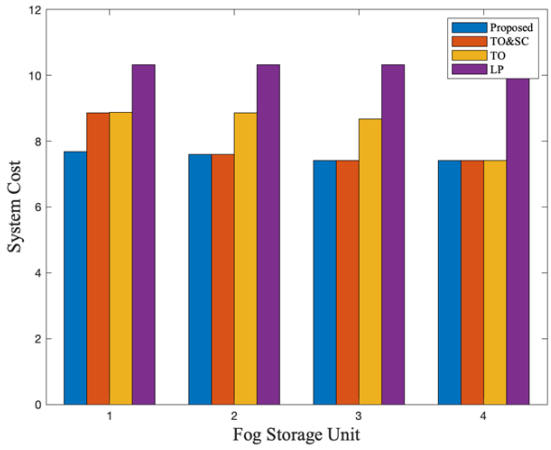

Fig. 5. Performance of different algorithms under different fog storage units.

experiences the steepest rise in system cost, as tasks must be handled entirely on local devices, leading to increased computational overhead. The TO strategy also incurs relatively high system costs since it relies on a fixed caching scheme and lacks cloud offloading capabilities, resulting in inefficient task execution when required services are unavailable.

In contrast, TO&SC achieves better performance by utilizing dynamic caching, which improves resource allocation and reduces processing delays. However, the proposed algorithm consistently outperforms all other methods, as it leverages both dynamic service caching and cloud-assisted computing, ensuring optimal task distribution while minimizing overall system cost. The results demonstrate that integrating intelligent caching mechanisms and cloud resources significantly e nhances e fficiency, pa rticularly as the number of AIoT devices increases.

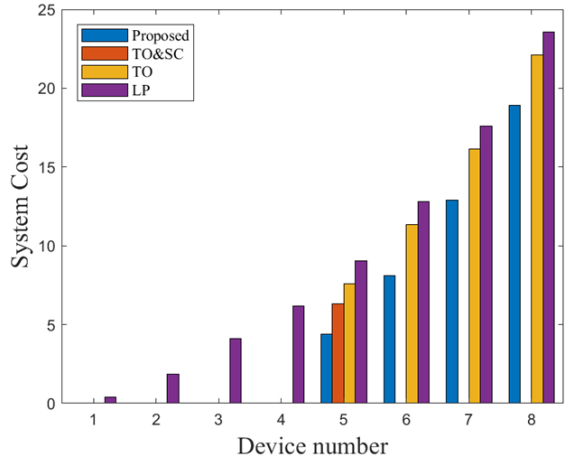

Fig. 6. Performance of different algorithms under different device numbers.

E. Discussion and Future Works

1) Impact of fog storage capacity: The dynamic service caching algorithm, selects frequently requested services to be stored at the fog using a knapsack-based method. Our results indicate that when the fog storage unit is limited, the caching mechanism is constrained, leading to more tasks being rerouted either to local processing or to the cloud. This rerouting incurs higher latency and energy costs, which is reflected in an increased system cost.

As the fog storage capacity increases, a larger number of popular services can be cached. This improvement reduces the probability that a task, originally offloaded to the fog, finds its required service unavailable. Consequently, more tasks are processed efficiently at the fog level, and the overall system cost decreases. However, our experiments also show that beyond a certain storage threshold, further increases yield diminishing returns, suggesting the existence of an optimal fog storage capacity for cost minimization.

2) Algorithmic adjustments and convergence: In parallel, our task offloading scheme employs a regret learning mechanism to iteratively adjust offloading decisions. This mechanism evaluates the utility of different offloading strategies—whether to process tasks locally, at the fog, or in the cloud—based on current system parameters and cached service availability. The iterative adjustment gradually converges to a near-optimal strategy that minimizes system cost. Notably, as the number of iterations increases, the regret values decrease, and the offloading decision stabilizes, reinforcing the robustness of our approach.

3) Comparative performance: When comparing the baseline offloading approach with our integrated dynamic caching and offloading adjustment, our simulation results demonstrate a clear performance improvement. Specifically, the integrated method consistently achieves a lower system cost across various fog storage unit settings. This improvement is attributed to two key factors:

Dynamic Service Caching:By proactively caching high-demand services, the system reduces the need for expensive offloading to the cloud, thus lowering delay and energy consumption.

Adaptive Offloading Decisions: The regret-based learning mechanism allows the system to adapt to changing conditions, ensuring that tasks are routed to the processing node with the minimal cost, even when service availability fluctuates.

4) Future works: One promising future direction is the integration of Deep Reinforcement Learning (DRL) to enhance the adaptability of task offloading and service caching in fog-cloud collaborative systems. Traditional methods rely on static policies or heuristic-based approaches, which may not effectively respond to dynamic network conditions, fluctuating workloads, and energy constraints. By leveraging DRL algorithms such as Deep Q-Networks (DQN) or Proximal Policy Optimization (PPO), the system can learn optimal offloading and caching strategies by continuously interacting with the environment and adapting to real-time changes. This approach enables autonomous decision-making, allowing AIoT devices to intelligently distribute computational tasks between fog and cloud layers while optimizing latency, energy consumption, and resource utilization. For instance, Wang *et al*. [25] proposed a DRL-based framework for dynamic task offloading in mobile edge computing, demonstrating its effectiveness in reducing offloading delays and system costs. By extending this concept to fog-cloud systems with service caching, future research can further improve scalability, responsiveness, and overall system performance, making it highly suitable for smart city applications, emergency response, and industrial automation.

V. Conclusion

In this paper, we explored an Emergency Management Systems (EMS) scenario to enhance cloud-assisted fog computing capabilities. Given the constraints of AIoT devices, which are both delay-sensitive and energy-intensive, our primary objective was to devise a solution that optimally balances these parameters and determines the most suitable execution locale for tasks—either locally or transferred to the cloud—to minimize system costs. In scenarios where the fog server is either overwhelmed or unavailable, we considered the use of cloud computing as an alternative. Our initial strategy involved developing a distributed offloading method based on non-cooperative game theory, focusing on local processing at AIoT devices or offloading to the fog server, addressing challenges related to task offloading and dynamic service caching. Subsequently, we applied the 0-1 knapsack problem-solving approach to implement adaptive service caching based on job popularity. This method ensures that tasks are either processed within the fog server, offloaded to the cloud, or managed locally as required. Extensive testing of the proposed algorithm demonstrated its effectiveness compared to existing methodologies, validating its potential for practical deployment in similar scenarios.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

This paper was completed through the collaborative efforts of all authors. Yancong Deng carried out the experiment and supervised the project. Yuhua Li verified the analytical methods, validated the data and wrote the manuscript. All authors reviewed, revised, and approved the final version.

Acknowledgment

We express our gratitude to all those who assisted in the writing of this report. Special thanks go to our supervisor, Dr. Yancong Deng, for his constant encouragement, guidance, and support in providing the necessary knowledge and designing the code implementation experiments. Additionally, we thank the online TA for their detailed Q&A sessions following each paper review.

References

[1] X. Dai *et al*., "Task Offloading for Cloud-Assisted Fog Computing With Dynamic Service Caching in Enterprise Management Systems," in IEEE Transactions on Industrial Informatics, vol. 19, no. 1, pp. 662-672, Jan. 2023, doi: 10.1109/TII.2022.3186641.

[2] Y. Wu, B. Shi, L. Qian, F. Hou, J. Cai, and X. Shen, "Energy-Efficient Multi-task Multi-access Computation Offloading Via NOMA Transmission for IoTs," vol. 16, no. 7, pp. 4811–4822, Jul. 2020, doi: https://doi.org/10.1109/tii.2019.2944839.

[3] C. Yi, S. Huang, and J. Cai, "Joint Resource Allocation for Device-to-Device Communication Assisted Fog Computing," IEEE Transactions on Mobile Computing, vol. 20, no. 3, pp. 1076–1091, Mar. 2021, doi: https://doi.org/10.1109/tmc.2019.2952354.

[4] Y. Ma, W. Liang, J. Li, X. Jia, and S. Guo, "Mobility-Aware and Delay-Sensitive Service Provisioning in Mobile Edge-Cloud Networks," IEEE Transactions on Mobile Computing, pp. 1–1, 2020, doi: https://doi.org/10.1109/tmc.2020.3006507.

[5] S. Bi, L. Huang, and Y.-J. A. Zhang, "Joint Optimization of Service Caching Placement and Computation Offloading in Mobile Edge Computing Systems," IEEE Transactions on Wireless Communications, pp. 1–1, 2020, doi: https://doi.org/10.1109/twc.2020.2988386.

[6] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading Dependent Tasks in Mobile Edge Computing with Service Caching," IEEE INFOCOM 2020 - IEEE Conference on Computer Communications, Toronto, ON, Canada, 2020, pp. 1997-2006, doi: 10.1109/INFOCOM41043.2020.9155396.

[7] S.-W. Ko, S. J. Kim, H. Jung, and S. W. Choi, "Computation Offloading and Service Caching for Mobile Edge Computing under Personalized Service Preference," IEEE Transactions on Wireless Communications, pp. 1–1, 2022, doi: https://doi.org/10.1109/twc.2022.3151131.

[8] R. Hanumantharaju, K. N. Shreenath, B. J. Sowmya, and K. G. Srinivasa, "Fog-Driven Approach for Distributed Intrusion Detection System in Auditing the Data Based on Blockchain-Cloud Systems," Cloud Computing and Data Science, vol. 5, no. 1, pp. 97–107, Nov. 2023.

[9] A. Gupta, S. Namasudra, and P. Kumar, "A secure VM live migration technique in a cloud computing environment using blowfish and blockchain technology," The Journal of Supercomputing, pp. 1–24, Aug. 2024.

[10] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou, "The complexity of computing a Nash equilibrium," SIAM Journal on Computing, vol. 39, no. 1, pp. 195–259, 2009, doi: 10.1137/070699652.

[11] X. Dai *et al*., "Task co-offloading for D2D-assisted mobile edge computing in industrial Internet of Things", IEEE Trans. Ind. Inform..

[12] H. Jiang, Z. Xiao, Z. Li, J. Xu, F. Zeng, and D. Wang, "An energy-efficient framework for Internet of Things underlaying heterogeneous small cell networks", IEEE Trans. Mobile Comput., vol. 21, no. 1, pp. 31-43, Jan. 2022.

[13] F. Zeng, Q. Li, Z. Xiao, V. Havyarimana, and J. Bai, "A price-based optimization strategy of power control and resource allocation in full-duplex heterogeneous macrocell-femtocell networks", IEEE Access, vol. 6, pp. 42004-42013, 2018.

[14] Z. Xiao *et al*., "A joint information and energy cooperation framework for CR-Enabled macro-femto heterogeneous networks", IEEE Internet Things J., vol. 7, no. 4, pp. 2828-2839, Jul. 2020.

[15] H. Jiang, X. Dai, Z. Xiao, and A. K. Iyengar, "Joint task offloading and resource allocation for energy-constrained mobile edge computing", IEEE Trans. Mobile Comput.

[16] X. Hu, K.-K. Wong, K. Yang, and Z. Zheng, "UAV-assisted relaying and edge computing: Scheduling and trajectory optimization", IEEE Trans. Wireless Commun., vol. 18, no. 10, pp. 4738-4752, Oct. 2019.

[17] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading tasks with dependency and service caching in mobile edge computing", IEEE Trans. Parallel Distrib. Syst., vol. 32, no. 11, pp. 2777-2792, Nov. 2021.

[18] J. Xu, L. Chen and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks", Proc. IEEE Conf. Comput. Commun., pp. 207-215, 2018.

[19] S. Hart and A. Mas-Colell, "A simple adaptive procedure leading to correlated equilibrium," Econometrica, vol. 68, no. 5, pp. 1127–1150, 2010.

[20] Z. Xiao *et al*., "Spectrum resource sharing in heterogeneous vehicular networks: A noncooperative game-theoretic approach with correlated equilibrium", IEEE Trans. Veh. Technol., vol. 67, no. 10, pp. 9449-9458, Oct. 2018.

[21] Z. Xiao *et al*., "Vehicular task offloading via heat-aware MEC cooperation using game-theoretic method," IEEE Internet Things J., vol. 7, no. 3, pp. 2038–2052, Mar. 2020.

[22] V. Farhadi *et al*., "Service placement and request scheduling for data-intensive applications in edge clouds", Proc. IEEE Conf. Comput. Commun., pp. 1279-1287, 2019.

[23] R. Jindal, N. Kumar, and H. Nirwan, "TFCT: A task offloading approach for fog computing and cloud computing", Proc. 10th Int. Conf. Cloud Comput. Data Sci. Eng., pp. 145-149, 2020.

[24] A. Hazra, M. Adhikari, T. Amgoth, and S. N. Srirama, "Stackelberg game for service deployment of IoT-enabled applications in 6G-Aware fog networks", IEEE Internet Things J., vol. 8, no. 7, pp. 5185-5193, Apr. 2021.

[25] Y. Wang *et al*. "Deep reinforcement learning for dynamic task offloading in mobile edge computing." IEEE Transactions on Network and Service Management, vol. 17, no. 4, pp. 2537-2551, 2020. DOI: 10.1109/TNSM.2020.3034794