A Conflict-Reduced Task-to-Host Matching Scheme for Task Offloading in Edge Computing-Enabled IoT Network

Wang Dayong [1,*], Kamalrulnizam Bin Abu Bakar [1], Babangida Isyaku [2], and Lei Liping [3]

[1] Department of Computer Science, Faculty of Computing, Universiti Teknologi Malaysia, Johor Bahru, Malaysia
[2] Department of Information Technology, Faculty of Computing and Information Technology, Sule Lamido University, Kafin Hausa, Nigeria
[3] School of Political Science and Public Administration, East China University of Political Science and Law, Shanghai, China
Email: wangdayong@graduate.utm.my (W.D.); knizam@utm.my (K.B.A.B.); bangis4u@gmail.com (B.I.); 2437@ecupl.edu.cn (L.L.)
*Corresponding author

***Abstract*—The continuous evolution of applications and the introduction of novel service models have significantly intensified the computational demands on Internet of Things (IoT) Mobile Terminal Devices (MTDs). In order to improve service capability, edge computing-enabled IoT shifts the computational burden from MTDs to edge computing hosts through task offloading. However, current task-to-host matching approaches exhibit significant limitations, particularly in their inability to anticipate tasks that are nearing the completion of data transmission and are about to arrive at the edge computing network. This shortcoming results in matching conflicts between tasks offloaded horizontally across sites and those directly offloaded from MTDs. To address this challenge, a Two-layer Asynchronous Conflict-reduced Task-to-host Matching (TACTM) scheme is proposed. The proposed scheme jointly considers the data upload progress of tasks and the dynamic load of edge computing hosts, aiming to reduce matching conflicts while optimizing task-to-host assignment. Simulation results demonstrate that the proposed TACTM scheme outperforms existing methods by achieving a 20.79% reduction in average conflict ratio, an 8.26% decrease in average processing time, a 4.30% drop in average task failure ratio, and a 4.05% improvement in average Quality of Experience (QoE).**

***Keywords*—edge computing, internet of things, task offloading, task-to-host matching**

I. Introduction

In recent years, the Internet of Things (IoT) has been undergoing rapid and sustained development, driven by the continuous emergence of new services and applications. This trend has resulted in significantly increased computational demands on Mobile Terminal Devices (MTDs), due to increasingly complex functionalities and service requirements [1, 2]. However, the computational capabilities of MTDs are inherently limited by cost and size constraints [3]. These limitations result in reduced service responsiveness. Furthermore, as most MTDs are battery-powered [4]. intensive computational tasks can rapidly drain their limited energy resources.

The advent of task offloading technology has introduced a promising solution for alleviating the computational load on MTDs. It enables MTDs to offload part or all of their tasks to resource-abundant service networks for remote processing [5]. Cloud computing platforms possess abundant computational resources suitable for executing tasks offloaded from MTDs. However, the distance between MTDs and cloud platforms is typically large [6, 7], resulting in increased data transmission latency [8, 9]. In contrast, MTDs are more inclined to offload tasks to edge computing networks, owing to the significantly reduced communication span between MTDs and edge nodes [10, 11].

Unfortunately, edge computing hosts have limited capacity relative to cloud platforms [12]. When a large number of offloaded tasks arrive, it becomes necessary to optimize the matching between tasks and execution hosts. This problem has been extensively studied, and a variety of optimization techniques have been widely introduced, including but not limited to: mathematical optimization algorithms [13, 14], heuristic algorithms [15, 16], game-theoretic approaches [17, 18], and AI-based optimization methods [19, 20]. However, existing approaches exhibit significant limitations. They lack the ability to estimate the arrival time of new tasks, relying instead on the current network state and outdated offloading decisions to perform task-to-host matching. As a result, matching conflicts are likely to arise between vertically offloaded tasks from MTDs and horizontally offloaded tasks originating from cross-site coordination.

Manuscript received May 29, 2025; revised June 17, 2025; accepted August 15 2025; published October 24, 2025.

To address this challenge, a Two-layer Asynchronous Conflict-reduced Task-to-host Matching (TACTM) scheme with task data transmission progress tracking is proposed to mitigate matching conflicts and improve task execution efficiency in edge computing networks. Specifically, our main contributions are listed as follows:

- Conducted a thorough analysis of the suboptimal task-to-host matching observed in edge computing networks and attributes the issue to the absence of effective mechanisms for avoiding matching conflicts.
- Proposed a two-layer asynchronous conflict-reduced task-to-host matching scheme, which jointly considers task data transmission progress, edge host computational load, and available resources to minimize matching conflicts.
- Established a simulation environment and conducted extensive comparative experiments. The simulation results demonstrate that the proposed TACTM scheme achieves more effective task-to-host matching in edge computing networks. Improvements were observed in conflict ratio, processing time, task failure ratio, and Quality of Experience (QoE).

The rest of this study is organized as follows. Section II examines current task-to-host matching methods and highlights their limitations. Section III elaborates on the proposed conflict-reduced task-to-host matching scheme. Section IV presents the performance evaluation and discusses the results. Finally, the conclusion is provided in Section V.

II. Literature Review

While task-to-host matching optimization in edge computing networks has been widely investigated, most existing approaches primarily consider the current state of the edge network, neglecting tasks that are still transmitting data and approaching the network. This limitation contributes to a rise in unanticipated matching conflicts.

Early research relied on a centralized model, where a single controller was tasked with managing all resource-to-task assignments. For example, Zhang *et al.* [21] introduced an algorithm that virtualizes resources associated with each Access Points (AP) into a unified pool, allocating CPU resources based on task requirements and available capacity in each time slot. However, this approach does not possess global optimization capabilities, and it fails to ensure the completion of tasks within the required time frame, as resources are allocated on a best-effort basis. Additionally, this method does not consider the differences in data volume between tasks, limiting its applicability in real-world scenarios.

To enhance adaptability, Tong *et al*. [22] introduced a DQN-based resource matching strategy that jointly optimizes task execution time and host load balancing. The decision policy is guided by a weighted combination of multiple metrics. While the method improves multidimensional scheduling balance, its static weight configuration struggles to adapt to dynamically changing task offloading patterns.

Ghasemzadeh *et al*. [23] later presented an enhanced Genetic Algorithm (GA) designed to optimize both task execution latency and load balancing simultaneously. The algorithm analyzes task delays and host workloads to find a near-optimal allocation with minimal computational overhead. However, it fails to take resource limitations of edge hosts into account and ignores the variability in task data size, making it less effective in complex, real-world environments.

To overcome the bottlenecks of centralized coordination, several studies have explored distributed matching strategies to improve responsiveness and scalability. Chu *et al*. [24] introduced a technique that breaks down the joint optimization problem into multiple smaller, independent sub-problems, facilitating efficient allocation of tasks to Virtual Machines (VMs). While this method is computationally efficient, it is designed for specific network environments and needs re-optimization if network conditions change. Furthermore, the parameters for VM resources are static, which limits flexibility.

In large-scale IoT systems, Gao *et al*. [25] developed a distributed framework for resource allocation using Multi-Agent Deep Deterministic Policy Gradient (MADDPG), in which multiple Deep Deterministic Policy Gradient (DDPG) agents optimize task success rates by allocating precise computational resources and bandwidth. However, this method only considers CPU resource constraints, neglecting the storage needs of tasks. Additionally, it does not address the issue of communication latency between the distributed agents, which could impact performance in large-scale settings.

In order to effectively manage the complexities of ever-changing networks and the diverse needs of tasks, advanced multi-tier allocation models have been proposed. Ren *et al*. [26] introduced an innovative Deep Reinforcement Learning (DRL)-driven framework for task offloading and resource allocation, which adapts dynamically to optimize task-to-host mappings. This method takes into account a variety of parameters, including transmission speeds, task processing durations, energy expenditure, resource availability, and storage requirements. Importantly, the approach does not rely on pre-existing knowledge of the network's status; instead, it refines its strategies based on continuous interaction with the environment. Despite its potential, the scalability of this deep learning model is a concern when the number of variables and constraints increases. Additionally, the method assumes a fixed number of offloaded tasks, limiting its capacity to adjust to fluctuations in task loads.

Zou *et al*. [27] presented a novel distributed asynchronous approach built on the Asynchronous Advantage Actor-Critic (A3C) reinforcement learning framework. In this setup, task allocation schedulers are distributed across edge locations where APs are situated. These schedulers engage in asynchronous communication with a centralized controller, enabling the continuous update and optimization of resource allocation strategies.

Initially, tasks are matched with local resources, but when these resources are exhausted, the system attempts to horizontally offload tasks to other nearby sites, taking into account the predicted task completion times at alternative locations. This solution is effective for balancing workloads across different sites, but it fails to address potential conflicts between the decision to offload tasks vertically (to nearby edge sites) versus horizontally (to distant edge sites), which can lead to suboptimal performance.

Wang *et al*. [28] proposed a hybrid method combining greedy scheduling with an advanced workload monitoring system to enhance task completion rates and reduce system delays. In this framework, tasks are initially assigned to the local edge hosts based on a greedy allocation approach. A higher-level controller continuously observes the resource usage across the system and allocates backup resources when hosts approach full capacity. If a host is near its limit, tasks are redirected horizontally to available hosts. While this method helps mitigate timeout failures, it introduces latency during the migration process, particularly when tasks are first directed to an overloaded host and only rerouted once the capacity is exceeded.

Zhang *et al*. [29] leverages Federated Learning (FL) techniques to optimize task offloading and resource allocation. In this study, the control component of the machine learning model is deployed on a cloud platform to coordinate distributed scheduling nodes across multiple edge computing sites. This design facilitates global optimization of task offloading and resource allocation from a centralized perspective. However, the approach does not explicitly address the optimization of horizontal offloading of computing tasks across multiple edge nodes.

In Ref. [30], a Stackelberg game model is employed to facilitate fine-grained decision-making for task offloading and resource allocation. Building upon this, a federated learning-based algorithm is integrated to extract insights from historical scheduling patterns, enabling the system to progressively accumulate optimization knowledge. This hybrid framework effectively combines the strategic accuracy of Stackelberg game theory with the global learning capabilities of federated learning. Nevertheless, one limitation of this approach lies in the relatively long learning curve, which may impair the system's responsiveness to dynamic environmental changes. Additionally, since IoT devices are directly involved in the offloading optimization decision-making process, their computational overhead is further increased, potentially straining their already limited processing capabilities.

As discussed above, although existing studies on task-to-host matching have explored various optimization strategies from different perspectives, they still exhibit limitations as presented in Table I. Although many studies attempt to optimize task-to-host allocation, existing approaches lack consideration for the estimation of arrival times for tasks still undergoing transmission. This oversight frequently leads to matching conflicts. Therefore, designing a matching mechanism that can dynamically perceive the transmission progress of tasks, adapt to the rapidly changing conditions of complex network environments, and effectively reduce scheduling conflicts remains a critical and ongoing research challenge in IoT systems supported by edge computing.

TABLE I. COMPARISON OF EXISTING APPROACHES

Work	Technique	Strengths	Weaknesses
[21]	Lyapunov optimization	Lightweight computation	Ignores task data volume differences
[22]	DQN	Balances load while minimizing execution time	Static weights lack adaptability
[23]	Genetic Algorithm (GA)	High cost-effectiveness	Near-optimal, not fully optimal
[24]	Utility-Based Approach	Efficient in computation	Restricted flexibility
[25]	Multi-Agent Deep Deterministic Policy Gradient (MADDPG)	Highly adaptable	Costly communication
[26]	Deep Reinforcement Learning (DRL)	Tiered resource allocation	Assume a fixed number of offloaded tasks
[27]	Asynchronous Advantage Actor-Critic (A3C)	Cross-site global optimization	Overlooks conflicts between vertical and horizontal task offloading
[28]	Greedy	Decoupled two-level matching	Overlooks conflicts between vertical and horizontal task offloading
[29]	Federated Learning (FL)	Cross-site global optimization	Lack of the optimization of horizontal offloading
[30]	Stackelberg game with FL	Refine local decision-making accuracy	Heavy computational load on the IoT device side

III. TWO-LAYER ASYNCHRONOUS CONFLICT-REDUCED TASK-TO-HOST MATCHING SCHEME

In this section, the two-layer asynchronous conflict-reduced task-to-host matching scheme is proposed to optimize the allocation of computational tasks to edge hosts. The proposed scheme jointly considers the transmission progress of task data and the computational load of edge hosts, effectively reducing task timeouts due to resource contention.

We consider an IoT network composed of multiple edge computing sites denoted by the set $M = \{1, 2, \dots, M\}$, and numerous devices denoted by the set $D = \{1, 2, \dots, D\}$, as illustrated in Fig. 1. Each edge site consists of a single AP co-located with multiple edge hosts, denoted by the set $H = \{1, 2, \dots, H\}$. High-speed data links are utilized to interconnect the edge computing sites, while communication between IoT devices and APs is established via wireless connections. Each device in the network is capable of generating computation tasks that can be offloaded to edge computing sites for processing. Let $I = \{1, 2, \dots, I\}$ denote the set of all tasks generated in the system, where each task is indexed by $i \in I$.

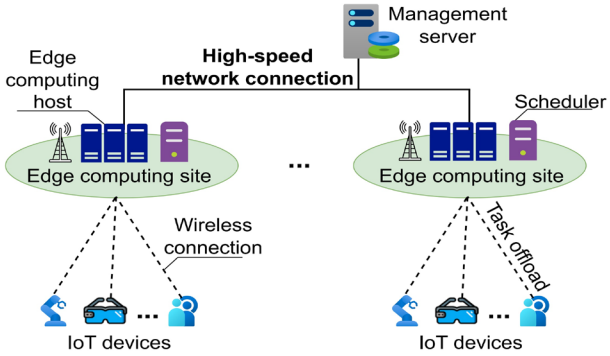

Fig. 1. Network architecture of edge computing-enabled IoT.

The TACTM scheme is designed with a hierarchical two-layer structure, and the algorithms within each layer execute independently in an asynchronous manner, as illustrated in Fig. 2. The lower layer performs task-to-host matching within each individual edge site, while the upper layer coordinates task allocation across multiple edge sites. while minimizing task matching conflicts.

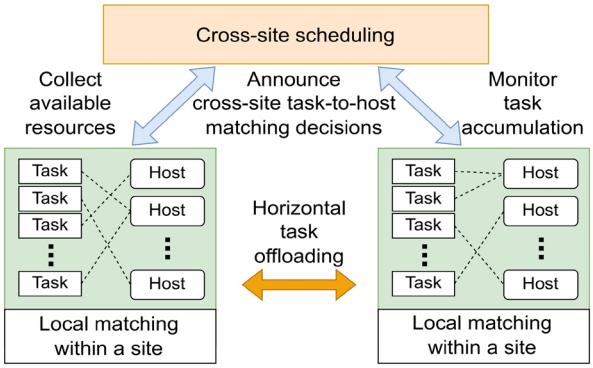

Fig. 2. Two-layer asynchronous task-to-host matching architecture.

A. *In-site Task-to-host Matching at the Lower Level*

In the lower-level algorithm, the depth-first approach is introduced to provide optimized task-to-host matching within each site. To reduce the computational cost of the algorithm, pruning is applied to avoid exploring a large number of meaningless allocation strategies.

Differing from previous methods, the computational resource demand of each task is recalculated due to the delay introduced during task data transmission before arriving at the edge computing site, as shown in Eq. (1).

$$CPU_{REQ}^{MIPS}(ts) = \frac{Instructions\ required}{Time\ left\ before\ the\ deadline} \quad (1)$$

where $CPU_{REQ}^{MIPS}(ts)$ is the Millions of Instructions Per Second (MIPS) required to achieve completion within the allotted time at time slot *ts*.

Moreover, the available resources of edge hosts are not allocated based on instruction counts or percentages, but rather in terms of MIPS. This enables support for the dynamic variation in computational resource requirements caused by changes in the remaining execution time of tasks. At each time slot *ts*, the dynamic variation of available computational resources on the edge host is determined by Eq. (2).

$$H_j^{cpu\_avl}(ts) = H_j^{cpu\_tot} - \sum Task_i^{cpu\_exe}(ts) \quad (2)$$

where $H_j^{cpu\_avl}(ts)$, $H_j^{cpu\_tot}$ and $Task_i^{cpu\_exe}(ts)$ represent the available CPU resources of edge host jjj at time slot tststs, the total CPU resources of host jjj, and the CPU resources occupied by task iii at time slot tststs, respectively.

Algorithm 1 outlines the procedure for in-site task-to-host matching.

Algorithm 1: In-site task-to-host matching

Input: Tasks pending matching and host status
Output: In-site matching result

```
// Define the boundary procedure
procedure boundary
    if Available_CPU_resources == 0 then
        return current_exploration_results
    end if
    for each pending task do
        if a task can be assigned to the host then
            Accumulate current_value
        else
            Estimate the potential value of the current path
            break
        end if
    end for
    return current_value
end procedure
// Define the search procedure
procedure df_search
    if the current path has insufficient value then
        Stop exploring and return
    end if
    if all tasks have been matched then
        Output matching result
    end if
    call procedure boundary
    if (current_value + ) < rest_value then
        Stop exploring and return
    end if
    call df_search recursively
    for each host do
        if available resources > demand then
            Continue the df_search recursively with the next task
        end if
    end for
end procedure
// Main program
Compute the CPU resource needs of each task in relation to their deadlines
call procedure df_search
```

In Algorithm 1, the procedure boundary, used to compute the matching boundary, is defined (Algorithm 1,

lines 2–15). Within this procedure, hosts with insufficient available resources are filtered out, as they are not eligible for further matching (Algorithm 1, lines 3–5). Then, each task is traversed, and the value of the current exploration path is computed based on the outcome of attempted matchings (Algorithm 1, lines 6–13). Finally, the procedure outputs the computed value of the current exploration path. In parallel, the procedure df_search is defined to perform recursive depth-first exploration of task-to-host matchings (Algorithm 1, lines 17–34). Within this procedure, two base conditions for terminating the search are first checked: the current path has insufficient value and all tasks have been successfully matched (Algorithm 1, lines 18–23). Next, the boundary procedure is invoked to estimate the potential value of the current exploration path. If the sum of the current path value and the estimated potential value is still lower than the remaining optimal value, the search is terminated, and the branch is pruned (Algorithm 1, lines 25–27). Consequently, the df_search procedure is recursively called by iterating over each host. If a host has sufficient available resources to meet the task's requirements, the search proceeds recursively to process the next task (Algorithm 1, lines 28–32). Finally, in the main program, each task's computational resource requirement is updated, and the df_search procedure is invoked to initiate the search for an optimized matching strategy (Algorithm 1, lines 36–37).

Specifically, the time complexity of the in-site task-to-host matching algorithm primarily depends on the number of tasks iii and the number of hosts j. In the worst-case scenario, each task may be matched to any of the i hosts, resulting in a total of j^i possible combinations. During each search step, the algorithm evaluates the current state and explores potential matches for all hosts, with an additional call to the boundary function to estimate the value of the current path. This leads to a per-step overhead of $O(i \cdot j)$, and thus a worst-case total time complexity of $O(i \cdot j^{i+1})$. However, the algorithm incorporates pruning strategies based on current value checks and upper-bound estimates through the boundary procedure, which effectively reduces unnecessary exploration. As a result, the actual runtime is significantly lower than the worst-case bound, especially when the pruning is effective.

The space complexity of the algorithm is mainly determined by the recursion depth and the storage of intermediate states. Since the algorithm uses depth-first search, the maximum recursion depth corresponds to the number of tasks, i.e., $O(i)$. Additionally, during the matching process, the algorithm maintains the state of each task and the available resources on each host, which requires $O(i + j)$ space. Therefore, the algorithm is relatively efficient and scalable, even in large-scale application scenarios.

B. Cross-site Matching at the Upper Level

The upper-layer algorithm is responsible for monitoring the task backlog at each site and performing cross-site task-to-host matching to allocate them to idle hosts. However, new tasks generated by IoT devices continuously arrive at edge computing sites through wireless data transmission. To avoid conflicts between horizontally offloaded tasks (i.e., cross-site) and newly arriving tasks from the vertical direction (i.e., from devices), TACTM continuously monitors the transmission progress of uploaded task data and estimates the arrival time of new tasks at their target edge computing sites.

Two-layer Asynchronous Conflict-reduced Task-to-host Matching (TACTM) establishes a task arrival record table for each edge computing site, as presented in Fig. 3. The arrival timeslot of each task is estimated based on its data transmission progress. Since the transmission speed of task data varies dynamically in each timeslot, the estimated arrival time of tasks at the edge computing sites is also updated dynamically in real time, as shown in Eq. (3).

$$T^{Arrival} = \frac{Remaining\ data}{Current\ transfer\ rate} + T^{Current} \tag{3}$$

where $T^{Arrival}$ and $T^{Current}$ correspond to the time when the task data reaches the edge network and the present time, respectively.

Thus, TACTM is able to simulate task-to-host matching in advance at each site by leveraging the in-site matching algorithm and temporarily reserving available host resources based on the predicted incoming tasks. In this way, cross-site task matching conflicts are reduced, as computing resources at the target site have already been pre-allocated to upcoming tasks.

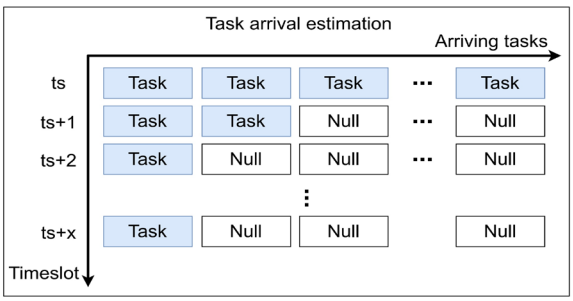

Fig. 3. Task arrival estimation.

The upper-level cross-site matching algorithm leverages A3C reinforcement learning, as it must jointly account for the status of all hosts across the edge network and the task accumulation at each site. To operate effectively in a dynamic environment, the algorithm must support adaptive learning and real-time optimization. Accordingly, the definitions of state, action, and loss are as follows.

State space: Define S as the set of all possible system states. At each time slot st, the system occupies a state $s(ts)$ such that $s(ts) \in S$, which can be expressed as Eq. (4):

$$S(ts) = \left\{CM_i^{Pending}(ts), ACM_m^j(ts)\right\} \tag{4}$$

where $S(ts)$ is the system state at time st, $CM_i^{Pending}$ is the required CPU MIPS for pending tasks, and ACM_m^j is

the available CPU MIPS of host j at edge computing site m.

Action space: In each time slot *ts*, decision-making is performed by the intelligent agent, aiming to optimize the system reward derived from task-to-host mapping. Denote *A* as the action space; each composite action is represented as Eq. (5):

$$A_i(ts) = \begin{cases} H_m^j \\ H^{none} \end{cases} \tag{5}$$

where $A_i(ts)$ and H_m^j represent the decision to assign task i at timeslot ts and host j at edge computing site m, respectively. In addition, H^{none} represents a decision where the task remains unmatched to an edge computing host during the current inference cycle. In actual implementation, the neural network inference yields a probability distribution indicating the likelihood of assigning the task to each edge computing host, as shown in Fig. 4.

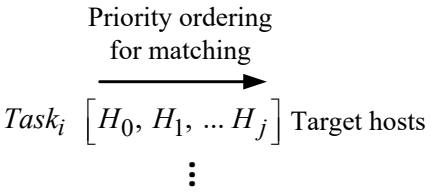

Fig. 4. Selection of model inference results.

Based on this distribution, TACTM directs the task to the host associated with the highest probability. The detailed logic for host selection is formulated in Eq. (6).

$$H_i^{selected}(ts) = \underset{H_j}{argmax}\, P_{i,j} \tag{6}$$

where $H_i^{selected}(ts)$ denotes the target host matched for task *i* at time slot *ts*, and $P_{i,j}$ represents the matching preference probability of task *i* for host *j*.

Reward: Since the objective of cross-site task-to-host matching is to maximize the allocation of available computational resources to pending tasks, the unified system reward is formally defined as Eq. (7):

$$Loss = -\frac{\sum CM_k^{Matched}}{\sum CM_i^{Pending}} \tag{7}$$

where $CM_k^{Matched}$ and $CM_i^{Pending}$ represent the amount of CPU MIPS required by the matched task and the total amount of CPU MIPS required by all pending tasks, respectively.

Since the upper and lower layers of the TACTM algorithm operate asynchronously, there is a certain probability that they may produce different matching decisions for the same task. To address this issue, TACTM assigns a lower priority to the decisions made by the cross-site matching algorithm in the upper layer, compared to those made by the in-site task-to-host matching algorithm in the lower layer. When a conflict occurs between the two layers, the cross-site matching decision is discarded as shown in Eq. (8).

$$H_i^{final} = \begin{cases} H_i^{in-site} & \text{if } H_i^{in-site} \neq \text{none} \\ H_i^{selected} & \text{otherwise} \end{cases} \tag{8}$$

where H_i^{final} indicates the final target host for task *i*, and $H_i^{in-site}$ denotes the in-site host selected by the lower-layer algorithm, and $H_i^{selected}$ is the cross-site candidate host selected by the upper-layer algorithm.

A summary of the workflow for cross-site task-to-host matching is presented in Algorithm 2.

Algorithm 2: Cross-site task-to-host matching

Input: Status of available computing resources across all hosts and the set of pending tasks within each edge site

Output: Cross-site matching result

1. **for** each time slot **do**
2. **if** number_of_accumulated_tasks > 0 **then**
3. Recalculate the computational resource requirements of each task according to the current time
4. Update the system state *S*
5. Infer the matching results *A* and select the match with the highest probability
6. Validate the matching to avoid duplicate assignments
7. Delete corresponding tasks from the pending queue
8. Output matching result
9. **end if**
10. **end for**

In Algorithm 2, the outer loop drives the algorithm to operate continuously at each timeslot (Algorithm 2, lines 1 and 10). At the beginning of each timeslot, the algorithm checks whether there is any task backlog at the edge computing sites. If no backlog is detected, cross-site task matching is skipped (Algorithm 2, lines 2–9). Conversely, if some sites do have accumulated tasks, the computational resource requirements of the queued tasks are recalculated based on the current time, and the system state parameters used by the reinforcement learning algorithm are updated accordingly (Algorithm 2, lines 3–4). Next, an A3C-based reinforcement learning model infers the target host recommendations for each task. To prevent duplicate assignments, the algorithm checks whether a task has already been matched locally within a site, since the upper-layer and lower-layer matching algorithms operate asynchronously. When a conflict occurs between the two layers, the cross-site matching decision is discarded (Algorithm 2, lines 5–6). Subsequently, successfully matched tasks are removed from the pending queue, and the matching results are recorded as output (Algorithm 2, lines 7–8).

Specifically, the time complexity of the A3C algorithm primarily depends on the number of agents M, the number of interaction steps each agent performs with the environment T, and the computational cost of the neural

network model, including both forward and backward passes. Overall, the time complexity per update can be expressed as $O(M \cdot T \cdot C_{model})$, where C_{model} denotes the cost of a single forward-backward cycle of the model. In practical applications, A3C leverages asynchronous parallel execution, allowing multiple agents to run concurrently and significantly improving training efficiency, despite the theoretical linear growth in computational cost. This makes it well-suited for the large-scale state–action space problem addressed in this study.

The actual deployment of the upper-layer algorithm should be determined based on the application scenario, as reinforcement learning algorithms incur a certain level of learning overhead. In scenarios where edge computing network resources are abundant, the upper-layer algorithm should be deployed at the edge network layer to reduce network transmission latency. In contrast, it can be deployed on a resource-rich cloud computing platform to improve the algorithm's execution speed.

IV. Result and Discussion

In this section, the experimental results are presented and thoroughly discussed. The performance of the proposed TACTM scheme is assessed through simulation-based evaluation. Initially, the simulation environment, system configurations, and parameter settings are introduced. Subsequently, the outcomes are analyzed from various perspectives.

TABLE II. Parameter Settings in the Simulation

No.	Parameters	Value
1.	Duration of simulation (Mins)	30
2.	Number of edge computing site	10
3.	Number of host in edge computing site	10
4.	MIPS of host on edge network	10,000
5.	Bandwidth of WLAN (Mbps)	100
6.	Data transmission model	MMPP/M1
7.	Number of mobile IoT devices	1000–4000
8.	Task length (GI)	1–45
9.	Task data size (KB)	0.5–2500

A. Simulation Settings

Performance verification of the TACTM mechanism was carried out through simulation trials. For this purpose, the EdgeCloudSim [31] tool was adopted to emulate the edge computing offloading environment. The experimental platform utilized a machine configured with an Intel Core i7 processor and 16 GB RAM to ensure stable simulation execution. To evaluate the adaptability of the TACTM scheme, task heterogeneity is incorporated into the experimental setup. Light and heavy computation tasks are randomly initiated by IoT devices in a mixed manner. The task lengths vary by up to a factor of 45. In addition, the size of the data associated with each task also differs. Specific configuration details are presented in Table II.

The simulation adopts a real-world distribution of computing sites, where 10 base stations are randomly selected from those deployed by Optus in the CBD area of Melbourne, Australia, to serve as access points for the edge computing network, covering a total area of 6.2 square kilometers [32], as shown in Fig. 5.

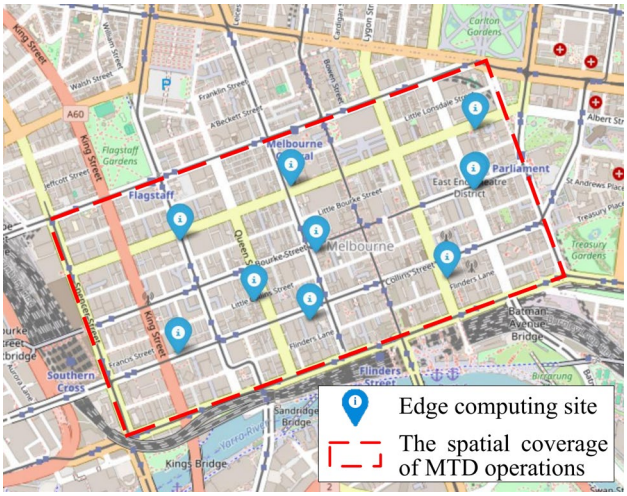

Fig. 5. The geographical distribution of edge computing sites.

In order to assess the effectiveness of the TACTM scheme, the average conflict ratio, average processing time, average multiple horizontal offload ratio, average task failure ratio, and average QoE are adopted as evaluation metrics for comparison with benchmark methods.

The task-to-host matching scheme designed to implement similar functionality was selected as the benchmark. The specifics are described below:

- First-Fit [31]. The most classic and widely adopted task-to-host mapping method. It checks hosts one by one for each task and stops when it finds the first host that meets the resource requirements of the task. It serves as the baseline in this study.
- M2TORA [33]. This approach employs multi-agent reinforcement learning to manage all hosts in the edge computing network as a shared resource pool, enabling coordinated optimization of task allocation.
- Vanilla-A3C [34]. This approach leverages deep reinforcement learning and employs asynchronous parallelism to optimize the task allocation strategy.

The following Section B presents a detailed discussion of the simulation results and numerical analysis based on this environment.

B. Analysis of Results and Discussion

Reducing matching conflicts is one of the primary objectives of this study. Fig. 6 presents a comparison of the average conflict ratio under varying numbers of Mbile IoT devices between the proposed TACTM scheme and the benchmark algorithms. It can be observed that the baseline method exhibits the highest conflict ratio, while TACTM achieves the lowest. This is primarily because the baseline approach greedily assigns tasks to hosts that satisfy computing resource requirements, while completely ignoring potential matching conflicts between horizontally and vertically offloaded tasks.

Additionally, although the two benchmark algorithms do not incorporate a conflict-aware assignment mechanism, their reinforcement learning techniques enable them to dynamically adapt to changes in the network environment. During this adaptive process, certain patterns of potential conflicts can be implicitly learned, resulting in a lower conflict ratio compared to the baseline. In contrast, the TACTM scheme features task-arrival awareness by tracking the data transmission progress of each task, which allows it to more effectively reduce the likelihood of task matching conflicts.

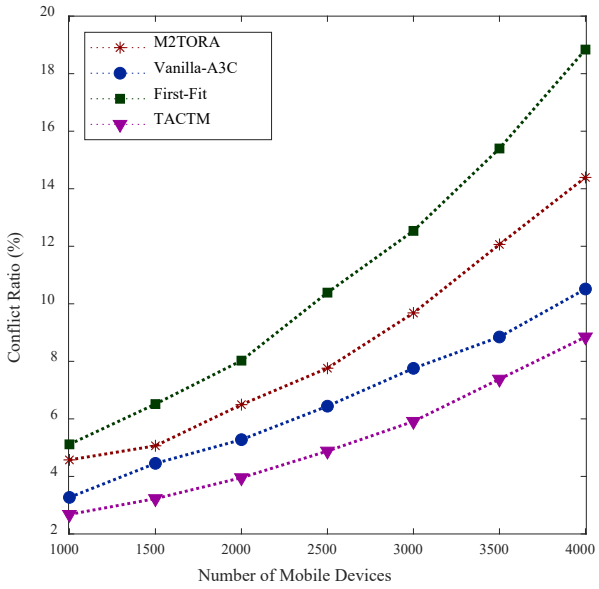

Fig. 6. Average conflict ratio with different number of IoT devices.

Fig. 7 illustrates the average processing time of four different task offloading schemes as the number of IoT devices increases from 1000 to 4000. The processing time is measured in seconds. The First-Fit algorithm exhibits the highest average processing time, primarily due to a high rate of task matching conflicts, which results in frequent execution delays. Additionally, this method lacks global optimization capabilities and relies on a greedy strategy to select target hosts. In contrast, M2TORA and Vanilla-A3C demonstrate lower average processing times, as both incorporate self-learning mechanisms and global optimization capabilities. However, M2TORA makes host allocation decisions during the offloading stage, without accounting for the uncertainty in task arrival times caused by interference during subsequent data transmission. On the other hand, Vanilla-A3C optimizes host matching based on tasks that have already arrived at the edge computing site, leading to more accurate task state information and thus lower processing time. However, due to the lack of explicit conflict control mechanisms, Vanilla-A3C still shows a higher processing time compared to the TACTM scheme.

There is a clear correlation between task timeouts and processing time, as higher processing times tend to result in an increased task failure ratio. Fig. 8 presents the average task failure ratio of the four task matching schemes as the number of IoT devices increases. The failure ratio is expressed as a percentage. Since the baseline method shows the highest processing time, it also results in the highest task failure ratio. In comparison, the benchmark schemes achieve lower failure ratios than the baseline. As anticipated, the TACTM scheme consistently achieves the lowest failure ratio among all methods. In particular, when the number of Mobile IoT Devices reaches 2500, the difference in average failure ratio between TACTM and Vanilla-A3C becomes notable, reaching 2.84%. This gap further widens to 9.65% when the devices count increases to 4000. These findings underscore the advantage of TACTM in reducing task failure rates, especially in scenarios with higher devices density in IoT networks.

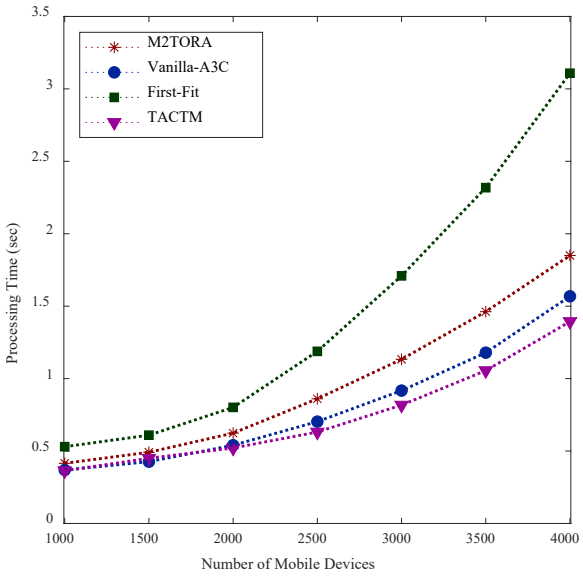

Fig. 7. Average processing time with different number of IoT devices.

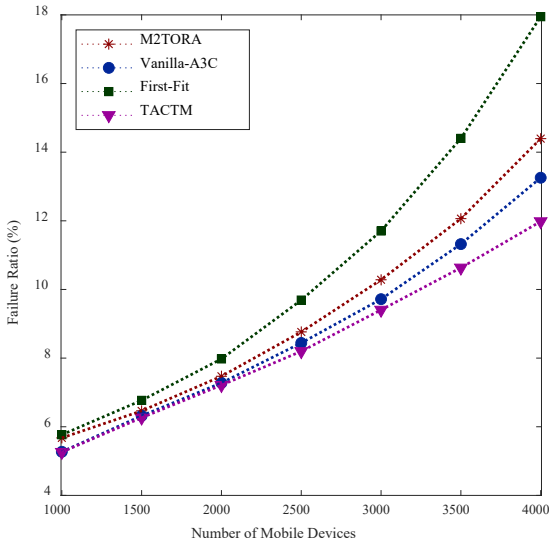

Fig. 8. Average task failure ratio with different number of IoT devices.

Fig. 9 illustrates the difference in user-perceived QoE when using IoT services, based on task offloading optimized by different matching methods. The TACTM scheme consistently delivers the highest QoE under all

devices quantities, which highlights the superior performance of this approach. Even at a scale of 4000 devices, TACTM achieves significantly better QoE compared to other schemes, indicating strong stability and reliability in large-scale environments.

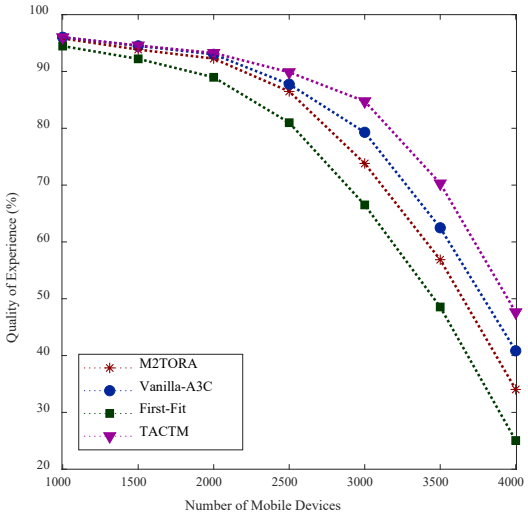

Fig. 9. Average QoE with different number of IoT devices.

In comparison, M2TORA and Vanilla-A3C perform less effectively in large-scale scenarios, as reflected by a substantial drop in QoE. Moreover, the turning point at which the QoE begins to decline rapidly occurs later under TACTM. Specifically, a noticeable decline in QoE for TACTM begins when the number of devices reaches 3000. In contrast, the benchmark methods experience a sharp decline in QoE at an earlier stage, suggesting limitations in scalability and adaptability.

In summary, the experimental results demonstrate that the proposed TACTM method outperforms existing approaches across various scenarios in terms of conflict ratio, processing time, task failure rate, and QoE, particularly under different scales of IoT device deployment. The consistent improvements observed throughout the experiments highlight the effectiveness and robustness of TACTM in addressing the challenges of task offloading in edge computing environments.

V. Conclusion

This study investigates the task offloading problem in edge computing-enabled IoT networks and introduces a novel scheme, termed TACTM. The proposed scheme aims to improve task execution efficiency and enhance overall IoT network performance by minimizing matching conflicts through the optimized task-to-host matching process. A detailed analysis of existing methods highlights their limitations, especially in managing tasks that are nearing the end of data transmission and approaching the edge computing sites. Such scenarios frequently result in matching conflicts during the offloading phase.

To address these issues, the TACTM scheme is designed by jointly considering the task data transmission progress and the dynamic load of edge computing hosts. The effectiveness of TACTM is validated through the development of a simulation environment and extensive comparative experiments. Experimental results demonstrate that, compared to existing methods, TACTM achieves significant improvements in multiple performance metrics. Specifically, it reduces the average conflict ratio by 20.79%, shortens the average processing time by 8.26%, lowers the average task failure ratio by 4.30%, and improves the average QoE by 4.05%.

Although the TACTM scheme demonstrates strong performance in simulation experiments, certain limitations of the current study are acknowledged. Further research is needed to extend TACTM to scenarios involving more diverse task types and more complex network topologies. In addition, we plan to explore methods for optimizing device energy consumption without compromising latency performance. Through these future efforts, TACTM is expected to offer a more comprehensive and effective solution to the task offloading problem in edge computing-enabled IoT environments.

Overall, the proposed TACTM scheme provides an effective solution to the task-to-host matching problem in edge computing-enabled IoT networks. By reducing matching conflicts and optimizing task allocation, TACTM significantly enhances task execution efficiency and overall IoT network performance. As edge computing technologies continue to evolve and application scenarios become increasingly diverse, TACTM is expected to offer valuable theoretical and practical contributions to IoT network performance optimization, while also serving as a foundation and inspiration for future research and real-world applications.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

WD was responsible for the conceptual framework and the drafting of the initial manuscript; the methodology was jointly developed by BI and WD; statistical analysis and data interpretation were carried out by WD and LL; BI and KBAB contributed to the revision and refinement of the manuscript; overall supervision of the research was provided by KBAB; all authors had approved the final version.

Acknowledgment

We would like to thank Universiti Teknologi Malaysia (UTM) for the provision of the venue, infrastructure, and necessary computing resources throughout the course of this work.

References

[1] X. Jin, W. Hua, Z. Wang *et al.*, "A survey of research on computation offloading in mobile cloud computing," *Wireless Netw.*, vol. 28, no. 4, pp. 1563–1585, 2022.

[2] B. Isyaku and K. B. A. Bakar, "Managing smart technologies with software-defined networks for routing and security challenges: A survey," *Comput. Syst. Sci. Eng.,* vol. 47, no. 2, pp. 1839–1879, 2023.

[3] H. Chen, W. Qin, and L. Wang, "Task partitioning and offloading in IoT cloud-edge collaborative computing framework: A survey," *J. Cloud Comp.*, vol. 11, no. 1, 86, 2022.
[4] K. Lone and S. A. Sofi, "A review on offloading in fog-based internet of things: Architecture, machine learning approaches, and open issues," *High-Confidence Computing*, vol. 3, no. 2, 100124, 2023.
[5] A. Mahapatra, K. Mishra, R. Pradhan *et al.*, "Next generation task offloading techniques in evolving computing paradigms: Comparative analysis, current challenges, and future research perspectives," *Arch. Computat. Methods Eng.*, vol. 31, no. 3, pp. 1405–1474, 2024.
[6] R. Aldmour, S. Yousef, T. Baker *et al.*, "An approach for offloading in mobile cloud computing to optimize power consumption and processing time," *Sustainable Computing: Informatics and Systems*, vol. 31, 100562, 2021.
[7] B. Isyaku, K. B. A. Bakar, F. A. Ghaleb *et al.*, "Dynamic routing and failure recovery approaches for efficient resource utilization in OpenFlow-SDN: A survey," *IEEE Access*, vol. 10, pp. 121791–121815, 2022.
[8] B. Isyaku, K. B. A. Bakar, W. Nagmeldin *et al.*, "Reliable failure restoration with bayesian congestion aware for software defined networks," *Computer Systems Science & Engineering*, vol. 46, no. 3, 2023.
[9] A. Maia, A. Boutouchent, Y. Kardjadja *et al.*, "A survey on integrated computing, caching, and communication in the cloud-to-edge continuum," *Computer Communications*, vol. 219, pp. 128–152, 2024.
[10] B. Isyaku, K. A. Bakar, M. S. M. Zahid *et al.*, "Route path selection optimization scheme based link quality estimation and critical switch awareness for software defined networks," *Applied Sciences*, vol. 11, no. 19, 9100, 2021.
[11] F. S. Abkenar, P. Ramezani, S. Iranmanesh *et al.*, "A survey on mobility of edge computing networks in IoT: State-of-the-art, architectures, and challenges," *IEEE Communications Surveys & Tutorials*, vol. 24, no. 4, pp. 2329–2365, 2022.
[12] L. A. Haibeh, M. C. E. Yagoub, and A. Jarray, "A survey on mobile edge computing infrastructure: Design, resource management, and optimization approaches," *IEEE Access*, vol. 10, pp. 27591–27610, 2022.
[13] I. A. Elgendy, W.-Z. Zhang, Y. Zeng *et al.*, "Efficient and secure multi-user multi-task computation offloading for mobile-edge computing in mobile IoT networks," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2410–2422, 2020.
[14] W.-Z. Zhang, I. A. Elgendy, M. Hammad *et al.*, "Secure and optimized load balancing for multitier IoT and edge-cloud computing systems," *IEEE Internet of Things Journal*, vol. 8, no. 10, pp. 8119–8132, 2021.
[15] X. An, R. Fan, H. Hu *et al.*, "Joint task offloading and resource allocation for IoT edge computing with sequential task dependency," *IEEE Internet of Things Journal*, vol. 9, no. 17, pp. 16546–16561, 2022.
[16] Y. Hao, Z. Song, Z. Zheng *et al.*, "Joint communication, computing, and caching resource allocation in LEO satellite MEC networks," *IEEE Access*, vol. 11, pp. 6708–6716, 2023.
[17] Y. Yang, C. Long, J. Wu *et al.*, "D2D-enabled mobile-edge computation offloading for multiuser IoT network," *IEEE Internet of Things Journal*, vol. 8, no. 16, pp. 12490–12504, 2021.
[18] Z. Niu, H. Liu, Y. Ge *et al.*, "Distributed hybrid task offloading in mobile-edge computing: A potential game scheme," *IEEE Internet of Things Journal*, vol. 11, no. 10, pp. 18698–18710, 2024.
[19] F. Algarni, "A novel quality-based computation offloading framework for edge cloud-supported internet of things," *Alexandria Engineering Journal*, vol. 70, pp. 585–599, 2023.
[20] X. Jiao, H. Ou, S. Chen *et al.*, "Deep reinforcement learning for time-energy tradeoff online offloading in MEC-enabled industrial internet of things," *IEEE Transactions on Network Science and Engineering*, pp. 1–14, 2023.
[21] Q. Zhang, L. Gui, F. Hou *et al.*, "Dynamic task offloading and resource allocation for mobile-edge computing in dense cloud RAN," *IEEE Internet of Things Journal*, vol. 7, no. 4, pp. 3282–3299, 2020.
[22] Z. Tong, B. Liu, J. Mei *et al.*, "D2OP: A fair dual-objective weighted scheduling scheme in internet of everything," *IEEE Internet of Things Journal*, vol. 10, no. 10, pp. 9206–9219, 2023.
[23] A. Ghasemzadeh, H. S. Aghdasi, and S. Saeedvand, "Edge server placement and allocation optimization: A tradeoff for enhanced performance," *Cluster Comput.*, vol. 27, no. 5, pp. 5783–5797, 2024.
[24] W. Chu, P. Yu, Z. Yu *et al.*, "Online optimal service selection, resource allocation and task offloading for multi-access edge computing: A utility-based approach," *IEEE Transactions on Mobile Computing*, vol. 22, no. 7, pp. 4150–4167, 2023.
[25] Z. Gao, L. Yang, and Y. Dai, "Large-scale computation offloading using a multi-agent reinforcement learning in heterogeneous multi-access edge computing," *IEEE Transactions on Mobile Computing*, vol. 22, no. 6, pp. 3425–3443, 2023.
[26] J. Ren, T. Hou, H. Wang *et al.*, "Collaborative task offloading and resource scheduling framework for heterogeneous edge computing," *Wireless Netw.*, 2021.
[27] J. Zou, T. Hao, C. Yu *et al.*, "A3C-DO: A regional resource scheduling framework based on deep reinforcement learning in edge scenario," *IEEE Transactions on Computers*, vol. 70, no. 2, pp. 228–239, 2021.
[28] H. Wang, H. Xu, H. Huang *et al.*, "Robust task offloading in dynamic edge computing," *IEEE Transactions on Mobile Computing*, vol. 22, no. 1, pp. 500–514, 2023.
[29] J. Zhang, F. Huang, S. Zhu *et al.*, "A resource allocation strategy in internet of vehicles based on multi-task federated learning and incentive mechanism," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–12, 2025.
[30] H.-S. Kang, Z.-Y. Chai, Y.-L. Li *et al.*, "Edge computing in Internet of Vehicles: A federated learning method based on Stackelberg dynamic game," *Information Sciences*, vol. 689, 121452, 2025.
[31] R. Casadei, G. Fortino, D. Pianini *et al.*, "A methodology and simulation-based toolchain for estimating deployment performance of smart collective services at the edge," *IEEE Internet of Things Journal*, vol. 9, no. 20, pp. 20136–20148, 2022.
[32] Q. He, G. Cui, X. Zhang *et al.*, "A game-theoretical approach for user allocation in edge computing environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 3, pp. 515–529, 2020.
[33] C. Zeng, X. Wang, R. Zeng *et al.*, "Joint optimization of multi-dimensional resource allocation and task offloading for QoE enhancement in cloud-edge-end collaboration," *Future Generation Computer Systems*, vol. 155, pp. 121–131, 2024.
[34] S. Tuli, S. Ilager, K. Ramamohanarao *et al.*, "Dynamic scheduling for stochastic edge-cloud computing environments using A3C learning and residual recurrent neural networks," *IEEE Transactions on Mobile Computing*, vol. 21, no. 3, pp. 940–954, 2022.