J Grid Computing (2021) 19: 18
https://doi.org/10.1007/s10723-021-09559-x

# Context-Aware Multi-User Offloading in Mobile Edge Computing: a Federated Learning-Based Approach

**Ali Shahidinejad · Fariba Farahbakhsh · Mostafa Ghobaei-Arani · Mazhar Hussain Malik · Toni Anwar**

Received: 5 May 2020 / Accepted: 29 March 2021 / Published online: 14 April 2021


**Abstract** Mobile edge computing (MEC) provides an effective solution to help the Internet of Things (IoT) devices with delay-sensitive and computation-intensive tasks by offering computing capabilities in the proximity of mobile device users. Most of the existing studies ignore context information of the application, requests, sensors, resources, and network. However, in practice, context information has a significant impact on offloading decisions. In this paper, we consider context-aware offloading in MEC with multi-user. The contexts are collected using autonomous management as the MAPE loop in all offloading processes. Also, federated learning (FL)-based offloading is presented. Our learning method in mobile devices (MDs) is deep reinforcement learning (DRL). FL helps us to use distributed capabilities of MEC with updated weights between MDs and edge devices (Eds). The simulation results indicate our method is superior to local computing, offload, and FL without considering context-aware algorithms in terms of energy consumption, execution cost, network usage, delay, and fairness.

A. Shahidinejad (✉) · M. H. Malik
Department of Computing and IT, Global College of Engineering and Technology, Muscat, Oman
e-mail: ali.s@gcet.edu.om

F. Farahbakhsh · M. Ghobaei-Arani
Department of Computer Engineering, Qom Branch, Islamic Azad University, Qom, Iran

T. Anwar
Department of Computer and Information Sciences, Universiti Teknologi Petronas, Bandar Seri Iskandar, Perak, Malaysia

**Keywords** Mobile edge computing · Computation offloading · Context-aware · Federated learning

## 1 Introduction

In recent years, data production in various scientific and industrial fields and the limitation of resources to process has resulted in the need for a rich processing environment outside of the user's equipment. This led to creating the cloud computing environment with almost endless physical and virtual processing resources [1, 2]. In computation offloading to the cloud, we face problems due to the large distance of end-users to the cloud; for example, in cases where we need a real-time response, such as healthcare, waiting for receiving a response from the cloud can pose serious problems. The computational offloading was proposed to solve this problem in the fog environment to create a computational level closer to the end-users [3]. For several years, a new trend was emerging and putting cloud computing functions on the network's edge. One of the incentives for this approach is the mass production of network EDs (including Wi-Fi router and access point station). Due to the significant processing power of these devices, high-throughput and delay-sensitive functions can be implemented [4]. This computation model is called mobile edge computing (MEC) [5]. This technology is developed by the European telecommunications standards institute (ETSI) [6]. The main focus of the MEC is on radio access networks (RANs) in 4G and 5G cellular networks.

MEC has advanced features like latency, user proximity, high bandwidth, and location awareness [7]. These enable MEC to run many new types of applications and multi-region services, such as business and health, augmented reality, video streaming services, and more [8]. At MEC, the user's distance is much closer than the user's distance to the cloud. One of the key technologies of MEC is computational offloading, which can be realized from both single-user and multi-user aspects. In a single-user computational system, at any given moment, a user can offload the computing task. In contrast, in a multi-user computational system, multiple users are allowed to move their tasks to other computing layers simultaneously. As a difference between these two systems in the multi-user offloading, one module with different data related to different users can be offload to EDs or the cloud. In the single-user system, there is no need for any data management for users. Although many works have been done in computational offloading in recent years, the concept of context-awareness has been used very limitedly [9] in past research. Our meaning about context is using the properties of offloading, application, mobile, sensors, network and media, and resources. The context in computing offloading decisions will be very influential because of mobile conditions like location, network status, and available computing resources [10].

One of the issues that arise in offloading is intelligent tools to detect current or underlying conditions and implement context-based behavior [11]. This ability can be referred to as context-awareness. As soon as they make a network available, they perform the offload without considering whether the offloading is in their favor or not [12]. The computational burden is not always beneficial to obtain the required level of efficiency and benefit offloading. Here we aim to improve the delay and energy consumption by the proposed offloading method. The distributed nature of MEC requires an appropriate offloading method. For this purpose, the FL can be useful in this regard. FL can coordinate the training process among multiple MDs. The DRL technique is very efficient in finding the optimal offloading policy in MEC. Since DRL needs much processing, thus the DRL agent has to be carefully designed and implemented. Some challenges and their solutions in FL are as follows [13].

- The whole training dataset is not accessible. This challenge is created for the nature of distributed computation, and it can provide the privacy of data for all users.
- Slow and unstable communication. Based on the proposed approach, MDs are not completely dependent on EDs. As some nodes become offline, only the weights are less trained or updated later, but the task performing or offloading is done continuously.
- The trade-off between privacy guarantees and system performance. The computation tasks can be encrypted by a fast and trusted cryptography algorithm in IoT.
- Interference among MDs (The MDs may be geographically close to each other. This introduces an interference issue when they update local models to the server. Channel allocation policies may need to be combined with the resource allocation approaches to address the interference issue). DRL can be considered to model the dynamic environment of MEC and make optimized decisions.
- Comparisons with other distributed learning methods. Some methods use neural networks up to a cut layer, or others ignore to transmit weights to an aggregating server. FL has a more straightforward implementation since the participants and the FL server run the same global model in each cluster.
- Learning convergence. We improve this challenge with the loss function, as mentioned in the DRL algorithm.
- Size of model updates. The combinations of weights and contexts help us to reduce the size of model updates.

Because real systems and environments are multi-user and not single-user, we use multipurpose computing offloading in our approach. Since these users are located in different locations and conditions, thus the offloading decision should be made with the knowledge of context and the existing conditions. This strategy solves the problems shown above and improves the service efficiency in computational offloading. We propose a multi-user conditioned MEC system that changes the conditions of a mobile computing resource. In this research, the contexts are collected using the monitor phase of the autonomous management (MAPE control loop) located at the edge level, including application context, mobile devices, sensors, networks, edge servers, and media. These contexts are analyzed in the Analysis phase, and then these contexts are sent to the proposed context-aware algorithm in the Planning

phase. Finally, the Execution phase executes the offloading instruction. Using the proposed FL algorithm, the updated weights related to the DRL algorithm are exchanged between MDs and EDs. Our key contributions in this paper are as follows:

1. We provide a MAPE control loop for MEC to decide whether to offload the tasks locally or remotely in the edge or cloud. Also, we use context information of the application, sensors, resources, edge servers, and network. These updated contexts improve the offloading process.
2. For optimal use of the distributed capability of MEC, we propose an FL-based offloading algorithm that uses the DRL to train the MDs and sends the updated weights to EDs and the cloud. It causes lower data transmission from MDs to EDs and protects the users' information.
3. The proposed approach is evaluated based on some metrics: energy consumption, execution cost, network usage, delay, and fairness.

The rest of this paper is organized as follows. In section 2, related works are summarized. The system model and network architecture are presented in section 3. In section. 4, we explain our offloading algorithms in detail. In Section 5, the evaluation results of our proposed algorithms are presented and compared with other methods. Finally, in Section 6, the conclusion is discussed, and suggestions are made for future work.

## 2 Related Works

In recent years, many studies have been investigated about MCC and MEC. We classify these studies to multi-user and context-aware offloading as follows. Also, we collect and analyze some researches about FL.

### 2.1 Multi-User Offloading

There are some studies about multi-user offloading. Here, we mention these papers based on their objectives and methods. Researchers studied different objectives such as energy consumption [14], computation delay [15], QoS (Quality of service), latency, and accuracy [16]. According to [17], as with cloud services such as PaaS (platform as a service), IaaS (infrastructure as a service), SaaS (software as a service), computation offloading is also considered as offloading as a service (OaaS) in cloud computing.

Some researchers minimized the cost under constraints and solve the offloading problem in multi-user MEC by backtracking, genetic algorithm, and greedy strategies [18]. Chang et al. [19] investigated the computational offloading with an efficient energy scheme in a multi-user fog computing system. They used queuing to model the execution processes on mobile and fog devices. The problem of efficient energy optimization was formulated to minimize energy consumption conditional on delay constraints. A distributed algorithm called ADMM (based on the periodic multiplier method) was presented to solve the formulated problem. Their simulation results showed better performance than other existing designs. The authors in [14] solved a multi-objective scheduling problem to optimize time and energy consumption. They could improve the objectives by a whale optimization algorithm in the MCC. Paper [20] also worked on the energy consumption and also cost for computation offloading of workflow applications in MEC using a Non-dominated Sorting Genetic Algorithm (NSGA). The results were better than no offloading and cloud offloading methods.

It has been argued in [21] that although computational offloading can reduce power consumption on mobile devices, it may delay further execution, including sending time between mobile devices and cloud servers. According to theoretical analysis, a multi-objective optimization problem is formulated with reducing energy consumption, execution delay, and payment cost, by finding the optimal computational offloading and transmission power for each mobile device.

The results show decreasing in the mentioned objectives.

In [22], a mixed-integer linear programming (MILP) optimization model was used. This paper considers two types of cloud patches: the local cloud patch and the global cloud patch, which have higher capabilities. The model presented in this paper reduces energy consumption while imposing a significant amount of delay.

Researchers in [23] provided some disadvantages of cloud processing, such as high latency and unstable QoS (Data dissemination, routing between mobile devices, and cloud servers). Assuming different real-time computing tasks on different devices, each task is decided to either run locally on the device itself or to be offloaded to one of the edge servers or the cloud server. This paper examined low-complexity computing offloading policies to minimize the quality of MEC network service

assurance of mobile devices' power consumption. Their method was superior to other compared approaches.

### 2.2 Context-Aware Offloading

Considering context information in the offloading problem is done based on different objectives and network architectures such as energy-saving and execution time [24] and latency [25] in MCC and MEC. Chen et al. [26] proposed a framework that supports mobile applications with computational offload capability for aware conditions. First, a design pattern was proposed to enable the application to be offloaded on demand. Second, an estimation model was presented to select the appropriate cloud source for offloading automatically. Their framework includes three modules: service selection module, computational offload, and runtime management. The evaluation results and comparison with traditional offshore samples showed that the proposed approach could improve runtime and power consumption.

Ghasemi-Falavarjani et al. proposed an offloading middleware to the aggregate cloud by considering energy level, processing power, runtime, and network bandwidth [27]. The resource allocation problem was formulated as a multi-objective optimization that aims to optimize the completion time of the task and the energy consumption of all participating mobile devices by satisfying the task boundary. They used an NSGA-II to obtain the best solution. Besides, a multi-attribute decision-making (MADM) technique was used to determine the best compromise solution based on the entropy technique and weighting for a priority order. Their valuation Results showed that the proposed method managed well the compromise between completion time and energy consumption.

The researchers in [28] analyzed the context-aware energy optimization for services on MDs. Their evaluation was based on three supervised machine learning methods as naïve Bayesian, decision tree, and random forest. They showed that using the machine learning method is better than others for reducing the service execution time and the energy consumption in MCC.

Roostaei et al. proposed a fault-tolerant aware mobility offloading (MAFO) approach that collects network information and user mobility over time and uses the Markov chain of the user's visited networks in different possible paths [29]. It also predicts the stoppage time of each network based on user mobility. The evaluation results showed that improvements in time and energy consumption. Authors in [30] have suggested a framework called Thinkair that simplifies developers' work to migrate their smartphone apps to the cloud. It uses the concept of smartphone virtualization in the cloud and provides computation offloading. It focuses on the resilience and scalability of the cloud and enhances the power of cloud computing by implementing a parallelization approach using multiple virtual machines. Their results showed better performance and lower power consumption than similar non-parallel methods. In [31], a framework is considered to decide whether to offload a given method to cloud servers. In this paper, a field-aware decision-making algorithm was designed, implemented, and evaluated called CADA, which uses user contexts and historical metrics to optimize the performance of mobile devices with various optimization criteria such as short response time. Their evaluation results demonstrated high accuracy and improved response time and energy consumption compared to other approaches.

### 2.3 FL-Based Offloading

Using cooperative models has shown good performance in IoT devices [32]. FL is a cooperative-based method that can be used in MEC. Here we first try to present a conceptual view of FL and then provide some offloading problems that FL solves.

FL allows devices to collaboratively train a shared model while keeping data privacy on devices. Thus, FL can be used as an enabling technology for ML model training at MEC. Each device can process its task by a learning model, which is applicable to all devices. After that, all devices can share their experience. As a result, we will have a global model by aggregating all learning models. The worth-mentioning point is that in this cooperation, any private data are not transferred between devices.

Generally, there are two main entities in FL: N data owners {1,2,…,N} and the model owner (FL server). According to Fig. 1, in the initialization (step 1), the FL server specifies the hyperparameters of the global model and the training process, e.g., learning rate. Then, in step 2, each data owner i (Mobile device) train a local model $w_i$ and send it to the FL server (Controller). In step 3, all collected local models in the FL server are aggregated $w = \cup_{i \in N} w_i$. Steps 2 and 3 are repeated until the global loss function converges or a desirable training accuracy is achieved [13].

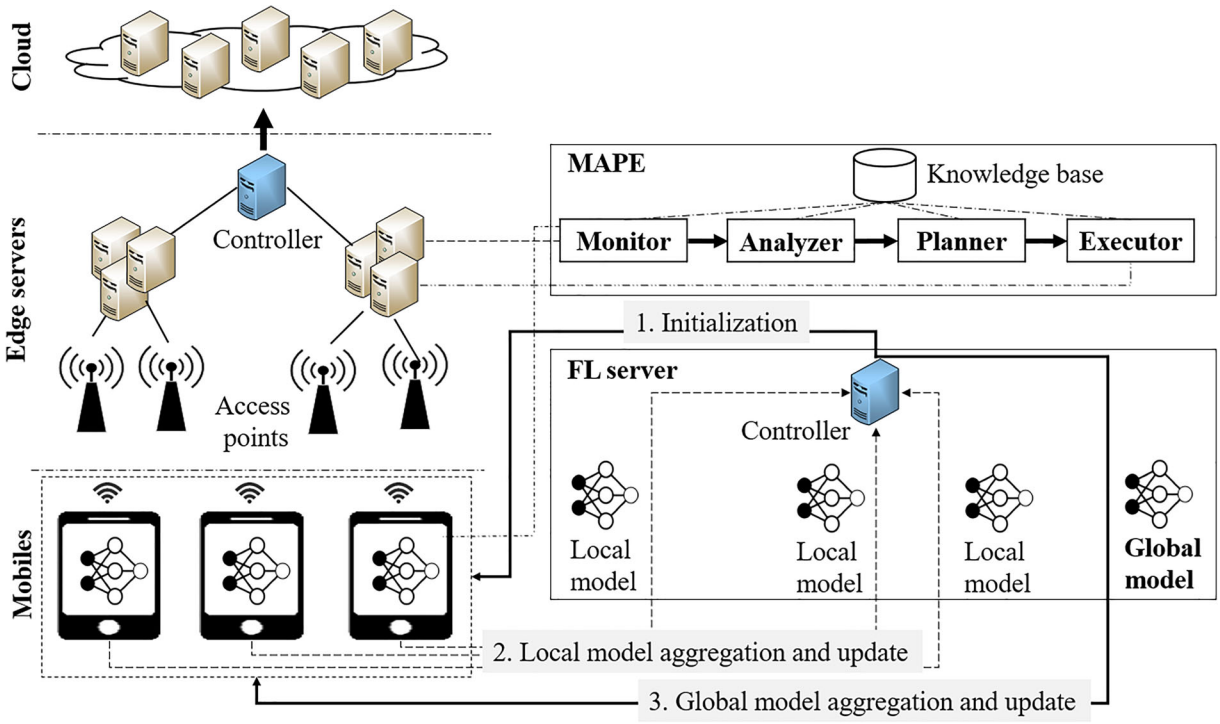


**Fig. 1** The architecture and system model

Since the offloading methods in MEC need a real distributed algorithm; thus, FL is an excellent way to this purpose. In [33], the authors presented an FL-based offloading in MEC. DRL algorithms are executed in MDs, and the updated weights are transfer between MDs and ED. Their used parameters were energy consumption and transmission time. Their results showed a better performance compared to centralized DRL. In centralized DRL, the tasks are waiting in a queue to get resources of devices. Therefore, many tasks may be dropped due to insufficient resources. Nevertheless, FL can offload some tasks to other devices which causes a lower drop task in devices.

A group of researchers proposed an aggregation model of EDs in the cloud by FL. They used the difference of convex functions (DC) representation for sparse and low-rank function [34]. It is demonstrated that the novel method was able to select more devices than other benchmark approaches. The paper [35] is presented based on a distributed DRL in MEC for caching and communication operations. This research includes three parts, information collecting, cognitive computing, and request handling. Their results showed some improvements in utility for the user equipment.

FL is also used to make decisions about computation offloading and energy allocation in MEC [36]. Here, a DRL-based algorithm is proposed to maximize the expected long-term utility. This method has better results than the centralized and greedy-based offloading algorithms.

As stated in the introduction section's contribution list, the main idea of this research is to use MAPE and FL-based offloading algorithm. Context information has been used in the previous works. We have tried to offload the modules with these contexts in the controlled MAPE loop with our distributed algorithm to improve the mentioned objectives. We try to compare our results with new researches. Also, federated learning is very close to the distributed learning paradigm. In previous studies, DRL or DL has been used in each device, and the devices do not offload the tasks to each other. We solve this issue by using federated learning.

The summary of offloading algorithms is categorized by technique, objectives, architecture, pros, and cons in Table 1.

## 3 System Model and Problem Formulation

In this section, we present the architecture and system model. Figure 1 shows the three-layer architecture of our proposed system.

**Table 1** Summary of offloading algorithms (ET: execution time. Arc: Architecture)

| Ref | Technique | Objectives | Arc. | Pros | Cons |
|---|---|---|---|---|---|
| [26] | FL-DRL | Energy and transmission time | MEC | Cooperative model with improve bandwidth crowding | Using only one ED |
| [27] | FL-DC | Accuracy and number of devices | Edge- Cloud | Selecting maximum devices | Ignoring edge computing objectives |
| [17] | Backtracking, genetic the algorithm, and greedy | Cost | MEC | Minimizing cost | Time-consuming method |
| [30] | FL-DRL | long-term utility | MEC | Analysis DRL parameters and energy efficiency | Ignoring privacy evaluation |
| [19] | LP, deep learning | ET, Energy | MEC | Improving network service quality and reducing power consumption | Long run time of the proposed algorithms |
| [16] | Queuing | Delay, cost, and energy | Fog | Decreasing transmission time in the worst condition | Non-optimal method in global goals |
| [37] | Oaas and matching algorithm | Cost | Fog | Real tools for offloading | Ignoring computational offloading time delay |
| [15] | Queuing and ADMM | Delay, energy | Fog | Cell-level alignment, minimizing power consumption | Solving the problem only up to the level of servers and override local implementation |
| [23] | Game theory, context (Network connection, runtime status) | Energy, ET | MCC | Decreasing execution time and energy consumption | High complexity and no comprehensive method |
| [25] | Markov chain, contexts (User mobility, device, program, cloud server, mobile network) | Energy, ET | MCC | Fault tolerance, low energy consumption, and ET | Non-comprehensive evaluation |
| [24] | OMMC, NSGAII, TOPSIS, contexts (Device context, processing power, CPU usage, network bandwidth) | Delay, energy, deadline | MEC | Trade-off between completion time and energy consumption | Ignoring method's complexity |
| [38] | Thinkair and contexts (Hardware, software, network condition, energy estimation model) | Cost, ET, energy | MCC | Resource allocation based on requests, parallel reconstruction of VMs | Unsuitable for IoT applications |
| [39] | CADA and contexts (Signal strength, transmission time, time-of-day, and location) | Energy, ET | MEC | Using daily time for offloading, decreasing energy consumption and ET | Weak energy model |

### 3.1 IoT

The IoT component is at the very bottom of the architecture, including communication devices that are connected through heterogeneous networks. In general, it aims to collect and process data through IoT devices to extract patterns and discover patterns or to perform predictive analysis or optimization and make smarter, more timely decisions. The IoT devices first collect the data, and each user sends their requests to the queue according to the data collected.

### 3.2 Controller

This component plays the role of the master node in our hierarchical model. It is located at the edge command center and is at the top edge of the edge. This component itself is a robust edge resource that manages resources for requests from the lower layer. The controller decides whether the request will be executed on the same edge layer or delivered to the cloud layer based on the users' context and existing resource conditions (existing edge server features).

### 3.3 Edge Server

The edge layer component consists of several edge servers that play the role of a slave in our hierarchical approach. The edge servers send information about their processing and storage capabilities to the controller. The controller selects one of the edge servers to execute the requests by matching the context information and resource capability.

### 3.4 MAPE

The MAPE control loop is the main component of our framework, including monitoring, analyzing, planning, and executing. This component collects the current conditions and resources available. Then it examines the available resources and decides whether to execute them on the edge layer or offload the super layer's computation. The MAPE control loop is located on the edge server controller component. Our context-aware algorithm is implemented in this component. To achieve autonomous computing, IBM has proposed a reference model for autonomous loops, known as the MAPE-K loop, which has four phases (monitor, analyze, plan, execute) [40]. All of these four phases exchange information using common knowledge.

In the monitoring phase, the properties of the environment are recorded by the sensors. The data is first received through sensors and intelligent equipment, and according to the data received, the requests for execution are made. The analysis phase deals with the processing of data collected from the monitoring phase. Any violation of the level of needs defined in the analysis phase is considered. Based on the information created in the previous two phases, an appropriate offloading decision is made in the planning phase. The execution phase executes the planned decision in the third phase. In fact, it is responsible for executing the programs approved by the analysis phase.

We propose a hierarchical model for the proposed system, in which the edge layer plays a master node role in which all four phases of the MAPE loop are implemented, and the other nodes play a slave role. With this smart, autonomous solution, decentralized collections are managed in a centralized system. Integrating centralized and distributed strategies can be important as an innovative strategy. Autonomous loop computing (MAPE) decision-making autonomously leads to better management of resources, reduced response time to heavily time-dependent applications and requests, and reduced system latency.

### 3.5 Cloud

When requests from the edge server controller are decided to be offloaded to the cloud layer, they are sent directly to the cloud gateway. The cloud layer delivers on-demand computing services from applications to storage and processing power.

### 3.6 Case Study

The VR-GAME (Virtual reality Game) application is a human-based game. According to the workflow of this application, EEG signals send to the client module. The client module sends consistent data to the concentration calculator module. The client module updates the game display to the player. The coordinator module gathers and distributes measured concentration among players [41].

The EEG value could be used to determine the interval between two sensed signals. Based on the application part of Fig. 2, the EEG sensor, display actuator, and

client module are placed in the mobile device. The concentration calculator and the coordinator modules can be placed in the EDs or cloud. The main problem of this paper is the offloading of modules as $\{M_1, M_2, .., M_k\}$ to edge servers as $\{ED_1, ED_2, .., ED_n\}$ or cloud. The problem formulation is explained as follows. The symbols used in this paper are defined in Table 2.

### 3.7 Local Execution Time

Suppose the required resources in MDs are provided. In that case, we can calculate the local execution time using Eq. (1), where μ is CPU cycles for processing the task and $\omega$ is the commonly adopted effective switched capacitance that depends on the architecture of chips [33]. According to [42], $\omega$ can be given by $\sum_{i=1}^{N} \alpha_i * C_{Li} * \Delta V_i$, where $C_{Li}$ is the physical capacitance, $\alpha_i$ is the activity weighting factor, each averaged over the N nodes in the circuit. Also, $\Delta V_i$ is the voltage change. It is worth mentioning that the $\omega$ value is calculated by the simulator.

$$T_i = \frac{\mu}{\sqrt{\frac{1}{\omega * \mu}}} \tag{1}$$

### 3.8 Data Rate between MDs and EDs

When an MD wants to communicate with an ED, a wireless link is established. The achievable data rate is calculated by Eq. (2). Here, $A$ is the power of interference plus noise, $s_i^e$ is the channel gain between the MD and an ED in epoch i [33]. This channel gain is static and independently taken from the state of MD.

$$DataRate_i = \omega * log_2\left(1 + \frac{s_i^e * f_i^{tr}}{A}\right) \tag{2}$$

The transmission power is calculated by Eq. (3) where $BW_i$ is the bandwidth of ED in epoch i and d is the transmission data size required for offloading a module.

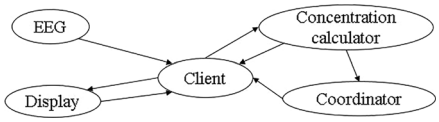


**Fig. 2** Application of virtual reality game

**Table 2** Symbol definition

| Symbol | Definition |
|---|---|
| μ | CPU cycles for processing the task |
| $g_i$ | Number of energy units |
| ω | Commonly adopted effective switched capacitance |
| $s^e_i$ | Channel gain between the MD and an ED in epoch i |
| $f^{tr}_i$ | Transmission power |
| d | Transmission data size |
| $P_i$ | Power of ith ED |
| $U_i$ | Utilization of ith FD |
| $U^p_i$ | Utilization in the previous updates time |
| T1 | The time frame of datacenter |
| T2 | The time frame of the host |
| TD | Difference between current and last process time |
| $U_{MIPS}$ | Utilization of MIPS |
| TAM | Total allocated MIPS |
| $E_c$ | Current energy consumption |
| $T_n$ | Current time |
| $P_h$ | Host power in last utilization |
| N | Number of FDs |
| C | Execution cost |
| $S_c$ | System clock |
| $T_{lu}$ | Last utilization update time |
| $R_M$ | Rate per MIPS |
| ω | Number of processors in a host |
| $U_l$ | Last utilization |
| $MIPS^A_k$ | Allocated MIPS of kth processor in the host |
| m1 | Number of all processors and allocated processors |
| m2 | Number of allocated processors |
| $T_{ED}$ | ED's execution time |
| $MIPS_k$ | MIPS of kth processor in the host |
| TMA | Total allocated MIPS of the host |
| $L_i$ | Total latency |
| $S_i$ | The total size of ith tuple |
| N′ | Total number of tuples |
| $T_{max}$ | Maximum simulation time |
| $T_{st}$ | Tuple start time |
| $T_a$ | Average CPU time of the tuple type |
| ET | Emitting time of a tuple |
| $T_s$ | Sending time of a module to another module |
| $N_{RT}$ | Number of receipt tuple types |
| Q | The number of devices contributed to the offloading |
| $x_j$ | Energy consumption of jth device |
| A | Selected action by the agent |
| R | Reward value |
| α | Learning rate |
| γ | Importance of the next rewards |
| $Q(S,a)^i_m$ | Q update value |

$$f_i^{tr} = \frac{BW_i}{d} \tag{3}$$

### 3.9 Edge server's Power Consumption

The power consumption of each ED is computed by Eq. (4). According to this equation, an edge server with the highest power is considered as a candidate for an offloading destination.

$$P_i = P_i^c + T1 + T2 \tag{4}$$

In Eq. (4), $P^c_i$ is the current power of FD and T1 is the energy consumption of the data center in the current time, and T2 is calculated using Eq. (5).

$$T2 = U_i^p + \frac{U_i - U_i^p}{2} * TD \tag{5}$$

In Eq. (5), $U_i$ is the utilization of ith FD, $U_i^p$ is the utilization in the previous updates time, and TD is the time difference between the current time and the last process time. $U_i$ is also computed using Eq.(6).

$$U_i = \frac{U_{MIPS}}{TAM} \tag{6}$$

The total allocated MIPS of all processing elements is updated as Eq. (7).

$$TAM = \sum_{i=1}^{N} \sum_{j=1}^{M} PEM_{ij} \tag{7}$$

TAM is the total allocated MIPS of an ED that is less than or equal to that ED's MIPS (TAM ≤ EDMIPS).

$$U_i^p = \frac{U_{MIPS}^P}{TAM} \tag{8}$$

In Eqs. (6) and (8), $U_i^p$ is the utilization value in the previous time, $U_{MIPS}$ is the utilization of MIPS, and TAM is the total allocated MIPS.

### 3.10 Edge server's Execution Time

MIPS calculates the runtime of modules in edge servers. The number of commands that any edge server can handle given its current workload is considered its current capacity. Therefore, according to Eq. (9), each module can capture and run it at $T_{ED}$, where MIPS is the million executable operations that an edge server can run per second.

$$T_{ED} = \frac{1}{MIPS} \tag{9}$$

### 3.11 Edge server's Bandwidth

Each ED includes some hosts as follows.

$$\{Host_1, Host_2, \ldots, Host_n\} \in ED_i \tag{10}$$

The main properties of hosts are RAM, bandwidth, storage, and Pes, as shown in Eq. (11):

$$BW_{Lower} \leq \sum_{i=1}^{N} BW_i \leq BW_{Upper} \tag{11}$$

Where $BW_{Lower}$ is the lower bandwidth, and $BW_{Upper}$ is the upper bandwidth of each ED. $BW_i$ is the bandwidth of ith host. N is the number of hosts. The total bandwidth of all hosts in each ED is between $BW_{Lower}$ and $BW_{Upper}$.

## 4 The Proposed Approach

As stated above, our goal is to apply the concept of context knowledge to a multi-user mobile edge computing system. The proposed framework for the context-aware system can be described as follows. In this system, we have two types of variables: independent variables and dependent variables. Independent variables are all input variables that the system receives in the form of transactions and does not interfere with their calculation, such as the types of fields that surround the environment. Dependent variables are variables that are obtained by using independent variables as inputs to the proposed system. Delay and energy consumption are those variables. The following sections describe the various tasks in MAPE.

### 4.1 MAPE

The MAPE control loop consists of four phases the monitor, analyze, plan, and execute. We explain each phase as follows:

#### *4.1.1 Monitoring*

This section monitors and collects input modules, including the contexts. The inputs include all requests received from IoT devices and fields collected from the environment which enter modules. The request is received with a

unique identifier. This request can be either computation or data. In this phase, independent parameters such as QoS and SLA are also monitored and written to the knowledge database. The contexts include application, mobile device, sensors, network, and media.

- **Offloading contexts**: Request id, requester name, sensitivity type (resource-based or time-based), QoS, and SLA requirements. Based on context information, the QoS depends on data rate between MDs and EDs as Eq. 2, edge power consumption an Eq. 4, and edge server's execution time introduced in Section 3.6.4.
- **Application contexts**: Total executed modules, runtime, allocated memory, priorities of modules, and source type. According to Eq. (12), the ith module has got more priority than the jth module if the module is before the jth module in the application's workflow.

$$Priority_i \leq Priority_j \tag{12}$$

- **Mobile contexts**: Average frequency of CPU, average CPU usage, and battery level. The CPU usage depends on the MIPS of each CPU.
- **Sensor contexts**: Sensor id, location, latency, destination module, tuple type, and transmission time.
- **Network and media context**: Cellular communication and bandwidth mode, Wi-Fi communication mode, cellular connectivity signal, and Wi-Fi.
- **Resource contexts**: Resource state, identification, memory, and storage.

These fields are stored in our knowledge bank's context database to be used later in the computing offloading operations.

#### *4.1.2 Analysis*

This component deals with the processing of data collected from the monitor component. In this phase, QoS and SLA are considered. If a resource is assigned to a computing request, which results in a breach of service quality, the analyst must detect this and issue the necessary alert. The second phase of the loop performs such analyzes. This phase has a close relationship with the knowledge bank and is constantly exchanging information with it. The analysis phase's output contexts are resource id, offloading request-id, QoS, and SLA types.

#### *4.1.3 Planning*

This phase contains the decision module of our system. Using the information from the previous phases, this section makes the final decision on whether to offload or execute locally. The decision module includes cost estimation and context-aware decision algorithm and finds the best destination for offloading the requester modules to edge-server or cloud. We present two offloading methods as MUCAO and FLUCO, as following.

*MUCAO* The first method for finding the best destination for offloading is a heuristic technique. Algorithm 1 includes two sections as initialization and MAPE [7]. In the initialization section, firstly, mobile devices, cloud, and applications are created. Secondly, edge servers are built based on the number of departments and mobiles. Finally, the application is submitted to the edge server controller, and iFogsim is started. In the MAPE section, four phases execute continuously. In the monitoring phase, the context of modules, sensors, tuples, network interfaces, mobile devices, cloud, and edge servers are collected. In the analysis phase, the cost of execution in the local device, edge server, and cloud is calculated, and the network interface state is checked. In the planning phase, the availability of local devices, edge servers, and cloud are checked. Finally, the offloading decision is executed.

**Algorithm 1** MUCAO

```
Create Mobile devices, cloud, applications (Modules, Edges, Tuples, Workow).
for i = 1 to DepartmentMax do
  for j = 1 to MobileMax do
    Edge server (Node name, MIPS, Ram, Storage, upper BW, lower BW, busy power, and idle power).
  end for
end for
Submit applications.
Start iFogsim.
while Modules enter from MDs do
  Monitor:
    Collect context of modules, sensors, tuples, network interfaces, mobile devices, cloud, and edge servers.
  Analyze:
    Calculate cost of execution in the local device, edge server, and cloud.
    Check network interface state.
  Plan:
  if Available(Local device) then
    if Available(Cloud) then
      Decision = MinCost(Local device,Cloud).
      Break.
    else
      Decision = local device.
    end if
  else
    if Available(Wi-Fi) then
      if Available(Edge server) & Available(Cloud) then
        Decision = MinCost(Local,Edge,CLoud).
      end if
    end if
  end if
  Execute:
  Offload module based on the decision.
end while
Stop iFogsim.
```

*FLUCO* The second proposed approach is based on the DRL. The DRL approach aims to learn the optimal MEC offloading policy from past experience. We try to extend this method to the distributed system. Our offloading algorithm implements in the MDs. The EDs and cloud devices analyze the updated weights from MDs. Then, each MD can decide to offload tasks to the best devices for execution. Here, we define the module offloading by DRL's agent as a tuple:

$$Agent = (M, S, A, Q) \tag{13}$$

In Eq. (13), M is the set of modules' attributes for allocation by agent, S is the set of all environment states, A is the set of actions like local execution, FDs, or cloud, and Q is the quality function that learning algorithm can select the best destination for module execution by that. These parameters use for the agent's action and calculated by Eq. (14).

$$Q : X^i * A \rightarrow \mathbb{R} \tag{14}$$

Here, $x^i$ is based on Eq. (15), A is the selected action by the agent, and R is the reward.

$$x^i = (z_1(x), z_2(x), \ldots, z_n(x)) \in X^z \tag{15}$$

As Eq. (16), each agent can explain modules and environments by Z.

$$Z = M \cup S = \{z_1, z_2, \ldots, z_n\} \tag{16}$$

The Q update function is defined by Eq. (17) where $\alpha \in (0, 1)$ is the learning rate, and $\gamma \in (0, 1)$ shows the importance of the next rewards.

$$Q(S,a)^i_m = Q(S,a)^i_m + \alpha\left[r + \gamma max_\alpha Q\left(S', a'\right)^i_m - Q(S,a)^i_m\right] \tag{17}$$

According to DRL, the maximum value of $Q(S, a)^i_m$ based on action $A$ is 1-$\epsilon$, and other actions have $\epsilon$ probability. Using a greedy policy technique is to avoid local optimum in the learning algorithms [43]. A reward function evaluates agent operations that should generate output very quickly so that learning and problem solving can be done without delay. The reward function in the proposed approach is calculated by Eq. (18).

$$Reward = \frac{P_i}{T_i} \tag{18}$$

$P_i$ is the available power of ith FD and $T_i$ is the execution time of an FD module. Since power and execution time values are in different ranges, a logarithmic function is used to normalize them in [0,1]. Thus $P_i = \frac{\log P_i}{10}$ *and* $T_i = \frac{\log T_i}{10}$.

*DRL Algorithm* The pseudo-code of the DRL is shown in Algorithm 2. The learning algorithm is executed for all modules. Then, the Q-table is initialized by 0. For all episodes, the possible actions and Q values of them are calculated, and the best action is selected by arg max Q. State S′ is transferred to state S. Here, the best destination for each module is selected. After calculating the reward function, updating the operation of Q and saving episodes in memory is done.

**Algorithm 2** DRL

```
for m = 1 to LastModuleInQueue do
    Initialize Q(S,a).
    for i = 1 to Episode_Last do
        Set S to S0.
        for j = 1 to S_Last do
            Select best a by calculation arg max Q.
            Action a, visit r and S'.
            Calculate Q(S, a)^i_m as Eq. (17).
            Transfer S to S'.
        end for
    end for
    Select a destination device for each module.
    Calculate real-time reward.
    Save S, r, S', and α in memory.
    Train Q policy by training samples.
    Update Q.
    Save the current episode in memory.
end for
```

Based on our approach, the DRL does not execute in EDs for three reasons.

1) The jeopardize of accessing personal data of MDs.
2) Encryption algorithms can protect data, but communication with the EDs weakens MDs' privacy.
3) Transferring a lot of data from MDs to EDs causes a lot of bandwidth consumption and burden wireless channels.

Furthermore, if the DRL agent runs on each MD, it will consume a lot of energy and time. For solving this problem, our proposed method is not based on the separate learning of each MD, and we use the distribution capability of the MEC. In fact, we propose FL for distributed training DRL agents. As a result, we can save a lot of energy.

*FLUCO Algorithm* In FL-based offloading, each ED is a controller to coordinate some MDs. Each MD can execute a DRL agent with less computation burden. FL does three steps:

1) Send the DRL agent's parameters from the ED.
2) MDs use to download data from EDs for upgrading their model.
3) Send updated DRL agent's parameters from MDs to ED (model aggregation). FL works in a parallel manner that increases the performance of the system. To design an optimal control policy on FL, we have to maximize the expected long-term utility as Eq. (19).

$$G(E,U) = \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} g(E_i, U(E_i)|E_1 = E) \qquad (19)$$

E is the network size, U is the system utility, $E_i$ is the initial network size, g(0) is the immediate utility at epoch i that is calculated based on the reward function in DRL. Based on the mentioned approach as Algorithm 2, DRL agents execute in MDs; training is performed, local execution or offloading to best ED are decided. Finally, trained weights are uploaded to EDs. EDs do not execute DRL agents and only update and aggregate their weights and send them to MDs. The weights aggregate by Eq. (20).

$$W_{r+1} = \sum_{k=1}^{Set_{Last}} \left( \frac{C_r^i}{C_r} * W_{r+1}^i \right) \qquad (20)$$

$Set_{Last}$ is the last set of available MDs, $C^i_r$ is the context of ith MD in round r, and $W^i{}_{r+1}$ is the weight ith module in the next round. The computation task is executed in MDs or offloaded to the best ED based on the DRL agent result. We propose multi ED in the MEC and update all EDs according to MDs' downloaded information.

**Algorithm 3** FLUCO

```
Create Mobile devices, cloud, application (Modules, EDs, Tuples, Workflow).
for x = 1 to Department_Max do
    for y = 1 to Mobile_Max do
        Create ED (Node name, MIPS, Ram, Storage, upper BW, lower BW, busy power, and idle power).
        Initialize the DRL agent with random weights W0 in the current ED.
        Initialize the gross training times T0.
    end for
end for
for M = 1 to MDs_Last do
    Initialize the contexts C^M_0
    Initialize the DRL W^M_0
    Download W0 from the closest ED.
    W^M_0 = W_0.
end for
Submit applications.
Start iFogsim.
while Modules enter from MDs do
    for r = 1 to r_Last do
        Monitor:
        Collect the context of modules, sensors, tuples, network interfaces, mobile devices, cloud, and edge servers.
        Analyze:
        Set_r = random set of available MDs.
        for i = 1 to Set_Last do
        Fetch W_r from ED as W^i_r = W_r.
        Update context C^i_rounr.
        Plan:
        Train the DRL agent with W^i_r on C_i.
        Upload trained W^i_r+1 to the closest ED.
        Notify the ED the time's T_r of local training.
        end for
        for j = 1 to ED_Last do
            Receive all model updates.
            Update T^j_r .
            Aggregate by Eq. (20).
        end for
    end for
    Execute:
    Offload modules based on the FDL result.
end while
Stop iFogsim.
```

The outputs of the planning phase are offloading request-id, resource type, offloading destination, and considered media for the relationship with a resource.

### 4.1.4 Execution

The execution phase is responsible for executing the offloading decisions. This component is closely related to the equipment and resources and stores the latest state of the resources previously mentioned in the knowledge bank for future use. The task manager collects information such as (method entries, libraries needed to execute the task, the network address of the download location) and puts it into an offloading package. The manager decides to run the task locally or sends it to the top layers as edge servers or cloud. The outputs of the planning phase are offloading request-id, resource type, offloading destination, and considered media for relation with the resource.

## 5 Evaluation

The performance of the proposed methods is presented in this section. The simulation environment in this research is the iFogsim library [41]. This simulator has got classes to implement resource management strategies. We have extended some classes as the ModulePlacementEdgeward for offloading and controller for more output metrics. Also, the VRGAMEFOG class is customized based on the architecture of this paper. We simulate the proposed algorithm and compare the results with other offloading methods as follows.

- **Original**: In this method, the computations in MDs execute locally. Thus, the computations do not offload to edge servers or the cloud. Since MDs continuously execute the tasks, they might not have enough resources. As a result, some tasks wait in a queue of resources, and the delays are increased.
- **Offload**: In this technique, the destination of tasks or modules is calculated based on the order of edge servers or cloud in the network [44]. Context-aware is not considered in this method. Here, all devices are in a list, and the controller assigns those modules that need resources to the elements of this list in order. This method is not optimal due to ignoring the properties of devices, applications, and network environment. Maybe a device at the top of the list is selected as the offloading destination, while some devices in the middle or last of the list are the best destination for offloading modules.
- **MUCAO**: This algorithm is presented in [7]. As discussed in the proposed approach section, this method is based on a MAPE loop and uses the execution cost in MDs, EDs, and cloud. Considering context awareness of devices, applications, and network environment leads to find the best device for offloading modules.
- **FLO**: As a state-of-the-art algorithm, FLO is an FL-based algorithm based on DRL [33]. Using one ED converts the computing architecture to the cloud. If the number of modules that need resources increase in MDs, just one ED might not answer all offloading requests. As a result, some modules wait for a long time. There are two differences between this work and our proposed algorithm. FLUCO uses many numbers of Eds, and contexts are considered for offloading decision making.

We run the simulation by three departments and four mobile devices. The comparisons are based on the best results of algorithms with the same configuration for each case study.

### 5.1 Simulation Configuration

Here, we present the VRGAMEFOG application configuration in edges, devices, connection latency, and hosts in Tables 3, 4, and 5, respectively. In Table 3, Pr is the periodicity (mS) of edges.

The host configuration is as follows. The architecture is ×86, OS is Linux, Storage is $10^6$B, BW is $10^4$ BS, VM model is Xen, the cost is 3 $, cost per memory is 0.05 $, cost per storage is 0.01 $, and time zone is 10. Table 4 shows the parameters of devices including MIPS, RAM (KB), UpBW (Upper bandwidth by kilobyte per second), DownBW (Down bandwidth by kilobyte per second), level in the hierarchical topology, the rate per MIPS, busy, and idle power (Megawatt).

Table 6 shows three different mobile types that have been used in this work. Type A is an Apple iPhone 11, type B is a Samsung Galaxy S10, and type C is a Huawei P30 pro. Their properties include CPU, memory, and battery. In this table, MT is a mobile type.

### 5.2 Metrics

We consider some metrics such as energy consumption, total execution cost, total network usage, delay, and Jain index to analyze our proposed approach and compare it with other offloading algorithms.

#### *5.2.1 Energy Consumption*

The energy consumption is calculated by Eq. (21) for all edge servers and cloud when they have serviced the input modules.

$$E = E_c + (T_n - T_{lu}) * P_h \tag{21}$$

We calculate the edge server's energy consumption by the power of all hosts in a certain time frame of execution. Where $E_c$ is energy consumption in the current state, Tn is the current time, $T_{lu}$ is the update time at the last utilization, and $P_h$ is the host's power in the last utilization. To calculate the total energy consumption, we have to sum all edge servers and the cloud's energy.

**Table 3** VR game application edge configuration

| Source Module | Destination Module | Pr | Tuple CPU length (B) | Tuple new length (B) |
|---|---|---|---|---|
| EEG | Client | 0 | 3000 | 500 |
| Client | Concentration Calculator | 0 | 3500 | 500 |
| Concentration Calculator | Coordinator | 100 | 1000 | 1000 |
| Concentration Calculator | Client | 0 | 14 | 500 |
| Coordinator | Client | 100 | 28 | 1000 |
| client | Display | 0 | 1000 | 500 |

### 5.2.2 Total Execution Cost

To obtain the execution cost, we calculate the total MIPS of hosts by the time frame. The time frame is calculated by the difference between the current time of simulation and the last utilization time.

$$Cost = \sum_{i=1}^{N} \left[ C + (SC - T_{lu}) * R_M * U_l * \sum_{k=1}^{\omega} MIPS_k \right] \quad (22)$$

In Eq. (22), N is the number of edge servers, C is the execution cost, SC is the system clock or current time of simulation, $T_{lu}$ is update time at the last utilization, $R_M$ is the rate per MIPS that is different for each inter-module edges, and TM is the total MIPS of the host. $U_l$ is the last utilization ($U_l$) that is calculated by Eq. (23). Where $MIPS^A_K$ and $MIPS_K$ are the allocated MIPS and MIPS of the kth processor in the host, and m1 and m2 are the number of all processors and allocated processors in a host, respectively.

$$U_l = Min\left(1, \frac{\sum_{k=1}^{m2} MIPS_k^A}{\sum_{k=1}^{m1} MIPS_k}\right) \quad (23)$$

### 5.2.3 Total Network Usage

Since tuples define the relationships between modules, thus resources' usages depend on the transferred tuples' size at a certain time. Total network usage is based on Eq. (24).

$$TNU = \frac{\sum_{i=1}^{N'} (L_i * S_i)}{T_{max}} \quad (24)$$

In Eq. (24), $L_i$ and $S_i$ are the latency and size of ith tuple overall, N′ is the total number of tuples, and $T_{max}$ is the maximum simulation time.

### 5.2.4 Application Delay

The delay of application execution is calculated by the system clock and the end time of a tuple.

$$T_{TN} = \begin{cases} SC - T_{st} & if\ T_a = 0, \\ \dfrac{T_{st} * N_{ET} + (SC - T_{st})}{N_{ET} + 1} & if\ T_a \geq 0 \end{cases} \quad (25)$$

The end time of the tuple is calculated by Eq. (25). Where $T_{st}$ is the tuple start time, SC is the system clock, ($SC-T_{st}$) is the execution time, and $N_{ET}$ is the number of executed tuple types. $T_a$ is the average CPU time based on the tuple type. CC is the system clock, and ET is the emitting time of a tuple. $T_s$ is transfer time between two modules.

**Table 4** Devices configuration

| Device | MIPS | Ram | Uplink BW | Downlink BW | Lv | Rate per MIPS | Busy Power | Idle Power |
|---|---|---|---|---|---|---|---|---|
| Cloud | 44,800 | 40,000 | 100 | 10,000 | 0 | 0.01 | 1648 | 1332 |
| Controller | 2800 | 4000 | 10,000 | 10,000 | 1 | 0 | 107,339 | 834,333 |
| $ED_s$ | 2800 | 4000 | 10,000 | 10,000 | 2 | 0 | 107,339 | 834,333 |
| MDss | 500 | 1000 | 10,000 | 10,000 | 3 | 0 | 8753 | 8244 |

**Table 5** Connection latency

| Device Name | Device Name | Latency (mS) |
|---|---|---|
| Cloud | Proxy – Server | 100 |
| Proxy – Server | Department (Gateway) | 4 |
| Department (Gateway) | Mobiles | 4 |
| EEG sensor | Mobile | 6 |
| Display | EEG sensor | 1 |

$$T_{TR} = \frac{T_{st} * N_{RT} + SC - T_s}{N_{ET} + 1} \quad (26)$$

The tuple receipt time is based on Eq. (26). $N_{RT}$ is the number of receipt tuple types. Application delay is calculated by the difference time between tuple end time in a module and tuple receipt time in another module.

### 5.3 Fairness

We evaluate the fairness of the offloading method based on the Jain index [32], which is computed by Eq. (27):

$$JainIndex = \frac{\left(\sum_{j=1}^{Q} x_j\right)^2}{Q * \sum_{j=1}^{Q} x_j^2} \quad (27)$$

Q is the number of devices that contributed to the offloading, and $x_j$ is the jth device's energy consumption. The Jain index is between $\frac{1}{Q}$ and 1; a better offloading method has a more Jain index.

### 5.4 Comparison Scenarios

Table 7 shows four different scenarios to analyze the proposed approach and other algorithms. Scenario 1 is based on a different number of users. Scenario 2 is considered for four different module sizes. Scenario 3 is for comparing the methods based on four mobile types. Also, we compare our proposed approach with others based on different intervals. The reason for using the values introduced in the diagram is according to the type of application. Since this application is introduced in the iFogsim emulator, so it comes with values by default. We tried to consider less and more of these parameters to get a reasonable estimate in terms of scalability, number and type of mobile devices, and module size.

**Table 6** Mobile types

| | Brand | CPU (GHz) | RAM (MB) | Battery (mA) |
|---|---|---|---|---|
| A | Apple iPhone 11 | 6*2.96 | 4000 | 3110 |
| B | Samsung Galaxy S10 | 8*2.30 | 8000 | 3400 |
| C | Huawei P30 pro | 8*2.03 | 8000 | 4200 |

**Table 7** Comparison scenarios

| No. | Description | Values |
|---|---|---|
| Scenario 1 | Number of users | 1, 3, 7, 10 |
| Scenario 2 | Module size (MB) | 1000, 2000, 5000, 10,000 |
| Scenario 3 | Mobile types | AB, AC, BC, ABC |
| Scenario 4 | Interval (ms) | 100, 200, 500, 1000 |

### 5.5 Scenario 1: Comparison of Offloading Performance Based on the Number of Users

One of the parameters to show the performance of the offloading methods is the number of users. Here, we compare the energy consumption, total execution cost, network usage, and delay of MUCAO in MEC by the number of users. As can be seen in Figs. 3, 4, 5, 6, and 7, there are values of the number of users by 1, 3, 7, and 10 in the horizontal axis.

Fog devices serve multiple users simultaneously. On the other hand, given the number of resources these devices have, when the number of users reaches a certain size, they reach a degree of optimization. This means that devices can perform resource management operations more efficiently. Due to the hierarchical structure of the network and users' distribution, it will

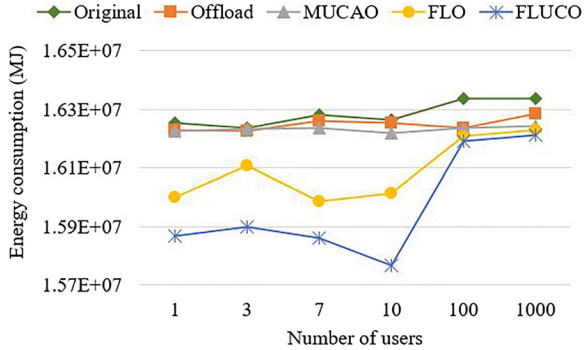


**Fig. 3** Energy consumption based on the number of users

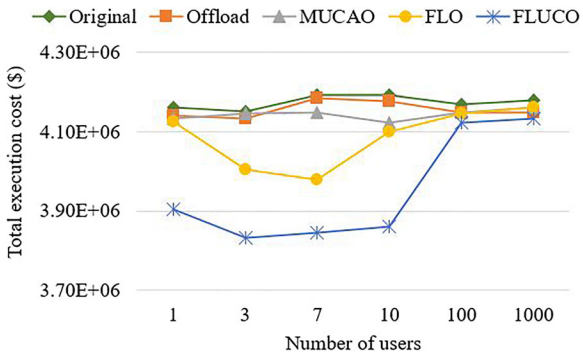


**Fig. 4** Total execution cost based on the number of users

be possible to improve the results even with the increase of users, which can be seen in the results.

### *5.5.1 Energy Consumption Based on the Number of Users*

Figure 3 shows that the energy consumption of MUCAO is less than the original and offload methods. MUCAO can decrease energy consumption in a higher number of users. As this figure, the maximum energy consumption is on the number of users by 7 for the original method by $1.635 * 10^7$ MJ. The minimum value is on the number of users by 10 for the FLUCO method by $1.58 * 10^7$ MJ. This result shows that the FL-based method with distributed structure causes less energy consumption than others. Also, adding context-awareness information to FLO and using more than one ED cause to create a better method as FLUCO for energy efficiency. The reason for the improvement is a distributed algorithm of FLUCO that executes in multi EDs. The FLUCO causes MDs with lower resources to transfer more of their modules to EDs. As a result, the workload in the network has been distributed in a balanced way. There is not much difference between the

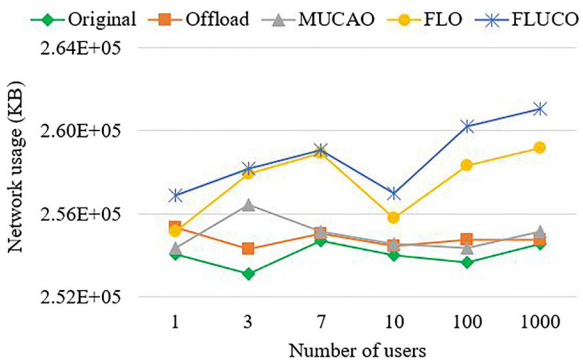


**Fig. 5** Network usage based on the number of users

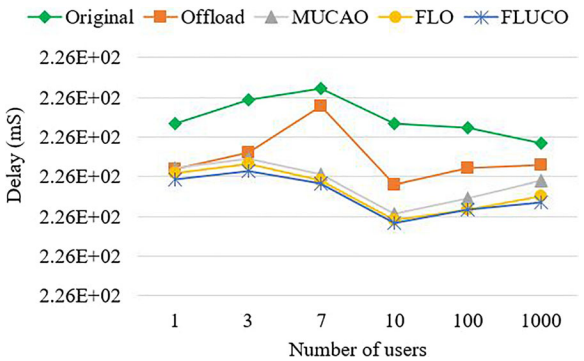


**Fig. 6** Delay based on the number of users

energy consumption of methods with increasing the number of users. Since FLUCO distributes modules in the network, and also the capacity of devices is restricted. Thus the number of modules in devices cannot increase. Finally, we have not got more computations to calculate energy consumption for them.

### *5.5.2 Total Execution Cost Based on the Number of Users*

The cost is one of the important metrics in this work. We can see in Fig. 4 with increasing the number of users from 3 to 7, and 10 causes increasing cost in original and offload methods. MUCAO has a balanced cost than these methods in many users with fluctuating between $4.12 * 10^6$ $ and $4.15 * 10^6$ $. Additionally, FLUCO with minimum energy consumption less than FLO and MUCAO is placed in the first rank of total execution cost. This result shows that with the increase in the number of users and distribution in the environment, the FLUCO has managed the cost well and brings economic savings. The main reason for this improvement is our distributed algorithm in some EDs that cause

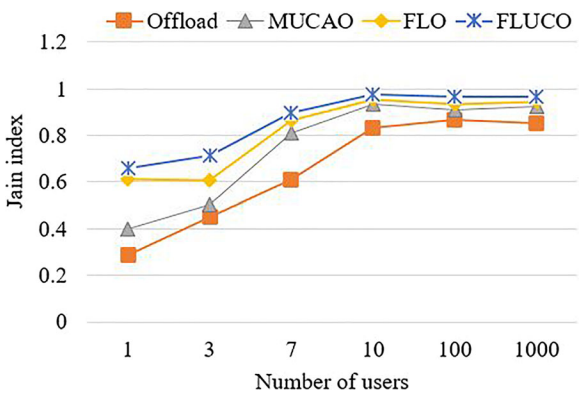


**Fig. 7** Jain index based on the number of users

choosing the best device with high performance and minimum delay.

### 5.5.3 Network Usage Based on the Number of Users

Figure 5 shows that MUCA can increase network usage by considering context-awareness. Increasing the number of users could not increase the network usage so much. The reason for this result is the best matching of offload destination instead of first matching. MUCAO has used network resources better than other methods. Since the main idea of the porposed method is distribution, the modules can be offloaded to a wide range of devices. That is why all devices in the network are almost busy with minimum free time.

### 5.5.4 Delay Based on the Number of Users

The delay of the application loop by MUCAO has a slight decrease, and its delay is lower than original and offload methods. According to Fig. 6, the maximum delay is related to the original method and the minimum value belongs to FLUCO, which shows that using context information and distributed algorithms cause to fast executing of requests and offloading process. This means that MDs using FLUCO can quickly find the best destination to offload their modules and save more time.

### 5.5.5 Jain Index Based on the Number of Users

We provide the fairness of offloading algorithms by Jain index value in Fig. 7. Since the original method has not any offloading thus, we compare others. As we mentioned, this metric uses the energy consumption and the number of edge servers that contribute to offloading. The maximum Jain index is related to FLUCO in the number of users by 10. This shows in FLUCO; more edge servers are used to offload modules. Also, this result proves better load balancing in the FLUCO than others.

## 5.6 Scenario 2: Comparison of Offloading Performance Based on Module Size

The module size is another metric for evaluating the offloading methods in this work. Based on Figs. 8, 9, 10, 11, and 12, there are module size's values by 1000, 2000, 5000, and 100,000 in the horizontal axis.

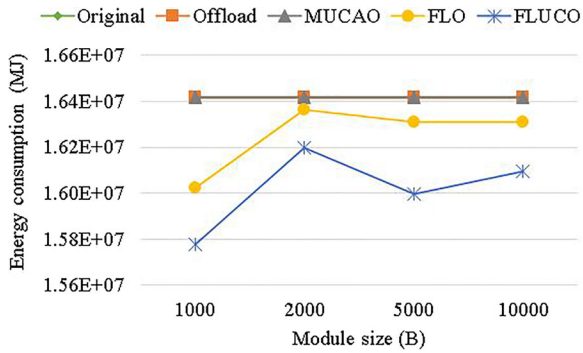


**Fig. 8** Energy consumption based on module size

### 5.6.1 Energy Consumption Based on Module Size

According to Fig. 8, the module size does not have a significant impact on energy consumption. The original, offload, and MUCAO show almost equal energy consumption. On the other hand, FLUCO and FLO have got better results. This means using distributed structure and context awareness can improve the energy consumption of the system. Since EDs and cloud resources have more capacities than MDs, different modules can be offloaded and executed to the network's upper layer. Also, regarding more devices and widely distributed modules, the module's size has not got a considerable change in energy consumption.

### 5.6.2 Total Execution Cost Based on Module Size

Fi. 9 shows the total execution cost for different approaches using different module sizes. The context-awareness of application, devices, and network environment allows MDs to have a better offloading decision, and thus FLUCO presents the lowest execution cost compared to other methods.

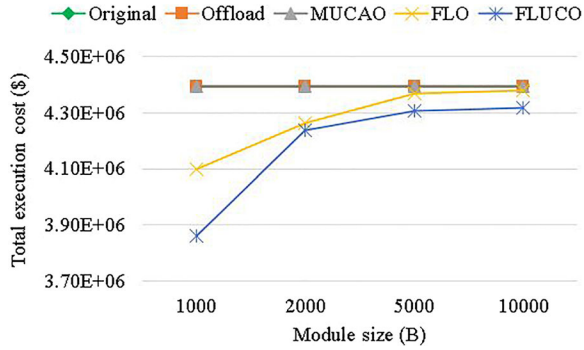


**Fig. 9** Total execution cost based on module size

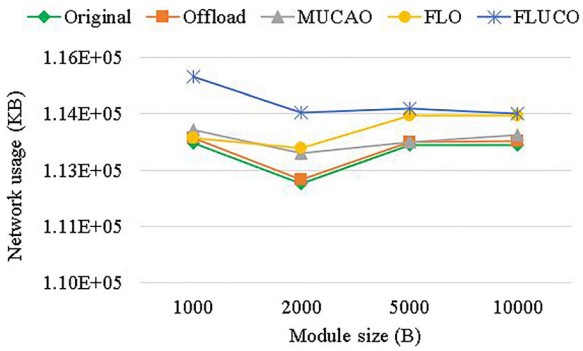


**Fig. 10** Network usage based on module size

### 5.6.3 Network Usage Based on Module Size

Figure 10 shows the network usage of different approaches under various module sizes. The minimum network usage is belonged to the original method with a module size of 2000 B by $1.125 * 10^5$. Increasing the module size causes raising total network usage, but this happens until a specific module size because EDs and cloud capacities are more than MDs.

### 5.6.4 Delay Based on Module Size

As shown in Fig. 11, increasing the module size causes a decrease in the delay for all methods. The reason for decreasing delay in the original method is the local execution of modules. Also, we should consider some of the modules might not execute locally for not being enough resources. Of course, the energy consumption of MDs will be increased. Since edge servers have more capacity than module sizes, they can execute offloaded modules in less time. Also, FLUCO has better results than others in module sizes by 2000 and 5000 B. In module size 1000 and 10,000 B, FLUCO has got a little improvement than FLO. This shows that a distributed

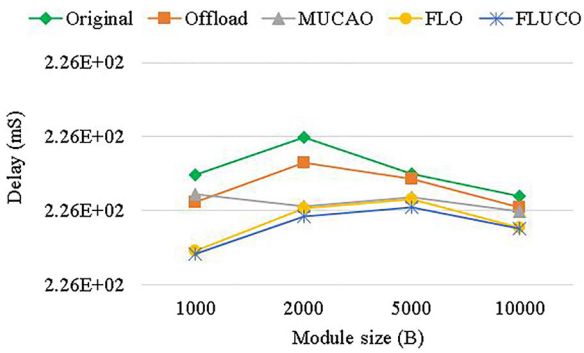


**Fig. 11** Delay based on module size

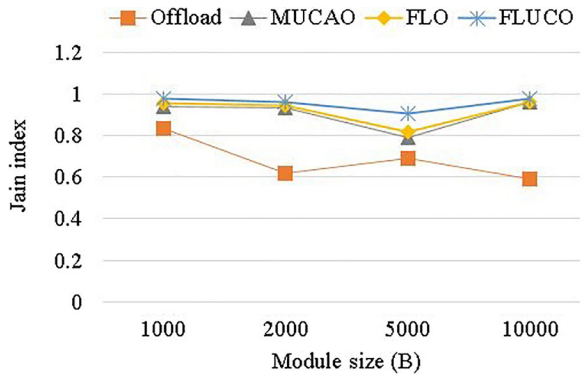


**Fig. 12** Jain index based on module size

algorithm can manage and offload them to the best devices when module size increases.

### 5.6.5 Jain Index Based on Module Size

In Fig. 12, the fairness of the offloading method is between 0.6 and 0.8. However, the range of this metric in MUCAO is between 0.8 and 1.0. This proves that MUCAO with considering context information is fairer than the offloading method. On the other hand, using distributes algorithms and more EDs convert FLUCO to the best algorithm. This means the energy consumption of all devices in the network was in a distributed manner. Also, decreasing the delay of modules cause all devices to consume less energy, so that the proposed approaches are better in this case.

## 5.7 Scenario 3: Comparison of Offloading Performance Based on Mobile Types

Figures 13, 14, 15, 16, 17 show the obtained results based on different mobile types, presented in Table 6.

### 5.7.1 Energy Consumption Based on Mobile Types

Analysis of energy consumption based on mobile types shows better results of FLUCO than others in all states AB, AC, BC, ABC. The results in Fig. 13 prove that the diversity of mobiles can cause less energy consumption by the proposed method. FLUCO, with minimum energy consumption of about $1.58*10^7$ MJ, is the best method than others. This method shows that a distributed algorithm in different devices can offload modules with less energy consumption. Also, heterogeneous devices with different configurations have got required resources for local computation.

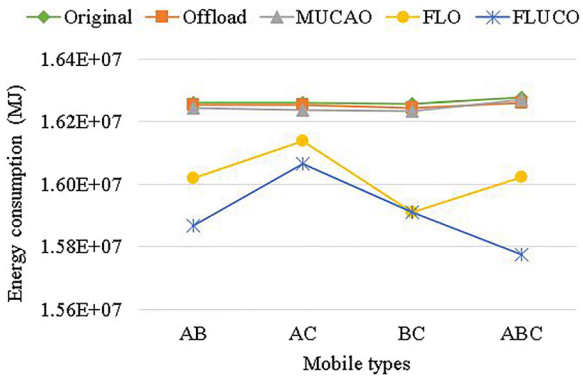


**Fig. 13** Energy consumption based on mobile types

### 5.7.2 Total Execution Cost Based on Mobile Types

According to Fig. 14, using different mobile types decreases total cost in all methods. FLUCO has better results than others. More capacity of CPU, memory, and battery causes mobile devices to execute more modules locally. This process decreases the offloading cost. Another reason for the improvement of the FLUCO can be fair offloading in a wide range of devices. Thus, decreasing the cost of a device and distributing fair offloading to other devices can lower the costs.

### 5.7.3 Network Usage Based on Module Types

Based on Fig. 15, network usage has a gradual increase by different mobile types. We can see in this figure that the FLUCO has maximum network usage in mobile type BC by 2.6 * $10^5$ KB. Thus, diversity in mobile types has a direct effect on network usage. Using a distributed structure of the network causes more network usage in the MEC. This means the FLUCO with a distributed method can use many devices in the network, and the number of jobless devices will decrease.

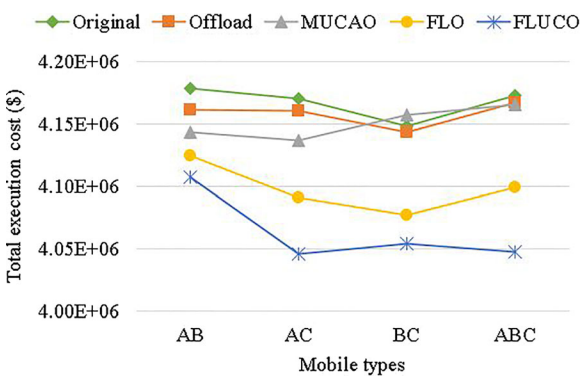


**Fig. 14** Total execution cost based on module size

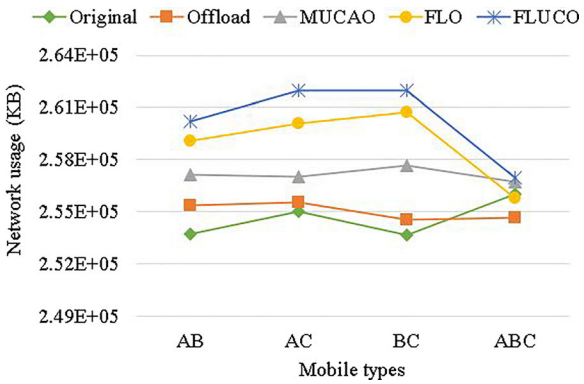


**Fig. 15** Network usage based on module types

### 5.7.4 Delay Based on Module Types

Figure 16 shows that delay in all mobile types AB, AC, and BC has sensitive changes. FLUCO has a minimum delay equal to 226.2 mS on mobile type ABC. The results prove that the increase in the diversity of MDs causes more challenges in delay. The distributed algorithms as FLO and FLUCO can do better than others. FLUCO has got less delay when the mobile type is AC. The reason for this improvement can be the context awareness in FLUCO than FLO.

### 5.7.5 Jain Index Based on Module Types

The fairness metric shows that FLUCO has the best results in all mobile types. Also, Fig. 17 proves the minimum fairness is related to the offloading method in mobile type AB by 0.6. Thus, distributed multi-user context-aware is a suitable offloading method in MEC. If the offloading method can place the modules on a wide range of devices, we will have a fairway.

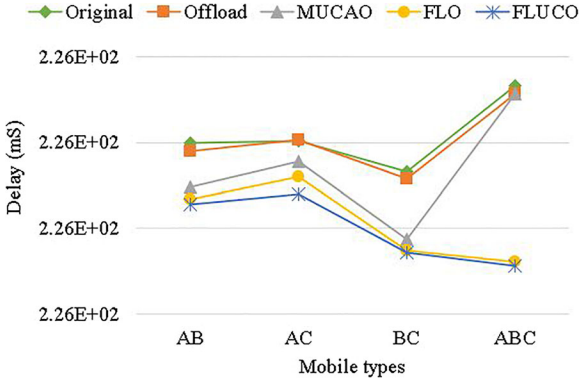


**Fig. 16** Delay based on module types

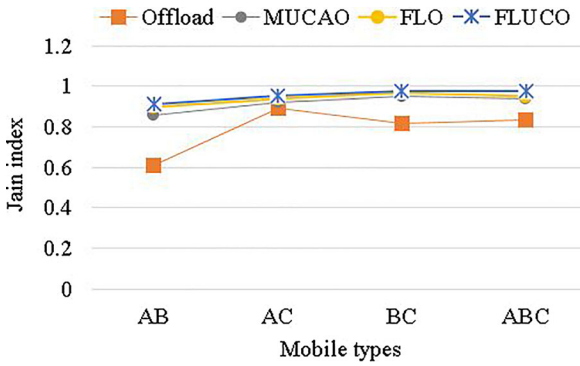


**Fig. 17** Jain index based on module types

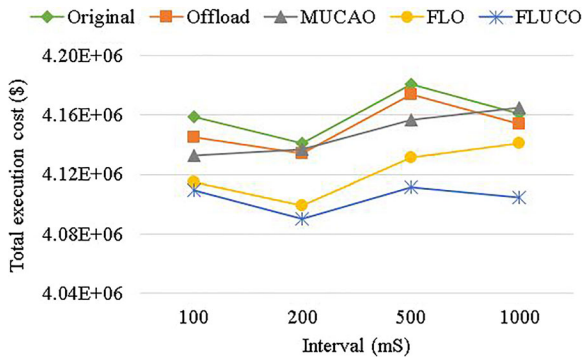


**Fig. 19** Total execution cost vs. interval

## 5.8 Scenario 4: Comparison of Offloading Performance Based on Interval

Offloading's interval shows the time distance between the resource management process. We control the workflow to the system by interval value. The energy consumption, total execution cost, network usage, delay, and Jain index are evaluated by offloading's intervals equal to 100, 200, 500, and 1000 mS. We set these values for the spacing between the input data goes back to the type of application. Since the application is intended to process input data in an average of 200 mS. Therefore, we consider numbers in the same range.

### *5.8.1 Energy Consumption Based on Interval*

Figure 18 proves that our proposed MUCAO and FLUCO methods can decrease energy consumption than the original, offload, and FLO methods. FLUCO with $1.57 * 10^7$ MJ is the best than others. Thus, FLUCO is very suitable for offloading in the VRGAMEFOG application. Since the interval value means the time distance between the resource management process, increasing that causes the offloading method will have more time for processing or offloading modules to best devices. As a result, we can see; generally, the FLUCO used this chance better than others.

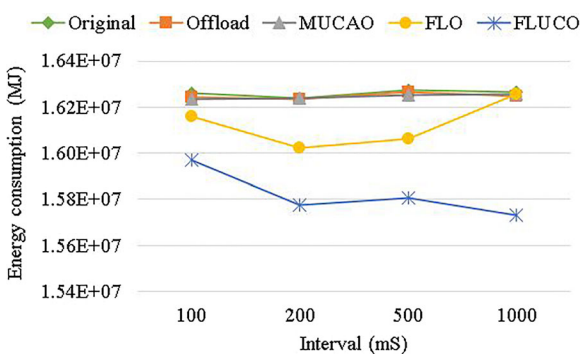


**Fig. 18** Energy consumption vs. interval

### *5.8.2 Total Execution Cost Based on the Interval*

Analysis of execution cost by all offloading methods shows that interval equals 500 causes more energy consumption. According to Fig. 19, the minimum execution cost of $4.9 * 10^6$$ is related to FLUCO by interval 200 mS. The worst execution is related to the original method by $4.18 * 10^6$$ in the interval of 500 mS without any offloading. The results prove the superiority of the distributed algorithm over others in MEC. The fair using of resources in devices causes less cost so that FLUCO can have better results than others with distribution and context awareness capability.

### *5.8.3 Network Usage Based on Interval*

The results of the simulation show the competition between all methods. They cause relative network usage values in the interval by 100, 500, and 1000 mS. Figure 20 indicates, in the average stats, FLUCO is the

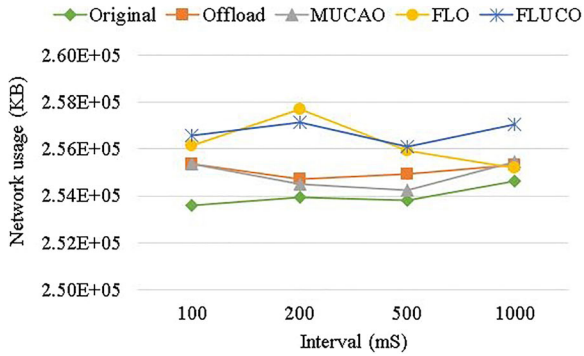


**Fig. 20** Network usage vs. interval

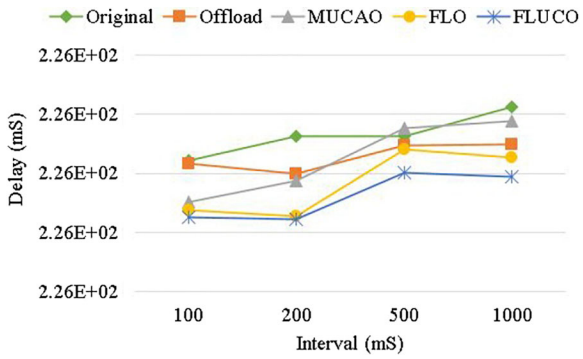


**Fig. 21** Delay vs. interval

best offloading method than others. The main reason for this improvement is the distributed structure of FLUCO, also using the properties of devices, application modules, and environment are essential in context-awareness.

### *5.8.4 Delay Based on the Interval*

Based on Fig. 21, the MUCAO method has a gradual increase in the delay parameter when the interval is grown. According to this figure, the original method has the worst result in the interval by 1000 mS. The offload method has a fluctuated result with the lowest in the interval of 200 mS and the highest in the interval by 500 mS. More analysis shows that FLO and FLUCO have got lower delays than others. The results show that these two methods can quickly offload the modules to the best devices with minimum delay. Furthermore, the lowest delay equal 226.3 mS is related to FLUCO in an interval of 200 mS.

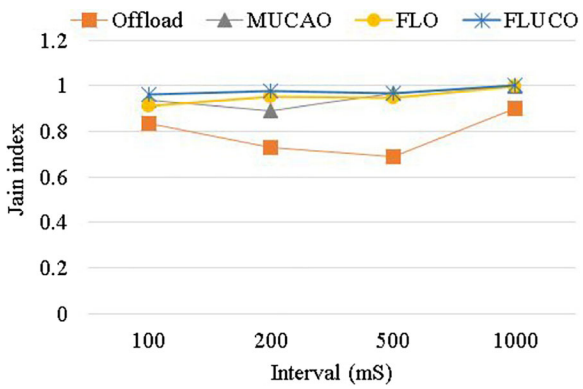


**Fig. 22** Delay vs. interval

### *5.8.5 Jain Index Based on Interval*

According to Fig. 22, MUCAO and FLUCO have a gradual increase in the Jain index with maximum fairness in the interval of 1000 mS. However, the offloading method has fluctuated values, and it could approximately close to MUCAO in 1000 mS. However, FLUCO with the highest Jain index is better than others. This proves that a distributed algorithm can be a fair offloading method. Thus, the dynamic context-awareness and distributed structure of the proposed algorithm can improve the performance of MEC.

## 6 Conclusion

In this paper, we investigated context-aware offloading by considering multi-user. We also present a distributed algorithm as FLUCO to got close to the MEC structure. To solve this problem, a MAPE loop is used in all offloading processes. Our method helps MDs to offload their modules to edge servers or cloud if they can not execute those locally with less cost. The results show that FLUCO is superior to original, offload, MUCAO, and FLO methods in energy consumption by 2%, 2%, 2.1%, and 0.7% in total execution cost by 3%, 3%, 2.34%, and 1.08% network usage by 2%, 2%, 1.21%, and 0.001% delay by 0.01%, 0.01%, 0.005%, and 0.001% and 0.002%, respectively. Also, FLUCO is fairer than offload, MUCAO, and FLO methods in the Jain index by 18%, 4.01%, and 1.6%, respectively. These results prove that our proposed offloading algorithm with context-aware information and distributed structure could improve the network performance in the mentioned metrics. As future work, we work on MEC with FL-based methods on other case studies. Cooperative mobile crowding is another challenge of FL in MEC for more research. FL is vulnerable to communication security issues such as Distributed Denial-of-Service (DoS) and jamming attacks. Also, we will study the protection of data privacy for MEC users.

**Data Availability** Data sharing not applicable to this article as no datasets were generated or analyzed during the current study.

## References


1. Paknejad, P., Khorsand, R., Ramezanpour, M.: Chaotic improved PICEA-g-based multi-objective optimization for workflow scheduling in cloud environment. Futur. Gener. Comput. Syst. **117**, 12–28 (2021)
2. Shahidinejad, A., Ghobaei-Arani, M., Masdari, M.: Resource provisioning using workload clustering in cloud computing environment: a hybrid approach. Clust. Comput. **24**(1), 319–342 (2021)
3. Shahidinejad, A., Ghobaei-Arani, M.: Joint computation offloading and resource provisioning for edge-cloud computing environment: a machine learning-based approach. Software: Practice and Experience. **50**(12), 2212–2230 (2020)
4. M. Ayoubi, M. Ramezanpour, and R. Khorsand, "An Autonomous IoT Service Placement Methodology in Fog Computing," Software*:* Practice and Experience*,* 2020
5. Wang, F., Xu, J., Cui, S.: Optimal energy allocation and task offloading policy for wireless powered mobile edge computing systems. IEEE Trans. Wirel. Commun. **19**(4), 2443–2459 (2020)
6. Hu, Y.C., Patel, M., Sabella, D., Sprecher, N., Young, V.: Mobile edge computing—a key technology towards 5G. ETSI white paper. **11**(11), 1–16 (2015)
7. Farahbakhsh, F., Shahidinejad, A., Ghobaei-Arani, M.: Context-aware computation offloading for mobile edge computing. J. Ambient. Intell. Humaniz. Comput. 1–13 (2021)
8. Aral, A., Brandic, I., Uriarte, R.B., De Nicola, R., Scoca, V.: Addressing application latency requirements through edge scheduling. Journal of Grid Computing. **17**(4), 677–698 (2019)
9. Farahbakhsh, F., Shahidinejad, A., Ghobaei-Arani, M.: Multiuser context-aware computation offloading in mobile edge computing based on Bayesian learning automata. *Transactions on Emerging Telecommunications Technologies*. **32**(1), e4127 (2021)
10. Liang, Z., Liu, Y., Lok, T.-M., Huang, K.: Multiuser computation offloading and downloading for edge computing with virtualization. IEEE Trans. Wirel. Commun. **18**(9), 4298–4311 (2019)
11. Luo, C., Goncalves, J., Velloso, E., Kostakos, V.: A survey of context simulation for testing mobile context-aware applications. ACM Computing Surveys (CSUR). **53**(1), 1–39 (2020)
12. Shakarami, A., Shahidinejad, A., Ghobaei-Arani, M.: An autonomous computation offloading strategy in Mobile edge Computing: a deep learning-based hybrid approach. J. Netw. Comput. Appl. **178**, 102974 (2021)
13. Lim, W.Y.B., Luong, N.C., Hoang, D.T., Jiao, Y., Liang, Y.C., Yang, Q., Niyato, D., Miao, C.: Federated learning in mobile edge networks: a comprehensive survey. IEEE Communications Surveys & Tutorials. **22**(3), 2031–2063 (2020)
14. Peng, H., Wen, W.-S., Tseng, M.-L., Li, L.-L.: Joint optimization method for task scheduling time and energy consumption in mobile cloud computing environment. Appl. Soft Comput. **80**, 534–545 (2019)
15. Yang, X., Fei, Z., Zheng, J., Zhang, N., Anpalagan, A.: Joint multi-user computation offloading and data caching for hybrid mobile cloud/edge computing. IEEE Trans. Veh. Technol. **68**(11), 11018–11030 (2019)
16. Z.-Z. Liu, Q. Z. Sheng, X. Xu, D. Chu, and W. E. Zhang, "Context-aware and adaptive QoS prediction for mobile edge computing services," IEEE Trans. Serv. Comput*.,* 2019
17. Tran, D.H., Tran, N.H., Pham, C., Kazmi, S.A., Huh, E.-N., Hong, C.S.: OaaS: offload as a service in fog networks. Computing. **99**(11), 1081–1104 (2017)
18. A. Shakarami, M. Ghobaei-Arani, and A. Shahidinejad, "A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective," *Computer Networks,* p. 107496, 2020
19. Z. Chang, Z. Zhou, T. Ristaniemi, and Z. Niu, "Energy efficient optimization for computation offloading in fog computing system," in *GLOBECOM 2017–2017 IEEE Global Communications Conference*, 2017, pp. 1–6: IEEE
20. Peng, K., et al.: An energy-and cost-aware computation offloading method for workflow applications in mobile edge computing. EURASIP J. Wirel. Commun. Netw. **2019**(1), 1–15 (2019)
21. Liu, L., Chang, Z., Guo, X., Mao, S., Ristaniemi, T.: Multiobjective optimization for computation offloading in fog computing. IEEE Internet Things J. **5**(1), 283–294 (2017)
22. Jararweh, Y., Al-Ayyoub, M., Al-Quraan, M., Lo'ai, A.T., Benkhelifa, E.: Delay-aware power optimization model for mobile edge computing systems. Pers. Ubiquit. Comput. **21**(6), 1067–1077 (2017)
23. L. Huang, X. Feng, L. Zhang, L. Qian, and Y. Wu, "Multi-server multi-user multi-task computation offloading for mobile edge computing networks," *Sensors,* vol. 19, no. 6, p. 1446, 2019
24. Salehan, A., Deldari, H., Abrishami, S.: An online context-aware mechanism for computation offloading in ubiquitous and mobile cloud environments. J. Supercomput. **75**(7), 3769–3809 (2019)
25. J. Cho, K. Sundaresan, R. Mahindra, J. Van der Merwe, and S. Rangarajan, "ACACIA: context-aware edge computing for continuous interactive applications over mobile networks," in *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, 2016, pp. 375–389
26. Chen, X., Chen, S., Zeng, X., Zheng, X., Zhang, Y., Rong, C.: Framework for context-aware computation offloading in mobile cloud computing. Journal of Cloud Computing. **6**(1), 1–17 (2017)
27. Ghasemi-Falavarjani, S., Nematbakhsh, M., Ghahfarokhi, B.S.: Context-aware multi-objective resource allocation in mobile cloud. Computers & Electrical Engineering. **44**, 218–240 (2015)
28. Nawrocki, P., Sniezynski, B.: Adaptive context-aware energy optimization for services on Mobile devices with use of machine learning. Wirel. Pers. Commun. **115**(3), 1839–1867 (2020)
29. R. Roostaei and Z. Movahedi, "Mobility-Aware and Fault-Tolerant Computation Offloading for Mobile Cloud Computing," 2018
30. S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel

execution in the cloud for mobile code offloading," in *2012 Proceedings IEEE Infocom*, 2012, pp. 945–953: IEEE
31. T.-Y. Lin, T.-A. Lin, C.-H. Hsu, and C.-T. King, "Context-aware decision engine for mobile cloud offloading," in *2013 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, 2013, pp. 111–116: IEEE
32. Shakarami, A., Shahidinejad, A., Ghobaei-Arani, M.: A review on the computation offloading approaches in mobile edge computing: a g ame-theoretic perspective. Software: Practice and Experience. **50**(9), 1719–1759 (2020)
33. Ren, J., Wang, H., Hou, T., Zheng, S., Tang, C.: Federated learning-based computation offloading optimization in edge computing-supported internet of things. IEEE Access. **7**, 69194–69201 (2019)
34. Yang, K., Jiang, T., Shi, Y., Ding, Z.: Federated learning via over-the-air computation. IEEE Trans. Wirel. Commun. **19**(3), 2022–2035 (2020)
35. Wang, X., Han, Y., Wang, C., Zhao, Q., Chen, X., Chen, M.: In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning. IEEE Netw. **33**(5), 156–165 (2019)
36. Shen, S., Han, Y., Wang, X., Wang, Y.: Computation offloading with multiple agents in edge-computing–supported IoT. ACM Transactions on Sensor Networks (TOSN). **16**(1), 1–27 (2019)
37. Boukerche, A., Guan, S., Grande, R.E.D.: Sustainable offloading in mobile cloud computing: algorithmic design and implementation. ACM Computing Surveys (CSUR). **52**(1), 1–37 (2019)
38. Nawrocki, P., Sniezynski, B.: Autonomous context-based service optimization in mobile cloud computing. Journal of Grid computing. **15**(3), 343–356 (2017)
39. Baraki, H., Jahl, A., Jakob, S., Schwarzbach, C., Fax, M., Geihs, K.: Optimizing applications for mobile cloud computing through MOCCAA. Journal of Grid Computing. **17**(4), 651–676 (2019)
40. Computing, A.: An architectural blueprint for autonomic computing. IBM White Paper. **31**(2006), 1–6 (2006)
41. Gupta, H., Vahid Dastjerdi, A., Ghosh, S.K., Buyya, R.: iFogSim: a toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. Software: Practice and Experience. **47**(9), 1275–1296 (2017)
42. Burd, T.D., Brodersen, R.W.: Processor design for portable systems. Journal of VLSI signal processing systems for signal, image and video technology. **13**(2), 203–221 (1996)
43. Sutton, R.S., Barto, A.G.: "Reinforcement Learning: an Introduction," Ed: Cambridge. MIT Press, MA (2011)
44. Tang, L., He, S.: Multi-user computation offloading in mobile edge computing: a behavioral perspective. IEEE Netw. **32**(1), 48–53 (2018)