*Computers, Materials & Continua*
DOI: 10.32604/cmc.2022.026370
***Article***

# Efficient Computation Offloading of IoT-Based Workflows Using Discrete Teaching Learning-Based Optimization

**Mohamed K. Hussein**[1,*] **and Mohamed H. Mousa**[1,2]

[1]Department of Computer Science, Faculty of Computers and Informatics, Suez Canal University, Ismailia, Egypt
[2]Department of Information Technology, College of Computer Science at AlKamil, University of Jeddah, Jeddah, Saudi Arabia
*Corresponding Author: Mohamed K. Hussein. Email: m_khamiss@ci.suez.edu.eg
Received: 24 December 2021; Accepted: 18 April 2022

**Abstract:** As the Internet of Things (IoT) and mobile devices have rapidly proliferated, their computationally intensive applications have developed into complex, concurrent IoT-based workflows involving multiple interdependent tasks. By exploiting its low latency and high bandwidth, mobile edge computing (MEC) has emerged to achieve the high-performance computation offloading of these applications to satisfy the quality-of-service requirements of workflows and devices. In this study, we propose an offloading strategy for IoT-based workflows in a high-performance MEC environment. The proposed task-based offloading strategy consists of an optimization problem that includes task dependency, communication costs, workflow constraints, device energy consumption, and the heterogeneous characteristics of the edge environment. In addition, the optimal placement of workflow tasks is optimized using a discrete teaching learning-based optimization (DTLBO) metaheuristic. Extensive experimental evaluations demonstrate that the proposed offloading strategy is effective at minimizing the energy consumption of mobile devices and reducing the execution times of workflows compared to offloading strategies using different metaheuristics, including particle swarm optimization and ant colony optimization.

**Keywords:** High-performance computing; internet of things (IoT); mobile edge computing (MEC); workflows; computation offloading; teaching learning-based optimization

## 1 Introduction

As the capabilities of the Internet of Things (IoT) and smart mobile devices (such as smartphones, smart cameras, sensors, and smart vehicles) have rapidly advanced, the prevalence of these devices has significantly increased. As a result, their applications, including vehicular network transportation systems, augmented reality, healthcare, smart buildings, and environmental systems, have become ubiquitous in every aspect of modern life [1]. However, with their development, these applications have become increasingly complex and therefore require considerable processing and storage capabilities.

Furthermore, IoT-based applications comprise multiple communication tasks performed on multiple IoT devices to achieve certain computational goals. Consequently, a typical advanced IoT application consists of complex distributed interdependent computing tasks that communicate during the runtime and are organized as a directed acyclic graph (DAG) to form an IoT-based workflow [2]. Additionally, because IoT devices suffer from limited resources (such CPU, memory, and energy limitations), the processing of such applications requires a high-performance computing environment to satisfy the quality-of-service (QoS) requirements of workflows and devices.

Cloud computing is a type of on-demand high-performance computing that provides large-scale distributed computing resources over the internet. These resources are provided in the form of virtual machines that are rented in accordance with certain needs at certain times as a high-performance computing environment for complex dependent tasks [3]. Cloud computing supports IoT workflows by allowing workflow tasks to be offloaded for execution on the cloud, which is called mobile cloud computing (MCC) [4]. Offloading the computation tasks of IoT workflows to the cloud reduces the execution time and the energy consumption of the IoT devices involved. However, IoT devices must access the cloud through a wide area network (WAN). As a result, workflows tend to suffer from high latencies and low bandwidths, which consequently affect the execution time of delay-sensitive IoT workflows. Therefore, mobile edge computing (MEC) has emerged as a means of providing computational support at the edge of an IoT network between the cloud and IoT devices [5]. Integrating the advantages of both edge computing and cloud computing improves the QoS requirements of IoT-based workflows in terms of delays by reducing latencies and increasing bandwidths because the edge servers are accessed via a local area network (LAN) [6]. Hence, an offloading strategy for IoT-based workflows in an MEC environment is required to improve the QoS parameters of IoT-based workflows and mobile devices. However, making decisions to offload a large number of tasks for IoT workflows with different QoS requirements either to edge servers (such as cloudlets) or to the cloud while considering workflow delays, task dependencies, the communication time between tasks, and the power consumption of IoT devices while simultaneously satisfying workflow deadline constraints is a challenging problem. Furthermore, because they have limited resources compared with the cloud, edge servers may become overloaded, which may lead to increases in workflow delays and the energy consumption of IoT devices [7]. As a result, the placement decisions for the offloaded tasks of IoT workflows in an MEC environment constitute an NP-hard problem [8].

### *1.1 Motivation*

The emergence of evolutionary and nature-inspired metaheuristic techniques has shifted high-performance computing research in a promising direction. Metaheuristic algorithms have proven effective in optimizing the energy within the IoT [9], cloud load balancing, security intrusion detection, and load balancing in IoT-fog computing [10]. Metaheuristics are stochastic iterative algorithms that imitate the process of evolution or the behavior of a group of animals to provide a near-optimal solution to a complex optimization problem in an acceptable time [11]. Nature-inspired metaheuristics such as swarm algorithms have been highly successful and effective at solving a large number of complex optimization problems in science and engineering [12]. Another promising and efficient population-based metaheuristic is known as teaching–learning-based optimization (TLBO), which was inspired by natural teacher–learner interactions and learner–learner interactions to optimize learner knowledge.

### *1.2 Contribution*

In this paper, a discrete teaching–learning-based optimization (DTLBO) search metaheuristic is proposed for the optimization problem of making the best placement decisions for the offloaded tasks of IoT-based workflows on the available computing resources (including mobile devices, edge servers, or the cloud) while considering workflow delays, task dependencies, the communication time between tasks, and the power consumption of IoT devices while simultaneously satisfying workflow deadline constraints. The main contributions of this paper are as follows:

- The goal of the proposed model is to minimize an objective function based on the energy consumption of IoT devices and workflow delays while satisfying the deadline constraints of IoT workflows in an MEC environment while simultaneously considering task dependencies, the communication time between tasks, and task delays. Additionally, a queueing network that predicts wait times and service times for tasks on edge servers is used to model the dynamic workloads on the edge servers.
- A novel DTLBO algorithm based on mutation and crossover operators is proposed to make the optimal placement decisions for the offloaded tasks of IoT-based workflows between edge servers, the cloud, and mobile devices.
- Extensive experimental evaluations are conducted to evaluate the efficiency of the proposed offloading strategy and to evaluate its performance in comparison with that of no offloading, offloading to the cloud, random offloading to edge servers, and offloading using different metaheuristics, such as particle swarm optimization (PSO), the genetic algorithm (GA), and ant colony optimization (ACO).

The remainder of this paper is organized as follows. Section 2 discusses the related work on task offloading techniques for MEC environments. Section 3 presents the system model and the formulation of the problem for IoT-based workflow computation offloading. The proposed DTLBO placement algorithm is described in Section 4, and the experimental results are presented in Section 5. Finally, Section 6 concludes the paper and discusses future work.

## 2 Background and Related Works

This section discusses the existing research related to the computation offloading of IoT applications, in which computation tasks are offloaded to the MEC environment to overcome the limited computing and energy capacities of IoT devices while satisfying certain QoS requirements of delay-sensitive IoT applications. IoT applications can be classified into two categories: independent-task IoT applications and interdependent-task IoT applications. The IoT applications in the former category consist of several independent tasks with no constraints on their order of execution. In contrast, the IoT applications in the latter category consist of several communicating tasks, none of which can run unless the preceding tasks have been executed. These interdependent tasks form a workflow such that the overall QoS parameters of the application are highly dependent on the QoS parameters of each task, the sequence of execution, and the communication time between the tasks of the application. The following subsections present a survey of the previous work on computation offloading in the MEC environment for IoT applications in each category. For more details, see [7,13–15].

### *2.1 Independent-Task IoT Application Offloading*

In [16], a computation offloading strategy is proposed for mobile task offloading in a mobile fog computing environment by employing a Markov-based model; however, this approach is dependent on the network topology. In [17], a deep learning-based computation offloading strategy in an MEC environment is proposed. In [18], a computation offloading strategy in an MEC environment using Bayesian learning automata is proposed. In [19], an offloading and edge provisioning strategy is proposed using a long short-term memory model to predict the workload, and a reinforcement learning-based algorithm is proposed to ensure the appropriate scaling of offloading decisions; the limitation is that this approach assumes that the MEC servers have unlimited capacity. In [20], a heuristic for mobile task offloading considering the trade-off between energy consumption and the delay time is proposed; this offloading heuristic is highly dependent on the application type, which must be determined before making offloading decisions. A similar method is proposed in [21] based on successive convex approximation. In [22], an offloading strategy based on a GA is proposed to determine the tasks to be scheduled on mobile devices and the tasks to be considered for offloading to the cloud; in this method, optimization is achieved in a timely manner depending on the relational graph size. In [23], a heuristic based on the use of Lyapunov optimization to minimize the task processing delay and energy consumption is proposed. In [24], a quality-of-experience-based offloading placement policy using fuzzy logic is proposed. This placement policy prioritizes different placements of independent requests in accordance with user expectations and the capabilities of fog instances; however, fog resources are limited, and thus, heavy computations are not applicable in that layer. In [25], two different scheduling heuristics are proposed to ascertain whether each mobile task is to be processed locally on its mobile device or offloaded to an edge server or a cloud server; both heuristics are based on linear programming relaxation (LR) and distributed deep learning-based offloading (DDLO) algorithms. The experimental results show that the DDLO algorithm achieves better performance than the LR-based algorithm. In [26], a scheduling algorithm based on Lyapunov optimization is proposed to select the set of offloading requests to be admitted into an MEC environment based on green energy considerations for the edge devices; however, densely distributed MECs are not considered to reduce the energy consumption or improve performance. In [27], a strategy for task offloading is presented to minimize the computation time of tasks and the energy consumption using game theory. In [28], two nature-inspired offloading algorithms based on ACO and PSO are proposed with the aim of minimizing IoT task delays on unmanned aerial vehicle (UAV)-assisted edge servers; the experimental results show that the offloading strategy based on ACO is superior in terms of the task execution time.

### *2.2 IoT-based Workflow Application Offloading*

An IoT-based workflow consists of a group of mobile tasks that are related such that there are dependencies in their execution. The execution of a task cannot start unless its preceding tasks have finished their execution. An IoT-based workflow can be modeled as a DAG. Each task in the IoT workflow is represented by a vertex, and the edges between vertices represent the data flows between tasks.

In [29], an offloading strategy for multiple real-time IoT workflows in an MEC environment is proposed. The proposed strategy offloads computationally demanding tasks with low communication requirements to the cloud and communication-intensive tasks with low computational demands to the fog. This offloading strategy does not consider the energy requirements of the IoT devices. In [30], a joint optimization strategy for service caching and task offloading is proposed that selects an edge server to assist a particular mobile unit in executing a set of interdependent computation tasks. The problem is formulated as a mixed-integer nonlinear optimization problem; however, the complexity of such optimizations is very high, especially when applied to large-scale networks. In [31], a min-cost offloading partitioning algorithm is presented that aims to find a dynamic partitioning plan for a mobile application by determining which portions of the application must run on the mobile device and which must run on cloud/edge servers under different configurations of cloud, edge server, and mobile resources. However, the mathematical model in this algorithm needs to be updated regularly for each change in the environment. In [8], a priority-based selection algorithm for scheduling multiple IoT workflows on edge servers is proposed that considers the execution time of each workflow and the energy consumption of the mobile devices. However, this heuristic does not consider the heterogeneous characteristics and the dynamic loads of the MEC environment.

Regarding evolution-and nature-inspired algorithms, a deadline-and cost-aware offloading strategy based on a GA is proposed in [32] for the placement of IoT workflows on cloud virtual machines; however, while this algorithm minimizes the placement cost under a deadline constraint, it does not consider edge servers. In [26], an IoT data placement strategy for IoT workflows is proposed based on a GA and PSO that aims to reduce the cost of workflows in an MEC environment. In [33], an offloading strategy for workflows originating from multiple IoT devices distributed among multiple fog/edge servers and cloud servers is proposed, where the problem is formulated as a weighted cost function to simultaneously minimize the execution time and the energy consumption. Using this weighted cost function, an optimization approach is designed based on a memetic algorithm; however, due to the existence of heterogeneous placement targets, additional communication costs should be considered. In [34], an energy-efficient and deadline-aware task offloading strategy for IoT workflows in an MCC environment is proposed based on PSO that aims to minimize the energy consumption of mobile devices while simultaneously satisfying the deadline constraints of mobile workflows. Unfortunately, multiobjective optimization strategies for complex workflows with constrained time and concurrent workflows remain difficult to implement, particularly when working within environments composed of multiple cloudlets. In [35], an offloading strategy for IoT workflows based on joint optimization is proposed to balance QoS requirements and privacy protection. This offloading strategy is based on the nondominated sorting GA and considers the optimization of time and resource utilization in edge computing; however, this strategy does not consider the energy consumption of IoT devices.

## 3 MEC System Model and Problem Formulation

This section first presents the architecture of the MEC environment used in this study. Next, a detailed discussion of the structure of IoT workflows is provided. Furthermore, the objective function and corresponding constraints, including energy consumption, task dependencies, and workflow deadlines, for the offloading of IoT workflows in the MEC environment are introduced.

### *3.1 MEC Model*

The MEC architecture consists of three communicating layers, as shown in Fig. 1: an IoT layer, an edge layer, and a cloud layer. The IoT layer includes IoT sensors and mobile devices. The IoT layer communicates with the cloud layer through a WAN. The cloud layer consists of high-performance servers that provide computing services in the form of virtual machines. The edge layer consists of several cloudlet devices. Each cloudlet device is a computing server with wireless networking capabilities that offers computing services to IoT devices in close proximity [36]. Each cloudlet server can communicate with the cloud through the WAN. The cloudlet servers communicate wirelessly to collect offloading requests and status information. Each IoT device communicates with the closest edge server to send offloading requests for workflows. One cloudlet server can receive workflow offloading requests from multiple IoT devices. All offloading requests are transferred to an offloading manager for placement decisions to determine whether the workflow tasks can be processed locally on their mobile devices, offloaded to an edge server through the LAN, or offloaded to the cloud through the WAN for processing while maintaining the QoS requirements in terms of the workflow delay and IoT device energy consumption.

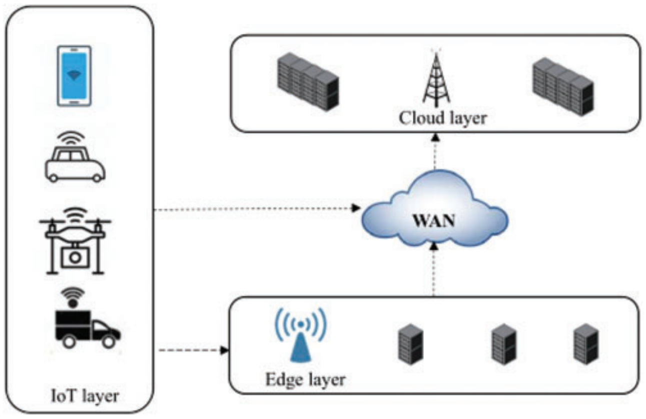

**Figure 1:** The MEC architecture

Suppose that there are $N_{cl}$ cloudlet nodes. Each cloudlet node $cl_j$ has $N_{vm}$ virtual machines with processing capacity $P_j^{cl}$, network latency $L_j^{cl}$, and network bandwidth $BW_j^{cl}$. The cloud has processing capacity $P^c$, network latency $L^c$, and network bandwidth $BW^c$.

### *3.2 IoT-based Workflow Model*

An IoT workflow consists of interdependent tasks modeled as a DAG, as shown in Fig. 2. Workflow $w_i$ is modeled as $w_i = \left(t_{ij},\ e_{ij}\right)$. The nodes in the DAG represent the set of tasks in the workflow, denoted by $t_{ij}, j = 1, 2, \ldots, N_{w_i}$, where $N_{w_i}$ is the number of tasks in workflow $w_i$. The edges $e_{ij}$ represent the dependencies between tasks. The execution of each task in the workflow cannot be started unless all preceding tasks have finished and all necessary data have been received from these preceding

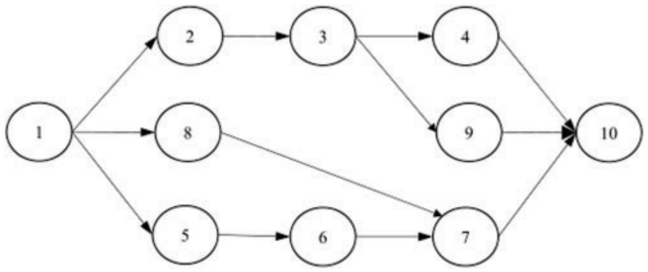

**Figure 2:** An example of a DAG representing an IoT workflow

tasks. For example, a task dependency relationship can be expressed as follows: $P\left(task\left(10\right)\right) = \left\{task\left(4\right),\ task\left(9\right),\ task\left(7\right)\right\}$.

### *3.3 System Model*

Suppose that $N_{wf}$ workflows are generated by the IoT layer. Each workflow $wf_i$ is represented by $wf_i = \left\{T_{wf_i},\ E_{wf_i},\ D_{wf_i}\right\}$, where $T_{wf_i}$, $E_{wf_i}$, and $D_{wf_i}$ are the set of tasks, the set of edges between tasks, and the deadline, respectively, for workflow $wf_i$. There are $N_{ts}^{wf_i}$ tasks in workflow $wf_i$. Each task is defined as $ts_i^{wf_i} = \left(TL_{ts_i}^{wf_i}, I_{ts_i}^{wf_i}, P_l\right)$, where $TL_{ts_i}^{wf_i}$, $I_{ts_i}^{wf_i}$, and $P_l$ represent the task length, the number of instructions in the task, and the processing capacity of the mobile device, respectively. Each edge in the set of edges between the workflow tasks is associated with a weight $d_{jk}^{wf_i}$ that is equal to the size of the data to be transmitted between task $ts_j^{wf_i}$ and task $ts_k^{wf_i}$.

### 3.3.1 The Objective Function

The main objective of the IoT workflow offloading manager is to find the optimal placement for the set of offloaded IoT workflow tasks among the available MEC computing resources, including mobile devices, edge servers, and the cloud, to minimize the following objective function:

$$minimize \sum_{i=1}^{N_{wf}} F\left(wf_i\right) \tag{1}$$

where $F\left(wf_i\right)$ is a multiobjective optimization function. The weighted sum is the common method for converting a multiobjective function into a single-objective function [37]. Therefore, a weighted metric combining the time delay of all workflows and the energy consumption of the IoT devices for a certain offloading configuration is used and is calculated using the following equation:

$$F\left(wf_i\right) = \omega_1 E_{wf_i} + \omega_2 T_{wf_i} \tag{2}$$

where $E_{wf_i}$ is the energy consumption resulting from the execution of workflow $wf_i$, $T_{wf_i}$ is the overall time delay for workflow $wf_i$, and $\omega_1$ and $\omega_2$ are weight factors $\in [0,\ 1]$ that establish a balance between the energy consumption and the workflow delay.

This objective function is subject to deadline constraints such that the execution time of each workflow $wf_i$ must be less than a specified deadline, as expressed by the following equation:

$$T_{wf_i} < D_{wf_i},\ \ \forall wf_i \in \left\{1, 2, \cdots, N_{wf}\right\} \tag{3}$$

### 3.3.2 Energy Consumption Model

The energy consumption for workflow $wf_i$ is the sum of the energy consumption values for all tasks in the workflow and is calculated using the following equation:

$$E_{wf_i} = \sum_{j=1}^{N_{ts}^{wf_i}} E_{ts_j}^{wf_i} \tag{4}$$

The energy consumption model adopted here is based on the CPU cycle used in [38], where the energy consumption for a single CPU cycle is calculated as follows:

$$E = \kappa \times\ P^2 \tag{5}$$

where $\kappa$ is an energy factor whose value depends on CPU chip technology (in this paper, its value is set to $\kappa = 10^{-26}$ [39]) and $P$ is the computing power of the device, which is the CPU frequency.

The amount of energy required to execute a task locally on the corresponding mobile device is calculated using the following equation:

$$E_{ts_i}^{wf_i} = \kappa \times (P_l)^2 \times I_{ts_i}^{wf_i} \tag{6}$$

For the offloaded tasks of the workflow, the energy consumption is mainly the energy required to upload each offloaded task, which is calculated as follows:

$$E_{ts_j}^{wf_i} = TL_{ts_j}^{wf_i} \times E_{trans} \tag{7}$$

where $E_{trans}$ is the energy required for the mobile device to transmit one unit of data.

### *3.3.3 Time Delay Model*

The execution time for workflow $wf_i$ is the sum of the computation times for all tasks in the workflow and the time for data transmission between tasks and is calculated using the following equation:

$$T_{wf_i} = CT\left(ts_{exit}^{wf_i}\right) \tag{8}$$

where $CT\left(ts_{exit}^{wf_i}\right)$ is the computation time for the exit task of the workflow, $ts_{exit}$, and is calculated using the following equation:

$$CT\left(ts_{exit}^{wf_i}\right) = \max_{ts_i^{wf_i} \in P\left(ts_{exit}^{wf_i}\right)} \left\{CT\left(ts_i^{wf_i}\right) + TR\left(ts_{exit}^{wf_i}\ ,\ ts_i^{wf_i}\right)\right\} \tag{9}$$

where $TR\left(ts_j^{wf_i}, ts_k^{wf_i}\right)$ is the transmission time between tasks $ts_j^{wf_i}$ and $ts_k^{wf_i}$ and is calculated using the following equation:

$$TR\left(ts_j^{wf_i}, ts_k^{wf_i}\right) = \frac{d_{jk}^{wf_i}}{R} \tag{10}$$

where $R$ is the transmission rate between the two tasks.

The computation time resulting from executing task $ts_i^{wf_i}$ locally on the mobile device is calculated using the following equation:

$$CT\left(ts_i^{wf_i}\right) = \frac{I_{ts_i}^{wf_i}}{P_l} \tag{11}$$

The computation time resulting from offloading task $ts_i^{wf_i}$ to a cloudlet is calculated as the sum of the transmission time, the waiting time in the queue, and the service time, as shown in Eq. (12):

$$CT_{cl}\left(ts_i^{wf_i}\right) = Tr_{cl_j}\left(ts_i^{wf_i}\right) + WT_{cl_j}\left(ts_i^{wf_i}\right) + ST_{cl_j}\left(ts_i^{wf_i}\right) \tag{12}$$

where $Tr_{cl_j}\left(ts_i^{wf_i}\right)$ is the transmission time of task $ts_i^{wf_i}$ between the mobile device and the cloudlet node, $WT_{cl_j}\left(ts_i^{wf_i}\right)$ is the waiting time in the queue of cloudlet node $cl_j$, and $ST_{cl_j}\left(ts_i^{wf_i}\right)$ is the service time on cloudlet node $cl_j$.

The transmission time $Tr_{cl_j}\left(ts_i^{wf_i}\right)$ considers the transmission latency, the task length, and the transmission bandwidth of the wireless channel between the mobile device and the cloudlet node and is calculated using the following equation:

$$Tr_{cl_j}\left(ts_i^{wf_i}\right) = L_{cl} + \frac{TL_{ts_i}^{wf_i}}{BW_{cl}} \tag{13}$$

where $L_{cl}$ and $BW_{cl}$ are the latency and bandwidth, respectively, of the cloudlet node.

The waiting time at cloudlet node $cl_j$, $WT_{cl_j}\left(ts_i^{wf_i}\right)$, is calculated using the $M/M/vm_{cl_j}$ queuing model for the cloudlet node, which assumes that there are $vm_{cl_j}$ virtual machines acting as servers in the queuing system and is expressed as follows:

$$WT_{cl_j} = \frac{\mu_{cl_j}\left(\lambda_{cl_j}/\mu_{cl_j}\right)^2}{\left(vm_{cl_j} - 1\right)!\left(vm_{cl_j}\ \mu_{cl_j} - \lambda_{cl_j}\right)} \tag{14}$$

where $\lambda_{cl_j}$ and $\mu_{cl_j}$ are the arrival rate and the service rate, respectively, at cloudlet node $cl_j$. The service time $ST_{cl_j}\left(ts_i^{wf_i}\right)$ is calculated as follows:

$$ST_{cl_j}\left(ts_i^{wf_i}\right) = \frac{I_{ts_i}^{wf_i}}{P_{cl}} \tag{15}$$

The computation time resulting from offloading a task to the cloud is calculated as the sum of the transmission time and the service time, as shown in Eq. (16).

$$CT_{cl}\left(ts_i^{wf_i}\right) = Tr_c\left(ts_i^{wf_i}\right) + ST_c\left(ts_i^{wf_i}\right) \tag{16}$$

The transmission time $Tr_c\left(ts_i^{wf_i}\right)$ considers the task length and the transmission bandwidth of the WAN channel between the mobile device and the cloud and is calculated as follows:

$$Tr_c\left(ts_i^{wf_i}\right) = L_c + \frac{TL_{ts_i}^{wf_i}}{BW_c} \tag{17}$$

where $L_c$ and $BW_c$ are the latency and bandwidth, respectively, of the cloud WAN connection.

The service time for task $ts_i$ in the cloud, $ST_{ts_i}^{wf_i}(c)$, is calculated as follows:

$$ST_c\left(ts_i^{wf_i}\right) = \frac{I_{ts_i}^{wf_i}}{P_c} \tag{18}$$

### *3.4 The Optimization Problem*

The main goal of this study is to computationally offload IoT workflows in an MEC environment to optimize the energy consumption of mobile devices and workflow delay considering task delays, task dependencies, the communication time between interdependent tasks, and the workload conditions on the edge servers while simultaneously satisfying workflow deadline constraints. The optimization problem can be formulated as follows:

$$\mathbf{P}: \qquad minimize \sum_{wf_i}\left(\omega_1 E_{wf_i} + \omega_2 T_{wf_i}\right) \tag{19}$$

such that

$$C1: \omega_1 + \omega_2 = 1,\ \ \omega_1,\ \omega_2 \in [0, 1] \tag{20}$$

$$\mathrm{C2}: \mathrm{T}_{\mathrm{wf_i}} < \mathrm{D}_{\mathrm{wf_i}} \tag{21}$$

where $\omega_1$ and $\omega_2$ are weight parameters for the tradeoff between the time delay and the energy consumption in the minimization goal and satisfy the constraint $C1$. Constraint $C2$ states that the workflow delay should be less than the deadline constraint.

## 4 Teaching Learning-Based Optimization Metaheuristic

TLBO is a stochastic population-based search algorithm that is inspired by knowledge transfer in a classroom environment [40]. The algorithm follows the teaching–learning structure, and thus, the algorithm is divided into two phases, namely, learning and teaching. In the teaching phase, different learners acquire knowledge through learners' interactions between themselves, and the teacher tries to improve the performance of all students using its knowledge level. In the teacher phase, a new solution is generated using the best solution and the mean of the population using the following equation:

$$X_{new} = X_{old} + r\left(X_{teacher} - T_f X_{mean}\right) \tag{22}$$

where $X_{new}$ is the new solution, $X_{old}$ is the current solution, $X_{teacher}$ is the best solution found, $X_{mean}$ is the mean of the population, $T_f$ is a teaching factor that weights the change using the mean, and $r$ is a random number in the interval [0 1] to be selected for each variable in the solution. A greedy selection is applied, and the new solution is accepted if better than the old solution.

In the student phase, knowledge is transferred through the interactions between the students. A new solution is generated using a random partner solution selected from the population, and greedy selection is applied to the new solution. The new solution is generated using the following equation:

$$X_{new} = X_{old} + r\left(X_{old} - X_r\right) \tag{23}$$

where $X_r$ is a random solution from the population.

During each iteration, greedy selection is applied for the population update. The objective function is evaluated for the new solution $X_{new}$ and the old solution $X_{old}$, and the solution with the better objective function survives until the next iteration. The following subsection introduces the proposed DTLBO algorithm.

To apply TLBO to a discrete optimization problem, first, the solution must be encoded using discrete values. Second, the solution update uses mutation and crossover operators to produce discrete values during the encoding of a solution, thereby generating a particular configuration for the offloading of IoT workflow tasks to computational resources, including mobile devices, cloudlet servers, or the cloud. Fig. 3 represents a solution encoding for the assignment of five different tasks, namely, 5 genes, to computational resources $\{C_1, C_3, C_4\}$. The length of the solution is the number of offloaded tasks $N_{tasks}$. In the teaching phase, learners update their knowledge using $X_{teacher}$ and $X_{mean}$\$. $X_{teacher}$ is defined using the best solution found among the learners in the entire population. $X_{mean}$ is defined using the mode operation, $X_{mode}$, over the entire population, which returns the maximum occurrence of the value of each specific gene in the population. Eq. (22) is modified using the mutation and crossover operators as follows:

$$X_{new} = p_c f_c\left(X_{old}, p_c f_c\left(X_{best}, p_m f_m\left(X_{mode}\right)\right)\right) \tag{24}$$

where $f_m$ and $f_c$ are mutation and uniform crossover operators, respectively, and $p_c$ and $p_m$ are the crossover and mutation probabilities, respectively. A random number is generated. If the random number is smaller than the mutation probability $p_m$, a mutation operation is applied to $X_{mode}$. A uniform crossover $f_c$ is applied between $X_{best}$ and the mutated $X_{mode}$ when the generated random number is smaller than the crossover probability $p_c$. Another uniform crossover is performed between the old solution and the result of the previous crossover. The accepted solution resulting from the crossover operator is based on greedy selection. Eq. (23) is modified as follows:

$$X_{new} = p_c f_c \left(X_{old},\ X_r\right) \tag{25}$$

where a uniform crossover is applied between the old solution and a random learner $X_r$.

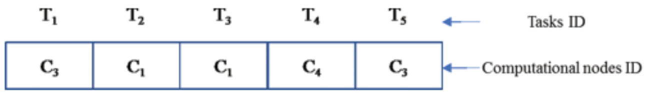

**Figure 3:** An example of a solution encoding for mapping tasks to edge resources

Algorithm 1 presents the steps of the proposed DTLBO metaheuristic. Initially, a random population is generated. The fitness of each individual solution in the population is calculated using Eq. (19). The solution with the minimum fitness is set as the best solution $X_{best}$. The algorithm iterates for a certain number of iterations $N_{iter}$. At each iteration, the teaching phase and the learning phase are conducted. In the teaching phase, $X_{mode}$ is calculated, where the maximum number occurrence for each value in each gene in the population is selected. $X_{mode}$ is mutated with probability $P_m$. A uniform crossover operator is applied between $X_{mode}$ and $X_{best}$ with probability $P_c$. A greedy selection is applied, and the output solution goes through another uniform crossover operator with the old value of the individual solution. Again, greedy selection is applied. During the learning phase, a random solution is selected, and a uniform crossover operator is applied at the recent value of each individual with its corresponding random solution. The resulting solution goes through greedy selection, and the solution with a smaller fitness value is selected for the next iteration. After each iteration, the solution with the minimum fitness value is selected as the best solution $X_{best}$.

## 5 Performance Evaluation

This section introduces the parameter settings of the proposed DTLBO algorithm and a description of the evaluation methodology. Finally, an in-depth performance evaluation of the proposed algorithm is presented.

### *5.1 Environment Setup and Parameter Settings*

Simulations were performed on a PC with a 2.60 GHz Intel Core i7–4510 CPU and 8 GB of RAM. The parameter settings for the conducted simulation experiments are shown in Tab. 1. In the experiments, ten cloudlet nodes with different computing capacities and network latencies were considered. Furthermore, ten different workflows were used in the evaluation, where each workflow is randomly generated and consists of a number of tasks between 10 and 15.

**Algorithm 1:** The proposed DTLBO metaheuristic

**Input:** The parameters $P_c$, $P_m$, $N_{tasks}$, $N_{iter}$, $N_{pop}$, and $N_{cl}$
**Output:** The global best solution $X_{best}$

```
  Generate a random population $X_i$, $i = 1,2, \ldots, N_{pop}$
  Evaluate the fitness value for each individual nest, $F(X_i)$, using Eq. (19)
  Find the best solution $X_{bset}$ which has minimum fitness value
   For iteration $t = 1 : N_{iter}$ do
      For individual $i = 1 : N_{pop}$ do
         // Teaching phase
         Calculate $X_{mode}$ over the entire population $X_i$
         Generate a random number $r$
         if $(r < P_m)$ apply mutation operation on $X_{mode}$
         Generate a random number $r$
         if $(r < P_c)$ $X_i^t$ = Uniform crossover operation between $X_{mode}$ and $X_{best}$
         Generate a random number $r$
         if $(r < P_c)$ $X_i^t$ = Uniform crossover operation between $X_i^{t-1}$ and $X_i^t$
         apply the greedy selection between $X_i^t$ and $X_i^{t-1}$
         // Learning phase
         Select a random partner $X_r^t$
         Generate a random number $r$
        if $(r < P_c)$
          $X_i^t$ by applying crossover operation between $X_r^t$ and $X_i^t$
          apply the greedy selection and update $X_i^t$
      end
   end
  Evaluate the fitness of each individual $X_i^t$
  Update $X_{best}$ with the solution which has minimum fitness
}
```

**Table 1:** Parameter settings for the evaluation experiments

| Parameter | | Value |
|---|---|---|
| Number of cloudlets | $N_{cl}$ | 10 |
| Mobile processing capacity | $P_l$ | 1000 MHz |
| Cloudlet processing capacity | $P_{cl}$ | 0.2 |
| Cloud processing capacity | $P_{cloud}$ | 3000 MHz |
| Cloudlet latency | $L_{cl}$ | [0.5–2.0] ms |
| Cloud latency | $L_{cloud}$ | 30 ms |
| Cloudlet bandwidth | $BW_{cl}$ | [2000–4000] kB/s |
| Cloud bandwidth | $BW_{clould}$ | [500, 1000] kB |
| Mobile idle energy consumption | $E_{idle}$ | 1 W |
| Mobile processing energy | $E_l$ | 5 W |

(Continued)

**Table 1:** Continued

| Parameter | | Value |
|---|---|---|
| Mobile transmission energy | $E_{trans}$ | 10 W |
| Number of workflows | $N_{wf}$ | 10 |
| Number of tasks in a workflow | $T_{ts}$ | 10–15 |
| Task length | $L_{ts}$ | [300, 800] kB |
| Number of instructions in a task | $I_{ts}$ | [10 × ^9–5 × 10^9] |

### 5.2 *Experimental Methodology*

The following methods are considered for comparison to evaluate the efficiency of the proposed algorithm:

- No offloading (NO): In this method, all workflow tasks are executed locally on their respective mobile devices. The results of this method are used as the baseline to evaluate the performance gain of the proposed offloading algorithms.
- Cloud offloading (CO): In this method, all workflow tasks are offloaded to the cloud.
- Random cloudlet offloading (CLO): In this method, all workflow tasks are randomly offloaded to cloudlets.
- PSO offloading: In this method, all workflow tasks are offloaded in accordance with the placement decisions of the PSO algorithm [41] with the parameter settings $C_1 = 0.8$ and $C_2 = 1.2$, $N_{iter} = 100$, and $N_{pop} = 25$. The parameters are determined according to the best result obtained after repeating the experiments several times.
- ACO offloading: In this method, all workflow tasks are offloaded in accordance with the placement decisions of the ACO algorithm proposed in [10].
- DTLBO offloading: In this method, all workflow tasks are offloaded in accordance with the placement decisions of the proposed DTLBO placement metaheuristic with the parameter settings shown in Tab. 2. The crossover and mutation probabilities $P_C$ and $P_m$ values are determined according to the obtained values of the objective function with several trial experiments. Fig. 4 shows the average of the corresponding objective function with different values of the parameter for 10 different runs. The figure shows that the minimum objective function is obtained for $P_m = 0.2$ and $P_C = 0.8$.
- Genetic algorithm (GA) offloading: In this method, all workflow tasks are offloaded with the placement decisions are of the GA algorithm [42] with the same crossover and permutation probabilities of DTLBO.

**Table 2:** Parameter settings of the proposed DTLBO metaheuristic

| Parameter | | Value |
|---|---|---|
| Number of iterations | $N_{iter}$ | 100 |
| Population size | $N_{pop} = 25$ | 25 |
| Mutation probability | $P_m$ | 0.2 |
| Crossover probability | $P_C$ | 0.9 |

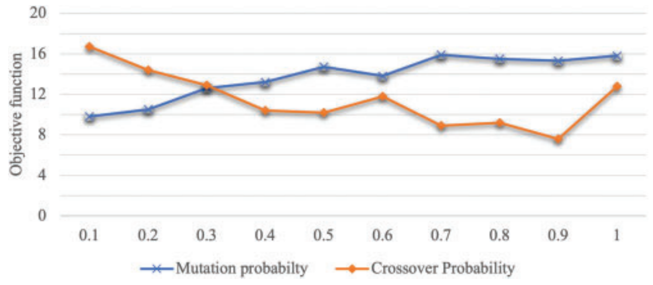

**Figure 4:** The objective functions for the corresponding mutation and crossover probabilities

### 5.3 *Experimental Results*

Fig. 5a shows the total delay resulting from the placement of ten workflows as the bandwidth increases. The proposed DTLBO algorithm shows the least delay among all the comparison methods, indicating that it makes better placement decisions for the tasks of the workflows. Furthermore, the NO and CO methods are the worst in the sense that they result in the largest delays. Compared with CLO, the proposed offloading strategy reduces the workflow delays by 47% in the best case. Additionally, Fig. 5b shows the resulting energy consumption as the bandwidth increases for each scheduling method. This figure shows that the proposed algorithm also achieves the lowest energy consumption. Compared with CLO, the proposed offloading strategy successfully reduces the energy consumption of the mobile device delays by 34% in the best case.

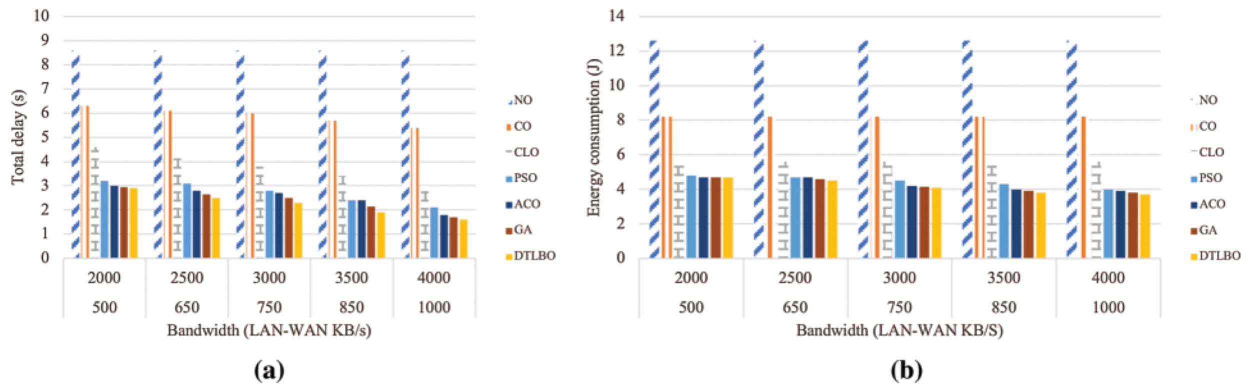

**Figure 5:** (a) The response times and (b) The energy consumption under different bandwidths

Fig. 6a shows the overall delay of the IoT workflows as the number of tasks increases. The proposed algorithm again yields the best results, demonstrating great potential for achieving performance gains in the case of large-scale offloading decisions for a large number of workflows. In this scenario, compared to the strategy using the CLO placement method, the proposed strategy successfully reduces the delay by 67% in the best case. Fig. 6b shows the resulting energy consumption as the number of workflow tasks increases for each scheduling method. The proposed offloading strategy again achieves the lowest energy consumption. Compared to the strategy using the CLO placement method, the proposed offloading strategy reduces the energy consumption of mobile devices by 22% in the best case.

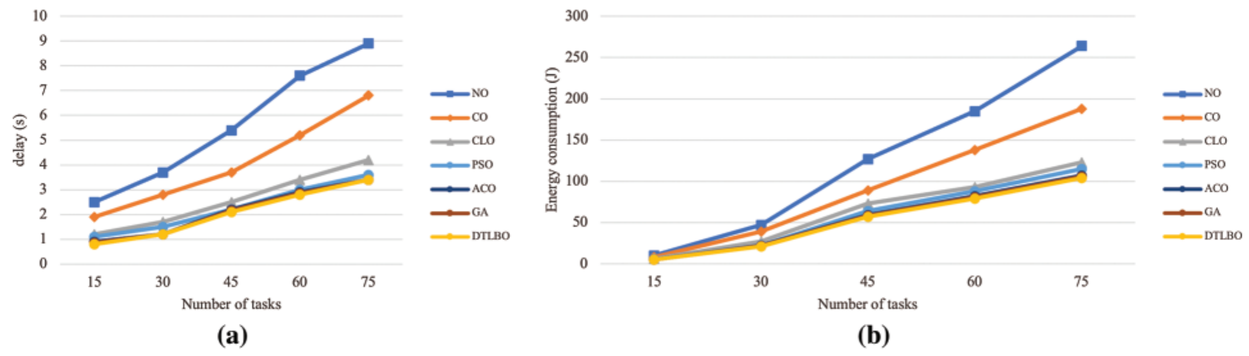

**Figure 6:** (a) The response times and (b) The energy consumption under different numbers of tasks

Fig. 7a shows the delay of the workflows as the task size increases. In this experiment, the delay increases as the task size increases, and the number of instructions increases accordingly. The proposed offloading algorithm maintains the highest performance gain in terms of the response times for the workflows. In this scenario, the proposed strategy improves the delay of the workflows by 60% compared to the offloading strategy using the CLO method. Furthermore, Fig. 7b shows the resulting energy consumption as the task sizes of the workflows increase. The proposed offloading strategy is seen to be effective at reducing the energy consumption by 18% compared to the CLO method.

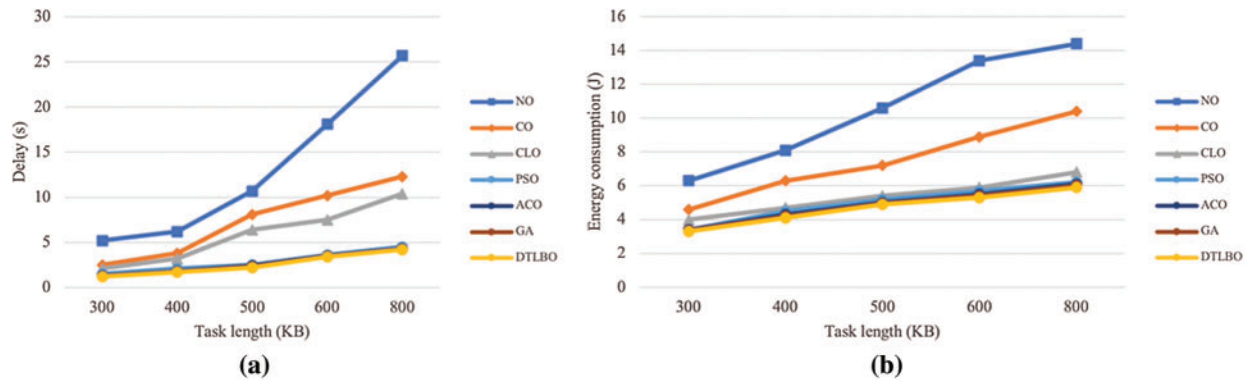

**Figure 7:** (a) The response times and (b) The energy consumption under different task sizes

Fig. 8a shows the delay of the workflows as the number of cloudlets increases. Compared to the CLO offloading strategy, the offloading strategy using the DTLBO metaheuristic still maintains the lowest workflow delay by 35%. Fig. 8b shows the resulting energy consumption as the number of cloudlets increases. Compared to the CLO strategy, the proposed offloading strategy is once again effective in reducing the energy consumption by 40%. Hence, the proposed offloading strategy is effective in achieving the lowest response times for IoT-based workflows and the lowest energy consumption for mobile devices.

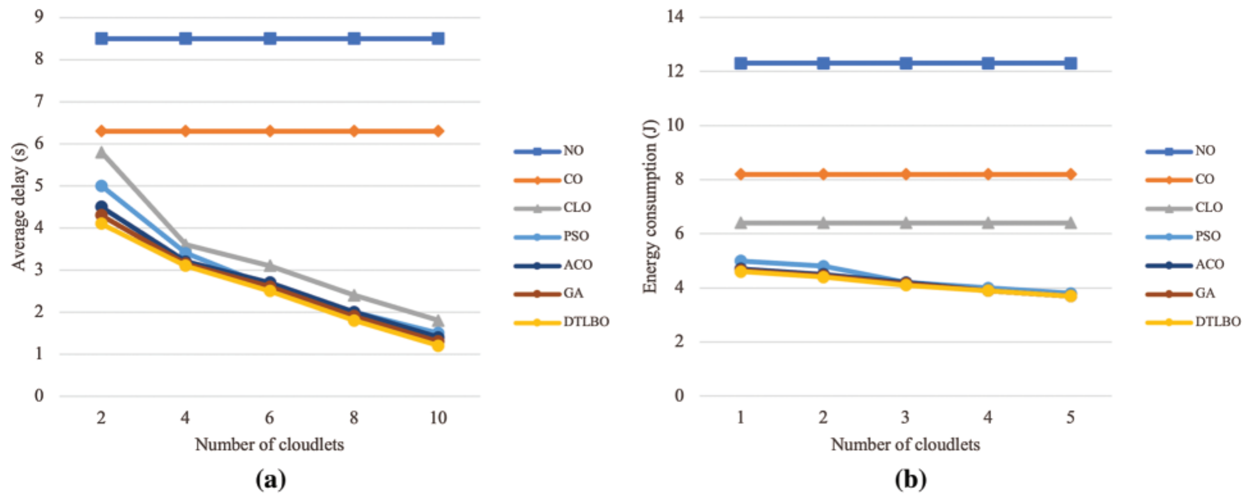

**Figure 8:** (a) The response times and (b) the energy consumption under different numbers of cloudlets

## 6 Conclusion and Future Work

This paper addresses the problems of computation offloading and scheduling decisions for IoT-based workflows with different QoS requirements in an MEC environment considering interdependent task delays, task dependencies, the time for communication between tasks, and the energy consumption of IoT devices while simultaneously considering the deadline constraints of workflows. Furthermore, an intelligent placement metaheuristic is proposed based on the TLBO metaheuristic. The proposed DTLBO metaheuristic is modified to suit the discrete nature of the optimization problem and to enhance the search of the offloading placement selection using mutations and crossover operators. Finally, the performance of the proposed algorithm is compared with that of different alternative search metaheuristics. The experimental results show that the proposed offloading strategy using the proposed algorithm effectively minimizes the response times for IoT-based workflows while maintaining the lowest energy consumption of mobile devices. In future work, the dynamic nature of workflow tasks and their applications to heterogeneous systems will be investigated. A direct joint multiobjective optimization method will also be compared with the proposed offloading strategy. Furthermore, because the use of UAV-based edge computing can achieve even lower task delays, a UAV-based offloading strategy will be investigated. Finally, an offloading strategy that considers the mobility of IoT devices is an important research challenge.

**Funding Statement:** The authors received no specific funding for this study.

**Conflicts of Interest:** The authors declare that they have no conflicts of interest to report regarding the present study.

## References

[1] T. Qiu, N. Chen, K. Li, M. Atiquzzaman and W. Zhao, “How can heterogeneous internet of things build our future: A survey,” *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2011–2027, 2018.

[2] Y. Zhang, Z. Zhou, Z. Shi, L. Meng and Z. Zhang, "Online scheduling optimization for DAG-based requests through reinforcement learning in collaboration edge networks," *IEEE Access*, vol. 8, pp. 72985–72996, 2020.

[3] M. Adhikari, T. Amgoth and S. N. Srirama, "A survey on scheduling strategies for workflows in cloud environment and emerging trends," *ACM Computing Surveys*, vol. 52, no. 4, pp. 1–36, 2020.

[4] S. Deng, L. Huang, J. Taheri and A. Y. Zomaya, "Computation offloading for service workflow in mobile cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 12, pp. 3317–3329, 2015.

[5] J. Ren, D. Zhang, S. He, Y. Zhang and T. Li, "A survey on end-edge-cloud orchestrated network computing paradigms," *ACM Computing Surveys*, vol. 52, no. 6, pp. 1–36, 2020.

[6] L. Lin, X. Liao, H. Jin and P. Li, "Computation offloading toward edge computing," in *Proc. of the IEEE*, vol. 107, no. 8, pp. 1584–1607, Aug. 2019.

[7] A. Shakarami, A. Shahidinejad and M. Ghobaei-Arani, "A review on the computation offloading approaches in mobile edge computing: A game-theoretic perspective," *Software: Practice and Experience*, vol. 50, no. 9, pp. 1719–1759, 2020.

[8] H. Kanemitsu, M. Hanada and H. Nakazato, "Multiple workflow scheduling with offloading tasks to edge cloud," In: D. Da Silva, Q. Wang and L. -J. Zhang, (Eds) *Cloud Computing–CLOUD 2019*, Cham: Springer International Publishing, pp. 38–52, 2019.

[9] M. H. Mousa and M. K. Hussein, "Efficient UAV-based MEC using GPU-based PSO and voronoi diagrams," *Computer Modeling in Engineering & Sciences*, 2022, https://doi.org/10.32604/cmes.2022.020639.

[10] M. K. Hussein and M. H. Mousa, "Efficient task offloading for IoT-based applications in fog computing using ant colony optimization," *IEEE Access*, vol. 8, pp. 37191–37201, 2020.

[11] S. A. Zakaryia, S. A. Ahmed and M. K. Hussein, "Evolutionary offloading in an edge environment," *Egyptian Informatics Journal*, vol. 22, no. 3, pp. 257–267, 2021.

[12] X. -S. Yang, "Nature-inspired optimization algorithms: Challenges and open problems," *Journal of Computational Science*, vol. 46, pp. 101104, 2020.

[13] A. Shakarami, M. Ghobaei-Arani, M. Masdari and M. Hosseinzadeh, "A survey on the computation offloading approaches in mobile edge/cloud computing environment: A stochastic-based perspective," *Journal of Grid Computing*, vol. 18, no. 4, pp. 639–671, 2020.

[14] 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*, vol. 182, pp. 107496, 2020.

[15] A. Shakarami, A. Shahidinejad and M. Ghobaei-Arani, "An autonomous computation offloading strategy in mobile edge computing: A deep learning-based hybrid approach," *Journal of Network and Computer Applications*, vol. 178, pp. 102974, 2021.

[16] F. Jazayeri, A. Shahidinejad and M. Ghobaei-Arani, "A latency-aware and energy-efficient computation offloading in mobile fog computing: A hidden markov model-based approach," *The Journal of Supercomputing*, vol. 77, no. 5, pp. 4887–4916, 2021.

[17] F. Jazayeri, A. Shahidinejad and M. Ghobaei-Arani, "Autonomous computation offloading and auto-scaling the in the mobile fog computing: A deep reinforcement learning-based approach," *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, no. 8, pp. 8265–8284, 2021.

[18] F. Farahbakhsh, A. Shahidinejad and M. Ghobaei-Arani, "Multiuser context-aware computation offloading in mobile edge computing based on Bayesian learning automata," *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 1, pp. e4127, 2021.

[19] A. Shahidinejad and M. Ghobaei-Arani, "Joint computation offloading and resource provisioning for edge-cloud computing environment: A machine learning-based approach," *Software: Practice and Experience*, vol. 50, no. 12, pp. 2212–2230, 2020.

[20] D. G. Roy, D. De, A. Mukherjee and R. Buyya, "Application-aware cloudlet selection for computation offloading in multi-cloudlet environment," *The Journal of Supercomputing*, vol. 73, no. 4, pp. 1672–1690, 2017.

[21] E. El Haber, T. M. Nguyen, D. Ebrahimi and C. Assi, "Computational cost and energy efficient task offloading in hierarchical edge-clouds," in *2018 IEEE 29th Annual Int. Symp. on Personal, Indoor and Mobile Radio Communications (PIMRC)*, Bologna, Italy, pp. 1–6, Sep. 2018.

[22] M. Goudarzi, M. Zamani and A. T. Haghighat, "A fast hybrid multi-site computation offloading for mobile cloud computing," *Journal of Network and Computer Applications*, vol. 80, no. C, pp. 219–231, 2017.

[23] Y. Nan, W. Li, W. Bao, F. C. Delicato, P. F. Pires *et al.,* "A dynamic tradeoff data processing framework for delay-sensitive applications in cloud of things systems," *Journal of Parallel and Distributed Computing*, vol. 112, pp. 53–66, 2018.

[24] R. Mahmud, S. N. Srirama, K. Ramamohanarao and R. Buyya, "Quality of experience (QoE)-aware placement of applications in fog computing environments," *Journal of Parallel and Distributed Computing*, vol. 132, pp. 190–203, 2019.

[25] 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, pp. 1446, 2019.

[26] W. Chen, D. Wang and K. Li, "Multi-user multi-task computation offloading in green mobile edge cloud computing," *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 726–738, 2019.

[27] Z. Hong, W. Chen, H. Huang, S. Guo and Z. Zheng, "Multi-hop cooperative computation offloading for industrial IoT–edge–cloud computing environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 12, pp. 2759–2774, 2019.

[28] M. H. Mousa and M. K. Hussein, "Efficient UAV-based mobile edge computing using differential evolution and ant colony optimization," *PeerJ Computer Science*, vol. 8, pp. e870, 2022.

[29] G. L. Stavrinides and H. D. Karatza, "A hybrid approach to scheduling real-time IoT workflows in fog and cloud environments," *Multimedia Tools and Applications*, vol. 78, no. 17, pp. 24639–24655, 2019.

[30] S. Bi, L. Huang and Y. -J. A. Zhang, "Joint optimization of service caching placement and computation offloading in mobile edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 19, no. 7, pp. 4947–4963, 2020.

[31] H. Wu, W. J. Knottenbelt and K. Wolter, "An efficient application partitioning algorithm in mobile environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 7, pp. 1464–1480, 2019.

[32] X. Ma, H. Gao, H. Xu and M. Bian, "An IoT-based task scheduling optimization scheme considering the deadline and cost-aware scientific workflow for cloud computing," *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, no. 1, pp. 249, 2019.

[33] M. Goudarzi, H. Wu, M. Palaniswami and R. Buyya, "An application placement technique for concurrent IoT applications in edge and fog computing environments," *IEEE Transactions on Mobile Computing*, vol. 20, no. 4, pp. 1298–1311, 2021.

[34] Y. Wang, L. Wu, X. Yuan, X. Liu and X. Li, "An energy-efficient and deadline-aware task offloading strategy based on channel constraint for mobile cloud workflows," *IEEE Access*, vol. 7, pp. 69858–69872, 2019.

[35] X. Xu, C. He, Z. Xu, L. Qi, S. Wan *et al.,* "Joint optimization of offloading utility and privacy for edge computing enabled IoT," *IEEE Internet of Things Journal*, vol. 7, no. 4, pp. 2622–2629, 2020.

[36] M. Satyanarayanan, "The role of cloudlets in hostile environments," in *Proc. of the Fourth ACM Workshop on Mobile Cloud Computing and Services-MCS '13*, Taipei, Taiwan, vol. 12, no. 4, pp. 1, 2013.

[37] J. Yan, S. Bi and Y. J. A. Zhang, "Offloading and resource allocation with general task graph in mobile edge computing: A deep reinforcement learning approach," *IEEE Transactions on Wireless Communications*, vol. 19, no. 8, pp. 5404–5419, 2020.

[38] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 4, pp. 974–983, 2015.

[39] J. Zhang, H. Xiping, N. Zhaolong, N. Edith, Z. Li *et al.,* "Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2633–2645, 2018.

[40] R. V. Rao, V. J. Savsani and D. P. Vakharia, “Teaching–learning-based optimization: A novel method for constrained mechanical design optimization problems,” *Computer-Aided Design*, vol. 43, no. 3, pp. 303–315, 2011.

[41] R. Poli, J. Kennedy and T. Blackwell, “Particle swarm optimization,” *Swarm Intelligence*, vol. 1, no. 1, pp. 33–57, 2007.

[42] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Boston, United States: Addison-Wesley, 1989.