ELITE: Energy and Latency-Optimized Task Offloading for DVFS-Enabled Resource-Constrained Devices in MEC

Akhirul Islam(✉) and Manojit Ghose

IIIT Guwahati, Guwahati, India
akhirul.islam@iiitg.ac.in

Abstract. Multi-access Edge Computing (MEC) technology offers promising support for modern, computation-intensive, and time-sensitive applications. Many of these applications are generated by resource-constrained handheld or mobile UE. Due to limited resources, offloading certain parts of these applications (tasks) to connected MEC servers becomes essential. However, MEC servers also have limited resources compared to cloud servers, highlighting the need for efficient task offloading policies for UE devices and optimal resource allocation policies for MEC servers. This paper introduces ELITE (Energy and Latency-optimized Task Offloading and Resource Allocation for DVFS-Enabled devices), a novel solution to the energy and latency minimization problem in a cooperative heterogeneous MEC architecture. The proposed policy aims to minimize the energy consumption of the UE devices and the latency of the applications while satisfying application deadlines and dependency constraints. Furthermore, we consider the UEs to be enabled by dynamic voltage and frequency scaling (DVFS). Through extensive simulations using a real dataset, we demonstrate that our proposed strategy surpasses the state-of-the-art policy, achieving a remarkable 10% reduction in latency and an impressive 2× reduction in energy consumption of UE devices.

Keywords: Multi-access Edge Computing (MEC) · DVFS · cooperative MEC system · Mobile Edge Computing (MEC) · energy and latency

1 Introduction

With the rise of compute-intensive and latency-sensitive applications like real-time online games, virtual reality (VR), image processing, and IoT applications, there is a growing need for a new network and computing paradigm to cater to these demands. Mobile Cloud Computing (MCC) has been a popular solution, forwarding resource-intensive tasks to powerful cloud servers [20]. However, the geographical distance between UE and cloud servers introduces significant delays. To address this latency issue, the Multi-access Edge Computing (MEC) framework has emerged as a promising approach, offering cloud computing capabilities at the edge of the network [14,29]. This enables the execution of resource-intensive, latency-sensitive applications, leading to improved performance and reduced delays [23].

With MEC servers possessing limited computing resources unlike central cloud, efficient resource management becomes crucial to maximize the benefits of the MEC framework. Addressing three key challenges [1] is essential: 1) Offloading, which determines whether an application should be executed locally or remotely (MEC or cloud); 2) Resource allocation, responsible for efficiently allocating computing resources to applications; and 3) Task scheduling, deciding the order of processing applications or tasks while meeting constraints like deadlines.

Various researchers have explored different system models in the existing literature. For example, two-tier models have been studied in [6], three-tier models in [27,30], and four-tier models in [2,5]. Some authors have also explored SDN-based models in [1]. However, most of these models focus either on the MEC server alone (two-tier) or on the cooperation between the MEC server and the cloud server (three-tier). The existing literature has not considered cooperation among neighbouring MEC servers regarding task offloading, even though coordination and cooperation among MEC servers are considered in the context of caching [7]. In our research, we propose a cooperative MEC server architecture in conjunction with the cloud, where MECs hosted in different base stations cooperate with each other. This approach is essential for two reasons: i) It facilitates efficient resource management for MEC servers, which are typically resource-constrained compared to the cloud, and ii) It helps reduce task latency by executing a larger number of tasks in neighbouring base stations instead of forwarding them to the cloud. In addition to the cooperative MEC system, we have also considered the heterogeneity of MEC servers in terms of CPU capacity (in MIPS), memory size, and storage size [9].

The existing literature on task offloading in MEC has focused on various objectives, including energy optimization, latency optimization, or both, with many works adopting independent task offloading strategies to minimize UE energy consumption and application latency. Very recently, in [4], the authors utilize dependent task modeling to optimize energy consumption and task latency, but they do not account for the latency deadline in their approach. Additionally, the consideration of DVFS-enabled UE devices has been largely overlooked in the literature, despite its potential to significantly reduce energy consumption by adjusting the CPU frequency, which is particularly beneficial for energy-constrained devices like battery-powered devices [22].

In this study, we embrace a comprehensive approach to executing applications within the cooperative heterogeneous MEC framework, aiming to minimize both the energy consumption of UEs and the overall latency. This is achieved by considering dependent task modeling. We summarize the contribution of this paper below.

1. We consider DVFS-enabled UE devices in our cooperative heterogeneous system model.
2. We put forth a series of strategies to efficiently execute applications within an MEC framework, optimizing both the energy consumption of the UE devices and the latency of the applications while considering deadline-aware dependent task models.
3. Extensive simulations based on real-world datasets were conducted using a standard simulator.

The paper is structured as follows: Sect. 2 presents the literature review. In Sect. 3, we introduce the cooperative heterogeneous system model considered in this work. The application model, energy consumption model of UEs, communication model, and computation model are presented in Sect. 4, 5, 6, and 7 respectively. Section 8 provides a detailed problem formulation. Our solution for the latency and energy optimization problem is described in Sect. 9. The simulation and evaluation of our proposed strategy are presented in Sect. 10, and finally, we conclude the paper in Sect. 11.

2 Literature Review

In recent years, computation offloading in Mobile Edge Computing (MEC) has attracted significant attention from researchers, leading to the proposal of various task offloading schemes. Among the crucial objectives of these schemes is the energy-latency tradeoff in MEC server tasks. Different authors have approached the task offloading problem, formulating it as energy optimization [12,27,30], latency optimization [19,25,28], or jointly optimizing both energy and latency [2,8,26].

Various authors have explored diverse system models as part of their task-offloading schemes. Based on existing literature, we can broadly categorize these models into different architectures: two-tier [6,15,26], three-tier [17,27,30], four-tier [2,5], and SDN-based [1] architectures. In the two-tier architecture, UE is at the first layer, MEC servers attached to a Base Station (BS) are at the second layer, and the remote cloud is absent. In the three-tier architecture, the remote cloud forms the third layer. The four-tier architecture includes an additional layer, where UEs communicate with the MEC server via an access point or an edge controller. In the SDN-based architecture, a centralized control plane acts as the backbone of the entire network.

In task offloading schemes, tasks can possess various properties and may be offloaded wholly or partially. While some existing works primarily consider input data size and CPU cycle requirements for task formulation [12,26], others also take into account task latency deadline and output size [27]. The literature often aims to optimize either energy consumption or latency, leading to the use of binary or partial offloading schemes. Some works focus on optimizing both energy and latency but still adopt a binary offloading model. When representing subtasks for partial offloading, most authors employ Directed Acyclic Graphs, while others divide tasks into multiple fractions.

Various authors have employed diverse methodologies to formulate and tackle the task offloading problem. Many of them have expressed the task offloading scheme as a mixed-integer program problem and demonstrated its NP-hardness. To solve the optimization problem, a significant number of authors have transformed the nonconvex nature of the problem into a convex one, achieved either by decomposing the original problem or without decomposing it. For solving the convex problem, different approaches have been utilized, such as Karush-Kuhn-Tucker (KKT) conditions, 0–1 integer programming problem, Lagrange dual decomposition, and the subgradient method. Additionally, several alternative methods, including greedy algorithms, approximate solutions, machine learning, heuristics, genetic algorithms, artificial fish swarm algorithms, and ϵ-bounded approximate algorithms, among others, have been employed to address these optimization challenges.

3 System Model

We are adopting a three-tier architecture for our MEC system, as depicted in Fig. 1. In this architecture, UE devices reside in the first layer, a cluster of base stations (DC) in the second layer, and a remote cloud operates in the third layer. To effectively manage the energy consumption of UE devices and adhere to task latency deadlines, we are incorporating DVFS-enabled UE devices [22], which enable us to control the CPU frequency [16]. The DCs in each of the BS that are part of a cluster cooperate with each other. Additionally, we have also considered the heterogeneity of MEC servers in terms of CPU capacity (in

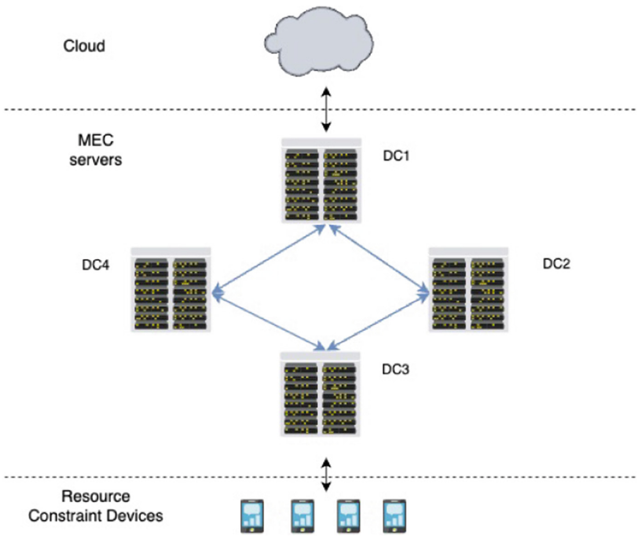

Fig. 1. System model

MIPS), memory size, and storage size. The UEs establish connections with the nearest base station through wireless communication links. These UE devices operate independently, giving them the flexibility to execute tasks locally or offload them to a remote server hosted in the base station. Consequently, MEC servers within these base stations receive tasks continuously from various UE devices.

The decision to adopt a 3-tier cooperative distributed architecture is grounded in two key factors. Firstly, the limited resource capacity of MEC servers compared to the central cloud necessitates a cooperative approach. Secondly, the increasing task sensitivity to latency in modern applications highlights the significance of an optimized cooperative architecture. By utilizing the resources of neighbouring MEC servers, we aim to meet latency deadlines, as the central cloud is often located far from the UE device. In situations where MEC servers in the nearest or neighbouring base stations cannot meet a task's resource requirements, we route the task to the cloud, assuming it possesses abundant computational resources. This 3-tier architecture achieves a balance between resource constraints, latency sensitivity, and resource availability, ensuring efficient task processing within latency requirements while optimizing resource utilization.

In each data center (BS), one of the nodes performs the coordinator function known as the coordinator node. The coordinator node ensures high availability, while the container for task execution is dynamically launched as needed. Periodically, every coordinator node updates its resource status to all other coordinators within the cluster. As part of its responsibilities, a coordinator node schedules a task to a compute node within its base station (DC) or forwards the task to another base station (DC). If a task cannot be executed in MEC, it will be forwarded to the remote cloud through a backhaul network. The set of base stations and UEs is denoted as B_n and W_n, respectively, with $n = 1, 2, 3, \cdots, n$.

4 Application Model

We consider an application comprising interdependent tasks, represented by a Directed Acyclic Graph (DAG) as the DAG representation is widely used in the literature to model various applications such as cognitive assistance [13], healthcare [11], data analytics [24] etc. In the DAG, each node represents a task, while the edges depict the dependencies between tasks. Each task in the DAG is characterized by a quadruplet $T_i = <d_i, c_i, IO_i, t_i^d>$, where d_i denotes the input data size, c_i indicates the CPU cycle requirement in million instructions (MI), IO_i represents the number of IO operations needed for the task, and t_i^d denotes the latency deadline. The application comes with a latency deadline of L_k^d. This deadline is distributed among all the tasks of that application (as explained in Sect. 9.2). To ensure a single terminal node in the application DAG when there are multiple terminal tasks, we create a dummy terminal task that depends on all the existing terminal nodes. This consolidation results in a single terminal node in each application DAG.

A scheduling plan for the DAG G is represented as $D_{i:n} = \delta1, \delta2, \delta3, \cdots, \delta_n$, where $n = |V|$ is the number of tasks, and δ_i indicates the offloading decision for task T_i. The decision variable $\delta_i = \{0, 1, 2, 3\}$, where 0 for local, 1 for the nearest BS, 2 for the neighbor BS, and 3 for the remote cloud execution. A task can be forwarded to another base station only once.

5 Energy Consumption Model for a UE

We adopt a power consumption model based on complementary metal-oxide semiconductor (CMOS) logic circuits [18] for the UE devices. Specifically, we focus on CMOS circuits-based UE devices, where the total power consumption consists of two main components: static and dynamic power consumption. Considering that dynamic power consumption significantly dominates the overall power usage [10,16], we concentrate solely on it. The dynamic power consumption is directly related to the supply voltage and frequency, and since the frequency usually scales with the supply voltage, the processor's dynamic power consumption is expressed as $\rho_c = K.f^3$, where K is a proportional coefficient.

Let us consider a task in an application that takes an execution time of t when running on a CPU with a frequency of f_{max}. If the processor operates at a different frequency level f $(0 < f \leq f_{max})$, the execution time is defined as $t/\frac{f}{f_{max}}$. Thus, the dynamic power consumption during the task execution is defined by Eq. (1) as considered in [16].

$$E = \int_0^{t/\frac{f}{f_{max}}} \rho_c \, dt = K.t.f_{max}.f^2 = \omega.t.\lambda^2, \; where \; \lambda = \frac{f}{f_{max}} \tag{1}$$

where ω is a coefficient and λ is the relative processor speed for the CPU while running at frequency f.

6 Communication Model

Let B_w be the bandwidth between UE devices and the base station, and B_b be the bandwidth between two base stations. The backhaul network, facilitating task forwarding from a base station to the remote cloud, has a bandwidth of β_c. For wireless and wired links, we consider data rates R_u and R_p, respectively.

Let the latency to upload a task T_i from UEs to the nearest BS be T_i^u. The total latency T_i^{nb} for offloading a task to a neighbour base station will be the combination of task uploading latency T_i^u and the latency of forwarding a task to another BS. Similarly, T_i^{rc} is the latency of a task T_i when offloaded to the centralized remote cloud. It is the combination of latency of uploading tasks from UEs to BS and from BS to the remote cloud.

7 Computation Model

The task offloading decisions determine whether a task is executed locally at the UEs, at the Mobile Edge Computing (MEC) server, or at the remote cloud. Various computation models based on the task's offloading decision are elaborated in the following subsections.

7.1 Local Computation

During local task execution, the task utilizes the UE's local processing unit. Let f_l^k denote the computational capacity of the kth UE in million instructions per second (MIPS). Let IO_i^t be the time required for IO operations then local task execution time and the UE energy consumption for task T_i can be represented as $T_i^l = \frac{c_i}{f_l^k} + IO_i^t$ and $E_i^l = \alpha . T_i^l . \lambda^2$ respectively.

7.2 Remote Computation

When a task is offloaded by a UE device, it can be executed either in a MEC server or a remote central cloud. During task offloading, the UE device utilizes its processing unit to transfer the tasks to a remote server. As a result, the energy consumption of the UE is influenced by the CPU cycle required to upload the tasks to the remote server. Let T_i^{ul} represent the time taken by a UE device to offload a task to a remote server. The energy consumption for the offloaded task T_i can be expressed as $E_i^u = \alpha . T_i^u . \lambda^2$.

The latency computation model remains consistent for tasks executed at the nearest Base Station (BS), a remote BS, or in the cloud. If F_m^k and F_c^k represent the CPU capabilities of the kth MEC and the cloud servers then the latency at MEC and the cloud can be represented as in Eq. (2).

$$T_i^{mec} = \frac{c_i}{F_{m^k}} + IO_i^t \ , \ T_i^c = \frac{c_i}{F_c^k} + IO_i^t \tag{2}$$

8 Problem Formulation

The primary goal is to minimize both the energy consumption of the UEs and the overall latency of the application. The total latency of a task T_i in a DAG for the application can be formulated using Eq. (3).

$$\begin{aligned} L_i = & \frac{(1-\delta_i)(2-\delta_i)(3-\delta_i)}{6} T_i^l + \frac{\delta_i(2-\delta_i)(3-\delta_i)}{2}(T_i^u + T_i^{mec}) \\ & + \frac{d_i(\delta_i - 1)(3-\delta_i)}{2}(T_i^{nb} + T_i^{mec}) + \frac{\delta_i(\delta_i-1)(\delta_i-2)}{6}(T_i^{rc} + T_i^c) \end{aligned} \tag{3}$$

The energy consumption for the UE remains constant regardless of whether a task is executed at the nearest Base Station (BS), a neighbouring BS, or the cloud, as the UE device always offloads the task to the nearest BS. Thus, the energy consumption of a task T_i can be expressed using Eq. (4) below.

$$\begin{aligned} E_i = & \frac{(1-\delta_i)(2-\delta_i)(3-\delta_i)}{6}(E_i^l) + \frac{\delta_i(2-\delta_i)(3-\delta_i)}{2}(E_i^u) \\ & + \frac{d_i(\delta_i - 1)(3-\delta_i)}{2}(E_i^u) + \frac{\delta_i(\delta_i-1)(\delta_i-2)}{6}(E_i^u) \end{aligned} \tag{4}$$

The total latency and energy consumption of a UE for an application is represented as $L_{app} = L_{ft}$ and $E_{app} = \sum_{i=1}^{n} E_i$ respectively where the L_{ft}

is the completion time of the terminal task of an application. If there are N applications in the system, then the total energy consumption at UEs and the latency can be represented as in Eq. (5).

$$L_{tot} = \sum_{i=1}^{N} L_{app} \ , \ E_{tot} = \sum_{i=1}^{N} E_{app} \tag{5}$$

The objective of the problem is to minimize the overall system cost, encompassing the total execution delay and UE energy consumption for all applications in the system. We express the total cost as a weighted sum of the total UE energy consumption and application latency. Thus, the minimization problem is represented as shown in Eq. (6) subject to the constraint defined in Equation (7).

$$minimize \ (AL_{tot} + BE_{tot}) \tag{6}$$

$$\sum_{i=1}^{n} L_i \leq L_d \ , \ \sum_{i=1} c_i \leq F_l \ , \ c_i \leq F_m^i \ , \ \sum_{i=1}^{n} c_i \leq \sum_{i=1}^{B_i} \sum_{j=1}^{m} F_m^j \tag{7}$$

The constraints presented in Eq. (7) apply to both the UE device and the MEC servers. The first constraint ensures that the total latency of an application must not surpass its latency deadline. The second constraint ensures that the total CPU cycle requirements of all the parallel tasks executing locally must not exceed the available CPU cycles of the UE. The third constraint guarantees that the CPU requirements for a task cannot exceed the available capacity of the MEC server. Lastly, the fourth constraint ensures that the total CPU cycle requirements of all tasks running in the MEC cluster at any given time must not surpass the total CPU capacity of the MEC cluster.

9 ELITE: The Task Offloading and Resource Allocation Strategy

The optimization problem stated in Eq. (6) is a challenging multi-objective mixed-integer programming (MIP) problem, proven to be NP-hard [31]. As a result, we propose an efficient heuristic algorithmic (Layered Scheduling Algorithm) approach to tackle this optimization problem. In the proposed strategy, we first rank the tasks of an application represented by a DAG for effective scheduling, as described in Sect. 9.1. Additionally, we compute the sub-deadline of all tasks within an application's DAG, using the total deadline for the entire application, as elaborated in Sect. 9.2

9.1 Task Ranking

We need to rank the tasks for scheduling and it is helpful in prioritizing multiple tasks that can be executed in parallel. Let CT_i^{nbs}, CT_i^{rbs}, and CT_i^{c} be the latency in worst case time for a task T_i if it executes in the nearest base station, remote base station, and remote cloud server respectively. So the rank of a task is calculated based on Eq. (8);

$$S_i^{avg} = \frac{CT_i^n bs + CT_i^r bs + CT_i^c}{3}\ ,\ T_i^{avg} = \frac{CT_i^m + S_i^{avg}}{2} \qquad (8)$$
$$R(T_i) = T_i^{avg} + max(Pred_k T_k^{avg})$$

where $max(Pred_k T_k^{avg})$ represents the maximum of the average latency of all the predecessor tasks of T_i and CT_i^m is the local execution latency of task T_i.

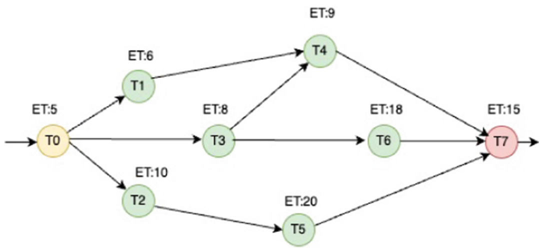

Fig. 2. An application DAG

9.2 Sub-deadline Calculation

In our approach, we divide the total latency deadline of an application and allocate sub-deadlines to its individual tasks. To achieve this, we compute the label-ratio ∂_l as in Eq. (9) and employ a modified version of the breadth-first traversal (MBFS) algorithm to assign ratios to each level in the DAG as depicted in Fig. 2. The label ratio signifies the maximum average execution time of tasks within a particular level. As tasks between levels can be executed in parallel, we determine the maximum waiting time for the next level based on the maximum latency of the previous level. Consequently, we assign equal sub-deadlines to all tasks within each level, taking into account the maximum average latency of tasks from the preceding level. The sub-deadline D_l of a task is calculated using Eq. (9), with D_{total} representing the total deadline of the application.

$$\partial_l = max_l(T_l^{avg})\ ,\ D_l = \frac{D_{total}}{\sum_{i=1}^{l} \partial_l} \partial_l \qquad (9)$$

Let's consider the instance of level L1, encompassing three tasks: T1, T2, and T3. In accordance with the data presented in Fig. 2, the average execution times for T1, T2, and T3 are 6, 10, and 8 respectively. Consequently, the label ratio for level L1 becomes $\max(6, 10, 8)$, yielding 10. Analogously, the label ratios for L0, L2, and L3 are 5, 20, and 15 respectively. Utilizing Eq. (9), the sub-deadlines for tasks T1, T2, and T3 are calculated as $\frac{100*10}{(5+10+20+15)}$, resulting in 20. It's worth noting that the total deadline for the analyzed application is denoted as 100. The full calculation of the sub-deadline for the application represented by a DAG in Fig. 2 is given in Table 1.

Table 1. Sub deadline calculation

level	Tasks	Label ratio	Sub Deadline	Total Deadline
L0	T0	5	10	100
L1	T1, T2, T3	10	20	
L2	T4, T5, T6	20	40	
L3	T7	15	30	

9.3 Task Offloadability

We have categorized the tasks of a workflow into three categories as given below.

1. Remote Tasks: A task is called a remote task if it is compute-intensive or IO-intensive as described in Algorithm 2.
2. Local Tasks: Tasks that require the UE to interact directly with the environment such as mage capturing cannot be offloaded.
3. General Tasks: The third type of tasks can be executed either in UE devices or in remote servers, and we have the flexibility to schedule these tasks based on various parameters.

9.4 Layered Scheduling Algorithm

The layered scheduling algorithm involves two levels of scheduling algorithms: one operating at the UE and the other at the coordinator node of the MEC, hosted in the nearest base station.

Scheduling Algorithm at UE. The scheduling algorithm, outlined in Algorithm 1, handles the execution of an application on the UE device, that includes multiple tasks. The algorithm makes decisions to either schedule these tasks locally or offload them to the MEC server hosted in the nearest base station. Three queues are maintained in the algorithm: TaskPool, TaskReadyQueue, and TaskInProgress. Initially, the TaskPool contains all tasks (S_t) except for the entry task (T_{entry}) of the applications, while the entry tasks of all running applications at the UE are added to the TaskReadyQueue. The algorithm then selects one task from the TaskReadyQueue and schedules it either locally or offloaded to a remote server. Once a task is scheduled, whether locally or remotely, it is added to the TaskInProgress. The steps in the algorithm are summarized below.

Algorithm 1. Task Scheduling Algorithm at UE device

```
1: TaskInProgressl← $\phi$
2: TaskPool← $S_t - T_{entry}$
3: $TaskReadyQueue \leftarrow T_{entry}$
4: while (TaskPool.size() > 0 or TaskReadyQueue.size() > 0 or TaskInProgress.size()> 0) do
5:     while $TaskReadyQueue.size() > 0$ do
6:         $Task_i \leftarrow$ dequeue from $TaskReadyQueue$
7:         if $Task_i$ $is\ remote$ then
8:             Offload the task to the nearest BS.
9:         else if $Task_i$ $is\ local$ then
10:            Schedule the task for local execution
11:        else
12:            $ET_i \leftarrow$ Worst-case execution time of the task in the MEC server.
13:            if $ET_i \leq$ Task deadline then
14:                Offload the tasks to the nearest BS.
15:            else
16:                Schedule the task for local execution
17:            end if
18:        end if
19:        TaskInProgress.$push(Task_i)$
20:    end while
21:    for $Task_j$ in $TaskInProgress$ do
22:        if $Task_j$ $is\ finished\ executing$ then
23:            $ChildrenTasks_j \leftarrow$ get all children tasks of $Task_j$
24:            Sort the $ChildrenTasks_j$ based on the descending order of its rank
25:            for $k \leftarrow 0$ to $ChildrenTasks_j.size()$ do
26:                $child_k \leftarrow ChildrenTasks_j.get(i)$
27:                if All parent tasks of $child_k$ is finished executing then
28:                    $TaskReadyQueue.Enqueue(child_k)$
29:                    TaskPool.$Remove(child_k)$
30:                end if
31:            end for
32:            TaskInProgress.$remove(Task_j)$
33:        end if
34:    end for
35: end while
```

1. A task is offloaded to the nearest BS if it is a remote task.
2. A local task is scheduled for local execution.
3. A general tasks: Offload the task to the MEC server if the worst-case execution time of the task in the MEC server is less than the latency deadline or else schedule for local execution.

Upon completion of a task's execution, we iterate through all its child tasks. If all the parent tasks of a child task have also finished, we add the child tasks to the ReadyQueue and remove the current from the TaskPool and TaskExecutionProgressQueue.

Algorithm 2. Task Categorization Algorithm

```
Input: CPU time($CT_i$), IO time ($IO_i$)
Output: Task category
$T_{tot} \leftarrow CT_i + IO_i$
if $\frac{CT_i}{T_{tot}} \geq 0.5$ then
    Return CPU-intensive
else if $\frac{IO_i}{T_{tot}} \geq 0.5$ then
    Return IO-intensive
else
    Return Normal
end if
```

Scheduling at MEC. The algorithm presented in Algorithm 3 is responsible for scheduling tasks to one of three locations: locally (nearest BS), neighbour BS, or a remote cloud. The coordinator node selects tasks from its task queue and checks if there is a suitable server in the current DC to execute the task. If a suitable server is found, the task is scheduled to the selected server. If no suitable server is available in the current DC, the coordinator node looks for a suitable server in the neighbouring DC. To determine the appropriate neighbour DC, we sort the neighbour DCs based on Euclidean distance and try to find the best-suited server, starting with the closest neighbour. If no suitable server is present in the MEC cluster, the task is forwarded to the central cloud. To find the best server in a particular DC we have used the following steps:

1. Enumerate the servers that have available CPU cores and a task latency less than the task deadline. If the list is not empty, return the server that executes a task with minimum latency.
2. Enumerate the servers where the task completion time is within the task deadline limit. If the list is not empty, return the server that executes a task with minimum latency.

Algorithm 3. Task Scheduling Algorithm at MEC coordinator node

```
function $GetFeasibleServer(serverList, Task_i)$
    for server in $serverList$ do
        $latency \leftarrow$ calculateLatency(server, $Task_i$)
        if $latency <=$ task deadline then
            return server
        end if
    end for
end function
function $GetBestServer$(Datacenter dc, $Task_i$)
    $svrsWithFreeCore \leftarrow$ get all servers with free available cores.
    $server \leftarrow GetFeasibleServer(svrsWithFreeCore, Task_i)$
    if $server$ != $\phi$ then
        return server
    end if
    $otherServers \leftarrow$ get all servers with a free available core.
    $server \leftarrow GetFeasibleServer(otherServesr, Task_i)$
    if $server$ != $\phi$ then
        return server
    end if
end function
dc $\leftarrow$ nearest DC
$server \leftarrow GetBestServer(dc, Task_i)$
if $server \ ! = \phi$ then
    $scheduleTask(Task_i, server)$
else
    $neighborDCs \leftarrow$ get all the neighbour MEC DCs in the cluster
    $neighborDCs \leftarrow$ sort $neighborDCs$ on the distance from the current DC
in ascending order.
    for $dc$ in $neighborDCs$ do
        $server \leftarrow GetBestServer(dc)$
        if $server \ ! = \phi$ then
            $scheduleTask(Task_i, server)$
            Break
        end if
    end for
end if
```

10 Simulation and Result Analyses

Our proposed strategy was simulated using the PureEdgeSim simulator [21]. The simulation area covered a 2000×2000 square meter space, containing three edge data centers (DCs), each with three physical servers. We incorporated three types of edge devices in the simulation, namely smartphones, Raspberry Pi, and laptops. Each edge device is connected to its nearest edge DC based on Euclidean distance. To facilitate the simulation, we utilized the Zenodo dataset [3], which comprises 50,000 jobs containing 1.3 million tasks. This dataset represents various applications represented by DAGs generated by IoT nodes.

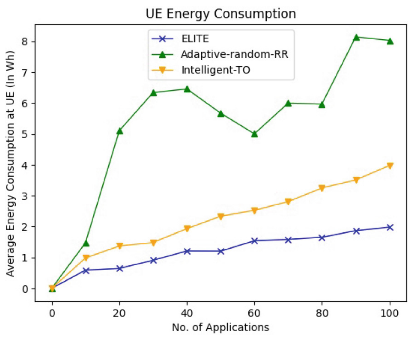

Fig. 3. UE Energy Consumption

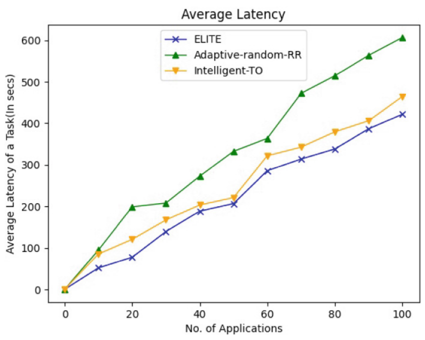

Fig. 4. Average Task Latency

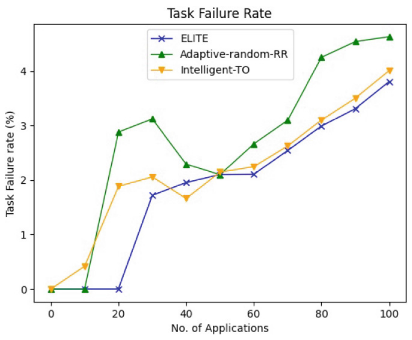

Fig. 5. Task Failure Rate

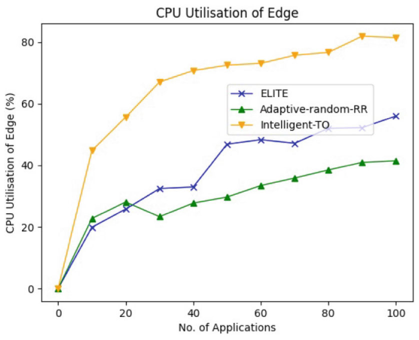

Fig. 6. Edge CPU Utilization

In order to assess the performance of our proposed layered algorithm, we conducted a benchmark comparison with two existing algorithms: “Intelligent task offloading for dependent tasks” presented by Chen et al. in [4] and the “Adaptive Random Round Robin Algorithm” described in Sect. 10.1.

10.1 State-of-Art Approach

1. **Intelligent task offloading for dependent tasks** [4]**:** In their work, researchers propose an energy and latency-optimized task offloading strategy for dependent tasks. They utilize a two-step process: organizing tasks

into layers based on dependencies using a layered algorithm and employing a cost function that considers task latency and UE device energy consumption for making offloading decisions. This approach allows for efficient task execution, choosing between local execution and offloading to a remote server based on the most favourable cost considerations.

2. **Adaptive Random Round Robin Scheme:** We adopted the same system model described in the ELITE algorithm except for the DVFS-enabled UE device. We offloaded all CPU-intensive and IO-intensive tasks to the MEC server, while normal tasks were randomly scheduled for either MEC execution or local execution. For MEC DC selection, we employed a Round-Robin algorithm, and the server selection within the DC was done randomly. The task categorization is the same as the ELITE algorithm.

Table 2. Simulation Parameters

Iteration	Applications	Total Tasks	UE devices
1	10	481	5
2	20	903	10
3	30	1314	15
4	40	1746	20
5	50	2194	25
6	60	2629	30
7	70	2973	35
8	80	3387	40
9	90	3855	45
10	100	4294	50

10.2 Results and Analyses

We performed the experiment using the parameters specified in Table 2. In Fig. 3, we plotted the average energy consumption of UE devices, which increases as the number of applications rises. This behaviour is attributed to the growing number of tasks executed on UE devices. Notably, our proposed layered algorithm outperforms the state-of-the-art by a factor of 2× in terms of UE energy consumption. This improvement can be attributed to two main factors: Firstly, our system model incorporates dynamic voltage and frequency scaling (DVFS) in UE devices, and secondly, our offloading decision algorithm demonstrates superior performance compared to others.

In Fig. 4, we presented the average latency of a task, which shows an increasing trend with the growing number of tasks. This increase is primarily due to the longer queue times experienced by both UE and MEC servers as the task load rises. Notably, our proposed model exhibits approximately 10% lower latency than the state-of-the-art algorithm. This improved performance can be attributed to two key factors: Firstly, our superior task offloading algorithm, and secondly, the resource allocation algorithm we introduced, which effectively reduces task latency by minimizing task waiting times.

As shown in Fig. 5, the task failure rate increases with the number of tasks, as the queue time for each task also increases, leading to task failures due to higher latency, considering the latency deadline of each task. In this aspect, our proposed layered algorithm outperforms the benchmarking algorithm. The CPU utilization of the MEC server is also plotted in Fig. 6, and it increases with the number of tasks. The results indicate that the average edge CPU utilization of the state-of-the-art model is approximately 55% higher than that of our proposed algorithm.

11 Conclusion

This research paper addresses the challenge of minimizing the latency of the applications and energy consumption of UE devices by employing dependent task modelling. Our proposed 3-tier system model incorporates a cluster of heterogeneous MEC servers in the MEC layer. In our model, we have considered DVFS-enabled UE devices, which aid in reducing their energy consumption by dynamically adjusting the CPU operating frequency. To optimize the problem, we formulate it as a bi-objective mixed integer programming (MIP), which is known to be NP-hard. To overcome this complexity, we introduce a near-optimal heuristic solution strategy that efficiently addresses both task offloading and resource allocation problems. Tasks are classified into local, remote, and general categories, with further subcategories based on their characteristics such as CPU-intensive, IO-intensive, and normal tasks. Our simulation results demonstrate the superior efficiency and performance of our approach compared to existing benchmarking algorithms.

References

1. Alameddine, H.A., Sharafeddine, S., et al.: Dynamic task offloading and scheduling for low-latency IoT services in multi-access edge computing. IEEE J. Sel. Areas Commun. **37**(3), 668–682 (2019)
2. Alfakih, T., Hassan, M.M., et al.: Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on SARSA. IEEE Access **8**, 54074–54084 (2020)
3. Ali Rezaee, S.A.: Jobs (DAG workflow) and tasks dataset with near 50k job instances and 1.3 millions of tasks (2020). https://doi.org/10.5281/zenodo.4667690
4. Chen, J., Leng, Y., Huang, J.: An intelligent approach of task offloading for dependent services in mobile edge computing. J. Cloud Comput. **12**(1), 1–14 (2023)

5. Chen, J., Chang, Z., et al.: Resource allocation and computation offloading for multi-access edge computing with fronthaul and backhaul constraints. IEEE Trans. Veh. Technol. **70**(8), 8037–8049 (2021)
6. Chouhan, S.: Energy optimal partial computation offloading framework for mobile devices in multi-access edge computing. In: International Conference on Software, Telecommunications and Computer Networks (SoftCOM), pp. 1–6. IEEE (2019)
7. Deka, V., Islam, A., Ghose, M.: Cloud-assisted dynamic and cooperative content caching in mobile edge computing. In: IEEE 19th India Council International Conference (INDICON), pp. 1–6 (2022)
8. Dinh, T.Q., Tang, J., et al.: Offloading in mobile edge computing: task allocation and computational frequency scaling. IEEE Trans. Commun. **65**(8), 3571–3584 (2017)
9. Ghose, M., Kaur, S., Sahu, A.: Scheduling real time tasks in an energy-efficient way using VMS with discrete compute capacities. Computing **102**(1), 263–294 (2020)
10. Ghose, M., Sahu, A., Karmakar, S.: Urgent point aware energy-efficient scheduling of tasks with hard deadline on virtualized cloud system. Sustain. Comput.: Inform. Syst. **28**, 100416 (2020)
11. Gia, T.N., Jiang, M., et al.: Fog computing in healthcare internet of things: a case study on ECG feature extraction. In: IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing, pp. 356–363 (2015)
12. Guo, H., Liu, J., Zhang, J.: Computation offloading for multi-access mobile edge computing in ultra-dense networks. IEEE Commun. Mag. **56**(8), 14–19 (2018)
13. Ha, K., Chen, Z., et al.: Towards wearable cognitive assistance. In: Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys 2014, pp. 68–81. Association for Computing Machinery, New York (2014)
14. Islam, A., Debnath, A., et al.: A survey on task offloading in multi-access edge computing. J. Syst. Architect. **118**, 102225 (2021)
15. Ji, T., Luo, C., et al.: Energy-efficient computation offloading in mobile edge computing systems with uncertainties. IEEE Trans. Wirel. Commun. **21**, 5717–5729 (2022)
16. Kim, K.H., Beloglazov, A., et al.: Power-aware provisioning of virtual machines for real-time cloud services. Concurr. Comput.: Pract. Exp. **23**(13), 1491–1505 (2011)
17. Kuang, Z., Ma, Z., et al.: Cooperative computation offloading and resource allocation for delay minimization in mobile edge computing. J. Syst. Architect. **118**, 102167 (2021)
18. Lee, Y.C., Zomaya, A.Y.: Energy conscious scheduling for distributed computing systems under different operating conditions. IEEE Trans. Parallel Distrib. Syst. **22**(8), 1374–1381 (2010)
19. Liao, Z., Peng, J.O.: Adaptive offloading in mobile-edge computing for ultra-dense cellular networks based on genetic algorithm. J. Cloud Comput. **10**(1), 1–16 (2021)
20. Liu, B., Xu, X., et al.: Task scheduling with precedence and placement constraints for resource utilization improvement in multi-user MEC environment. J. Syst. Architect. **114**, 101970 (2021)
21. Mechalikh, C., Taktak, H., Moussa, F.: PureEdgeSim: a simulation framework for performance evaluation of cloud, edge and mist computing environments. Comput. Sci. Inf. Syst. **18**(1), 43–66 (2021)

22. Mokaripoor, P., Hosseini Shirvani, M.: A state of the art survey on DVFs techniques in cloud computing environment. J. Multidiscip. Eng. Sci. Technol **3**(5), 4740–4743 (2016)
23. Ranaweera, P., Jurcut, A.D., Liyanage, M.: Realizing multi-access edge computing feasibility: security perspective. In: IEEE Conference on Standards for Communications and Networking (CSCN), pp. 1–7. IEEE (2019)
24. Reza, H., Diyanat, A., et al.: MIST: fog-based data analytics scheme with cost-efficient resource provisioning for IoT crowdsensing applications. J. Netw. Comput. Appl. **82**, 152–165 (2017)
25. Song, F., Xing, H., et al.: Offloading dependent tasks in multi-access edge computing: a multi-objective reinforcement learning approach. Futur. Gener. Comput. Syst. **128**, 333–348 (2022)
26. Tran, T.X., Pompili, D.: Joint task offloading and resource allocation for multi-server mobile-edge computing networks. IEEE Trans. Veh. Technol. **68**(1), 856–868 (2018)
27. Vu, T.T., Van Huynh, N., et al.: Offloading energy efficiency with delay constraint for cooperative mobile edge computing networks. In: 2018 IEEE Global Communications Conference (GLOBECOM), pp. 1–6. IEEE (2018)
28. Wang, J., Hu, J., Min, G., et al.: Computation offloading in multi-access edge computing using a deep sequential model based on reinforcement learning. IEEE Commun. Mag. **57**(5), 64–69 (2019)
29. Wang, L., Deng, X., et al.: Microservice-oriented service placement for mobile edge computing in sustainable internet of vehicles. IEEE Trans. Intell. Transp. Syst. (2023)
30. Yu, H., Wang, Q., Guo, S.: Energy-efficient task offloading and resource scheduling for mobile edge computing. In: 2018 IEEE International Conference on Networking, Architecture and Storage (NAS), pp. 1–4. IEEE (2018)
31. Zhang, J., Liu, C., et al.: A survey for solving mixed integer programming via machine learning. arXiv preprint arXiv:2203.02878 (2022)