electronics

Article

Offloading Decision and Resource Allocation in Mobile Edge Computing for Cost and Latency Efficiencies in Real-Time IoT

Chanthol Eang [1,†], Seyha Ros [1,†], Seungwoo Kang [1], Inseok Song [1], Prohim Tam [1], Sa Math [2] and Seokhoon Kim [1,3,*]

[1] Department of Software Convergence, Soonchunhyang University, Asan 31538, Republic of Korea; ngchanthol1@gmail.com (C.E.); rosseyha003@gmail.com (S.R.); oooksw12@sch.ac.kr (S.K.); sis5041@sch.ac.kr (I.S.); prohimtam@gmail.com (P.T.)
[2] Department of Telecommunication and Electronic Engineering, Royal University of Phnom Penh, Phnom Penh 12156, Cambodia; math.sa@rupp.edu.kh
[3] Department of Computer Software Engineering, Soonchunhyang University, Asan 31538, Republic of Korea
* Correspondence: seokhoon@sch.ac.kr
† These authors contributed equally to this work.

Abstract: Internet of Things (IoT) devices can integrate with applications requiring intensive contextual data processing, intelligent vehicle control, healthcare remote sensing, VR, data mining, traffic management, and interactive applications. However, there are computationally intensive tasks that need to be completed quickly within the time constraints of IoT devices. To address this challenge, researchers have proposed computation offloading, where computing tasks are sent to edge servers instead of being executed locally on user devices. This approach involves using edge servers located near users in cellular network base stations, and also known as Mobile Edge Computing (MEC). The goal is to offload tasks to edge servers, optimizing both latency and energy consumption. The main objective of this paper mentioned in the summary is to design an algorithm for time- and energy-optimized task offloading decision-making in MEC environments. Therefore, we developed a Lagrange Duality Resource Optimization Algorithm (LDROA) to optimize for both decision offloading and resource allocation for tasks, whether to locally execute or offload to an edge server. The LDROA technique produces superior simulation outcomes in terms of task offloading, with improved performance in computation latency and cost usage compared to conventional methods like Random Offloading, Load Balancing, and the Greedy Latency Offloading scheme.

Keywords: offloading decision; Lagrange duality optimization; mobile edge computing; real-time IoT; resource allocation

Citation: Eang, C.; Ros, S.; Kang, S.; Song, I.; Tam, P.; Math, S.; Kim, S. Offloading Decision and Resource Allocation in Mobile Edge Computing for Cost and Latency Efficiencies in Real-Time IoT. *Electronics* **2024**, *13*, 1218. https://doi.org/10.3390/electronics13071218

Academic Editor: Bahman Javadi

Received: 15 February 2024
Revised: 17 March 2024
Accepted: 24 March 2024
Published: 26 March 2024

1. Introduction

Mobile Edge Computing (MEC) is a computing methodology that can work on the cloud's resource in terms of providing application and content closer to users at the edge network within the Radio Access Network (RAN) [1]. Some applications, such as Augmented Reality (AR), real-time gaming, Virtual reality (VR), and remote sensing, can immediately process data for data offloading and downloading by user devices [2,3]. Nowadays, we are moving towards technologies of smart healthcare, such as Internet-of-Medical-Things (IoMTs) or Sensor Nodes (SNs). Those technologies need SNs and edge computing to enable an efficient healthcare system, which requires real-time health monitoring using medical care applications on wearable gadgets or cellphones. Sensors that are equipped on the IoMT devices generate enormous amounts of data, and those data are transported to the edge server or cloud computing server, where intensive data processing is conducted. Finally, the health information is sent back to end users (e.g., doctors, nurses, and patients) for further disease analysis [4]. To overcome the high-latency issue, the edge computing paradigm has been used, where in the edge environment extends the cloud environment and caters

to IoMT applications at the edge network [5,6]. Edge computing is equipped with limited capacity of computation resources and batteries based on its physical size, and it can be deployed across the edge of the networks [7]. To ensure reliable SN performance for real-time health monitoring and long battery lifetime, an AI-enabled energy management technique is required to reduce the overall energy consumption of SNs and improve their battery lifetime [8]. Currently, the solutions being considered to handle the battery lifetime of SNs are to offload the computation-intensive tasks to a higher-level infrastructure, known as edge computing. However, task offloading has become a research trend in many prior works, focusing on when and where to offload the computation-intensive workload of SNs to the edge network or cloud [9]. MEC servers can be deployed with Roadside Units (RSUs) for Internet of Vehicles (IoV), where computing resources are available for tasks offloaded from user devices [10–13]. These applications also require significant computation resources and must adhere to strict time constraints [14,15]. The computing tasks or data gathered by user devices have a limited upper-bound delay, and exceeding this delay can lead to task failures. Existing studies in the field of computation offloading primarily focus on offloading decisions, determining which tasks should be executed locally or offloaded to the edge server [16–19]. Researchers consider various attributes of tasks, such as computational intensity, data dependency, deadline constraints, and sensitivity to network latency. These characteristics influence the decision-making process regarding whether a task is suitable for offloading and where it should be offloaded. Different optimization objectives may guide offloading decisions, such as minimizing latency, conserving energy, maximizing resource utilization, or improving Quality of Service (QoS). Researchers explore various trade-offs between these objectives and develop algorithms that balance conflicting goals effectively. Existing studies in the field of computation offloading aim to develop intelligent decision-making mechanisms that optimize the allocation of computational tasks across distributed computing resources, thereby improving system performance, efficiency, and user experience. Due to this challenge, we also aim to improve the performance of user devices to be effectively utilized in edge computing systems by introducing the resource optimization method via Lagrangian duality optimization and integrating it with the newly developed offloading decision by assigning priority to saving computing power on each user device as well as considering the time constraints of IoT applications utilized in our model.

Research Problem and Contribution

The energy required for processing IoT applications on user devices is limited by the capacity of batteries, which are constrained by their life cycle. Offloading tasks is an enhanced technology that helps mitigate the limited energy problem for user devices. While edge computing allows tasks to be offloaded to edge servers instead of edge clouds, the rapid expansion of computation-intensive tasks (such as IoMTs, AR, VR, and autonomous driving) can lead to resource congestion on MEC servers and lack of autonomy management, significantly impacting task execution latency and privacy preserving data [20–22]. Thus, smart offloading decision and resource optimization methods are required to be utilized to improve the performance of computation offloading [23–25].

In our research, we focus on offloading decisions and resource optimizing to minimize energy consumption and task computation latency, as well as the computation cost in 5G networks. We use the Lagrange algorithm to adjust the optimal resources for each task to execute on MEC servers. The main contributions of this work are described below.

- The proposed Lagrange Duality Resource Optimization Algorithm (LDROA) has a unique workflow because the algorithm can make intelligent offloading decisions and optimize resources for tasks on edge servers, thereby reducing the total cost and total latency. This method guarantees that the collective resources of the MEC servers adequately fulfills the task demands and time limitations, all the while conserving energy on the user devices. This method can achieve superior performance by reducing energy consumption in user devices and achieving optimal latency for IoT applications.

- The joint resource allocation and offloading decision can minimize the total computing overhead of tasks, including completion time, according to time constraints of IoT applications. The proposed approach aims to provide benefits for user device energy consumption, which can increase battery lifetime without the need for frequent charging. LDROA can solve the problem well by providing smart offloading decision and optimal resource allocation for tasks offloaded to MEC.
- Our optimization technique enables the conversion of a constrained optimization problem into an unconstrained one by introducing Lagrange multipliers to form a dual problem, which allows for the optimization of the minimization to lower bound for energy, latency, and cost. The simulation setup is executed for multiple congestion conditions with optimal performance in terms of total latency and costs.
- The algorithm demonstrates superior performance compared to existing methods, such as Random Offloading, Load Balancing, and Greedy Latency scheme, particularly in terms of reducing total latency and achieving cost savings. It pursues a dual optimization strategy, aiming to simultaneously address both latency reduction and energy consumption efficiency. By considering these two crucial aspects, the algorithm ensures not only improved system responsiveness, but also enhanced energy efficiency across the network.

The remainder of this paper is structured as follows. Section 2 presents the related work concerning offloading decision and resource allocation. Section 3 presents the proposed methodology, which covers system architecture and system model. The proposed solution flowchart and algorithm are also included in Section 3. Section 4 presents the simulation settings, parameters, and the existing schemes. Section 5 explains the experimentation of the proposed LDROA algorithm, simulation setting, and its result. Finally, a conclusion of our work contributions and future work are presented in Section 6.

2. Related Works

MEC allows for devices to offload tasks to a nearby edge without transmitting to or from a cloud, which can decrease the utilization of resource, bandwidth in the backhaul, and computing power from user devices [26]. MEC optimizes resource allocation, minimizes latency, and enhances the overall efficiency of the network ecosystem. This approach not only streamlines data processing but also mitigates the potential bottlenecks associated with centralized cloud computing, resulting in a more responsive and agile network infrastructure. With the continuous advancement of cloud and edge computing models, computation offloading has been embraced as a method to boost the computing capacity of user devices [27]. Resource allocation involves the scheduling of resources to determine the most effective pairing of available resources while meeting all of the users' requirements. Resources are assigned based on factors such as the number of requests in the system, the type of request, its priority, and the acceptable delay for each request [28]. Resource allocation is a critical aspect of network management, encompassing the strategic scheduling and allocation of resources to ensure optimal utilization while satisfying the diverse requirements of users. This process involves intelligently pairing available resources, such as computing power, memory, storage, and network bandwidth, with the tasks or services requested by users. By carefully orchestrating these allocations, resource allocation mechanisms aim to maximize efficiency, minimize latency, and enhance the overall performance of the system. It must consider various factors, including the characteristics of tasks or services, the capabilities of available resources, Quality of Service (QoS) requirements, user preferences, and system constraints. Balancing these considerations is essential to achieve a harmonious allocation that meets the diverse needs of users while maintaining the stability and reliability of the network. In recent years, numerous studies have investigated the utilization of edge offloading for applications that require minimal delay. In [29], the proposed cloudlet framework, which includes the device, cloudlet, and cloud layers, created a three-layer approach to task offloading. High-computing tasks are offloaded to the cloudlet and cloud layers, while low-computing

and high-communication cost tasks are completed on the device layer. By doing so, the framework avoids transmitting large amounts of data to the cloud, resulting in a reduction in processing delay. This reduction in data transmission not only minimizes processing delays but also alleviates network congestion, conserves bandwidth, and enhances overall system responsiveness. Additionally, by leveraging edge computing resources for local data processing and analysis, the framework facilitates faster decision-making and real-time responsiveness, particularly in latency-sensitive applications. In [30], the Energy-Efficient Dynamic Offloading Algorithm (EEDOA) was proposed to optimize energy consumption, which can be implemented in real time to make task offloading decisions with polynomial time complexity. Theoretical analysis has been conducted to demonstrate that the EEDOA can approximate the minimum transmission energy consumption while also bounding the queue length. In [31], the proposed approach aims to jointly minimize the system energy consumption and maximize the number of offloaded tasks. For solving this problem, the authors combined the Ant Colony Optimization (ACO) algorithm with load balancing and proposed a Load Balancing ACO (LBA) algorithm. Numerical results show that the proposed algorithm outperforms the traditional algorithms such as greedy algorithm and distance first algorithm.

In [32], the authors introduced a new approach for deciding task offloading in MEC using deep reinforcement learning. The decision-making process for task offloading is represented as a Markov Decision Process (MDP). The primary goal is to minimize the combined impact of offloading delay and power usage. This goal is divided into rewards for each time slot. Next, in [33], the author proposed an algorithm based on Deep Deterministic Policy Gradient (DDPG) combined with replay memory strategy. This approach aims to achieve a flexible offloading decision plan and effective coordination of wireless resources between the backhaul and fronthaul. On the other hand, the joint computation offloading and resource allocation-based MEC computing systems are utilization schemes to handle the multiple task computing paradigms. In [34], a NOMA-based MEC system was considered for facilitation in terms of multiple computing tasks from user devices. Heuristic algorithms have been proposed to solve the problem, where the aims to trouble the computation allocation for NOMA-based MEC servers and time allocation are optimized by bisection search allocation. With that, in [35], since massive devices in IoT typically have a limitation resource, the authors proposed a novel NOMA scheme, optimal to handle the low-complexity sub-option to the total offloading throughput maximum. However, a NOMA was proposed that combines the merits of Time-Division Multiple Access (TDMA) and traditional approach NOMA. Therefore, based on the proposed NOMA scheme, the problem of maximizing the total offloading throughput by optimizing the device scheduling, the time allocation, and the computation resource allocation was considered. Hence, MEC resource allocation could be considered to be time allocation, and channel allocation for minimizing the overall delay of all tasks was investigated. In the context of cooperative task offloading resources, the author of [36] proposed the UAV-enabled MEC system to serve as MEC service for handling multiple UEs. The computation and communication resource responsibility to minimize the weighted sum energy consumption of UEs and UAV was investigated. The proposed approaches aim to provide optimization of UAV trajectory as well as the computation and communication resources based on the Successive Convex Approximation (SCA) method optimized via the Largangian Dual methods. In [37], the authors proposed a UAV assisted with a Fog-IoT network framework that ensures network delay-sensitive tasks and energy consumption for disaster management. Therefore, they proposed to optimally adjust the 3D placement of the UAV-FNs in terms of delay constraint and energy saving to prolong the network lifetime through witlessly transferring power. In [38], the authors proposed an ensemble of semi-parametric models based on the kernel-based Probabilistic Neural Network (PNN) for resource optimization of IoT fog networks. The paper aims to provide the fog computing benefits in facilitating real-time IoT application [39] with the goal to optimize response times in mission-critical tasks.

3. Proposed System Model

3.1. System Architecture

Task offloading comes with a communication cost that needs to be considered for efficient processing. The key challenge is to minimize energy and latency consumption during the communication and computation process. Precise energy, latency, and cost estimation models are important for making informed decisions about task offloading, considering the energy cost of communication according to the time constraints of IoT applications. Similarly, accurate latency estimation models help in determining the appropriate offloading strategy, considering energy, latency, and costs. The goal is to transfer tasks to the edge server if it consumes less energy and the total latency is less than the time constraints of IoT applications. As long as the total latency exceeds the time constraints, our algorithms will choose to execute tasks locally if they can process tasks within the time constraints without considering user device energy consumption. To achieve this, an energy model, latency, and cost model are developed to enable user devices to calculate the energy, latency, and computation cost consumption accurately during task offloading. This approach aims to design an energy-aware and latency-aware offloading strategy. Our proposed network system scenario in Figure 1 represents multi-user devices and multi-edge servers for assisting user devices from overheating task computing. Computational systems have finite resources such as computing power, memory, and bandwidth that need to be shared among multiple tasks or devices. This means that efficient resource allocation and management are crucial to meeting the requirements of the system. In the context of our work, we mentioned that the IoT application is a mission-critical delay-sensitive signaling application. This type of application involves signaling for critical tasks like Mission-Critical Push-to-Talk signaling (MC-PTT) or MC video signaling, where time is a critical factor, and delays need to be minimized. The proposed approach integrates MEC into the 5G network, which can enhance the performance of the network by optimizing resources for each task.

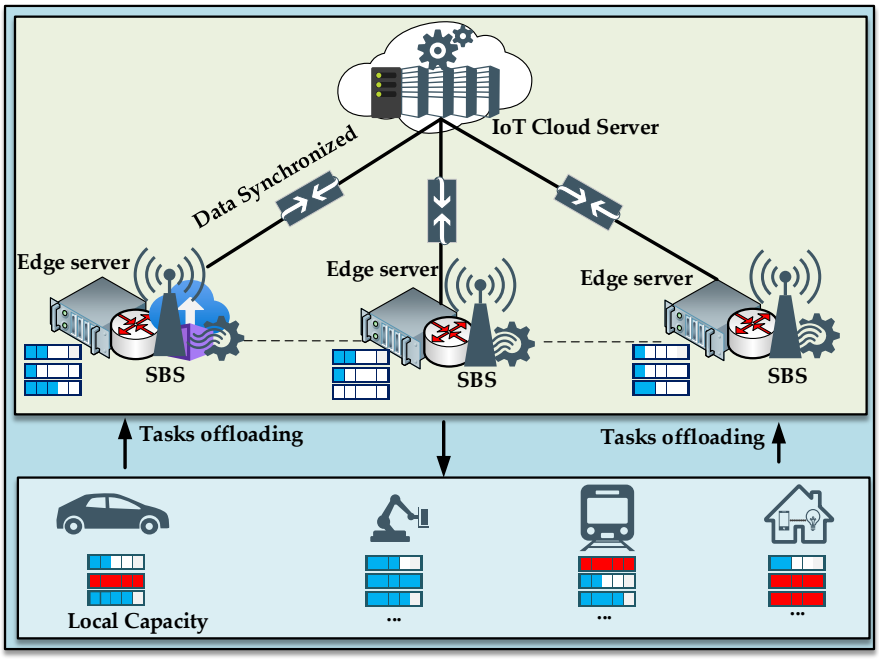

Figure 1. Network system scenario.

3.2. Communication Model

We consider a radio network system to be an edge server consisting of a set $N = \{1, 2, \ldots n\}$, where N refers to the number of user devices. Next, we set $J = \{1, 2, \ldots j\}$, where J refers to Radio Remote Heads (RRHs), and $M = \{1, 2, \ldots m\}$ refers to the number of MEC servers

used in our network scenarios. In our network scenario, all the RRHs are interconnected through a fast gigabit switch to a common processing center comprising a set of Baseband Units (BBUs) denoted as $K = \{1, 2, \ldots k\}$. We assume that each BBU can serve one or more RRHs, and the RRHs can cooperate with each other for downlink transmissions and uplink transmissions to the user device. The channel gain between UE n and RRH j is denoted as g_{nj}. The system bandwidth allocated from RRH j to UE n is denoted as w_{nj}. The fixed transmission power of user device n for offloading task i to RRH j is denoted as p_n^{tr}. The noise power is denoted as n_0. Next, intercell interference is a type of interference that occurs in cellular networks when multiple cells (i.e., base stations) use the same frequency band to communicate with their respective user devices. The signals transmitted by neighboring cells can interfere with each other and cause degradation of the received signal quality, leading to errors in data transmission. In our work, we denote inter-cell interference as Los_{nj}. The transmission rate of computational task i from user device n to base station j is expressed as follows:

$$R_{nj} = W_{nj} log_2 \left(1 + \frac{p_n^{tr} g_{nj}}{n_0 + Los_{nj}}\right) \tag{1}$$

$$t^{loc} = \frac{C_i}{V_i^{loc}} \tag{2}$$

where t^{loc} is the local completion time for user devices n for accomplishing task i in milliseconds (ms). V_i^{loc} is the computing capacity of user devices n in Megahertz (MHz). The computing capacity of an IoT device can vary greatly depending on the specific device and its purpose. IoT devices can range from simple sensors that collect data and transmit it to a central server for processing to more complex devices that have on-board processing capacity. In our work, we use simple sensors on IoT devices which have small computing capacity for implementations. C_i is the total number of computational capacities required to complete computational task i measured in task CPU cycles. In addition, we can obtain the local energy consumption for user devices to accomplish task i as follows:

$$e^{loc} = \alpha C_i \left(V_i^{loc}\right)^2 \tag{3}$$

where α denotes a coefficient associated with the chip architecture [40]. Next, e^{loc} is the local energy consumption for user devices n to accomplish task i. In the context of computing and user devices, energy consumption is typically measured in units of watt seconds (Ws). One watt second is equal to the energy consumed by a device that consumes one watt of power for one second. To convert energy consumption from watt seconds to Joules (j), the formula is as follows: energy (j) = power (W) $\times$ time (s). Therefore, energy consumption is a standard unit of energy measurement.

Also, the computation cost for offloading task i to the edge server refers to the process of initializing latency and energy consumed for offloading task i from user devices to the edge server. The cost formula can be denoted as

$$cost^{loc} = t^{loc} + e^{loc} \tag{4}$$

3.3. Task Offloading Model

Task offloading on the edge servers refers to the process of transferring computational tasks from the user device to more powerful servers located at the edge of the network. The transmission time of task i from user device is given in (5), where D_i is the data size of computation task i to be distributed to the edge server for computation and t^{off} is the offloading time of task i from mobile device to the edge server.

$$t^{off} = \frac{D_i}{R_{nj}} \tag{5}$$

Energy offloading to the edge server means the energy for offloading computing tasks from a user device to an edge server, where e^{off} is the energy for offloading task from task-i to the edge server and p_n^{tr} is the fixed transmission power of mobile device n for offloading task i to the edge server. Computation cost for offloading task i to the edge server can be expressed as (6), where $cost^{off}$ is the cost of energy and latency for offloading task i.

3.4. Computation Offloading Model at the Edge Server

The computation time consumption of task i on the edge server can be expressed as (7), where t^{edge} is the computation time consumption of task i in the edge server, C_i is the required resource for accomplished task i, and V_i^r is the computation resource provided by the MEC server to accomplish task i. V_i^r is measured in Gigahertz (GHz). According to [41], it is generally thought that the download data rate is relatively higher and the size of the computational result is typically smaller than input data, so the downlink delay and energy utilization of this stage are ignored.

$$cost^{off} = t^{off} + e^{off} \tag{6}$$

$$t^{edge} = \frac{C_i}{V_i^r} \tag{7}$$

Energy computation consumption at the edge server can be expressed as (8), where e^{edge} denotes the energy amount for compute task i at the edge server, $P_j^r = z^r \left(V_i^r\right)^{k^r}$ is the computation power consumption [42,43] with effective switched capacitance $z^r \geq 0$ and positive constant $k^r \geq 1$).

$$e^{edge} = \frac{P_j^r C_i}{V_i^r} \tag{8}$$

The computation cost for computing task i at the edge server can be denoted as (9) and the total computation cost for offloading task i at edge server as (10), where $cost^{off\_total}$ is the total cost for offloading the task from the user device to the edge server without considering energy consumption at the edge server due to the fact that the edge server always connects to grid power.

$$cost^{edge} = t^{edge} + e^{edge} \tag{9}$$

$$cost^{off\_total} = cost^{off} + t^{edge} \tag{10}$$

3.5. Offloading Problem Formulation and Analysis

Our proposed IoT system scenario is mission-critical delay-sensitive signaling for which the time constraint is 60 ms based on 3rd Generation Partnership Project (3GPP) QCI Release 17 3GPP TS 23.203 V17.1.0 (2021-06) list [44].

We let $x = \{x_i | i \in I\}$ denote the offloading-decision matrix. To ensure that the task is only offloaded to the MEC server or locally executed, the following constraints must be satisfied: $x_i = 0$ or $x_i = 1$, where if $x_i = 0$, the task will be locally executed, and if $x_i = 1$, the task will be executed at the MEC server. Additionally, we denote $N_{off} = \{n \, \epsilon \, N, m \, \epsilon \, M | x_i = 1\}$ as the set of user devices that offload their tasks to the MEC server. Next, we denote the computational-resource matrix for the MEC as the set of user devices that offload their tasks to the MEC server. Finally, we denote the computational-resource matrix for the MEC as $V = \{V_m^r | m \, \epsilon \, M, n \, \epsilon \, N\}$, where $V_m^r > 0$ is the resource allocation of the edge server to computational task i offloaded from user device $m \, \epsilon \, M$. Normally, the amount of computing resources at the edge server is limited; therefore, the following constraint must be satisfied: $\sum_{n \epsilon N_{off}} V_m^r \leq V_m^{max}, \forall n \epsilon N$, where V_m^{max} denotes the maximum computing capability of edge servers. Moreover, $V_m^r = 0$ if n does not belong to N_{off}, which means that the edge server does not adjust any resources to the mobile node.

3.6. Offloading Decision Sub Problem

In our analysis, we specifically focus on cost offloading without accounting for the energy consumed at the edge, in contrast to the cost incurred by local execution. This comparison is guided by the conditions outlined in Equation (11). Subsequently, we formulate the new offloading decision problem, as articulated in Equation (12). Equations (13) and (14) outline the essential conditions that must be satisfied for a computation task to undergo offloading. These conditions serve as pivotal criteria guiding the decision-making process regarding whether a task should be offloaded to the edge or executed locally. This condition stipulates that the latency incurred by offloading the task to the edge must not exceed a predefined threshold. By imposing such a constraint, we ensure that offloading tasks to the edge remains viable only if it results in latency improvements compared to local execution. This consideration is particularly crucial for latency-sensitive applications where responsiveness is paramount. Furthermore, it introduces another critical condition related to computational resource availability at the edge. This condition dictates that the edge server must possess adequate computational resources to execute the offloaded task effectively. By assessing the computational capabilities of the edge server against the computational requirements of the task, we ascertain whether offloading is feasible from a resource perspective. This condition plays a pivotal role in preventing resource overloading at the edge and ensuring the efficient utilization of available computational resources. Together, they establish a robust framework for evaluating the suitability of offloading computation tasks to the edge. By considering factors such as network latency and computational resource availability, we aim to make informed offloading decisions that optimize both performance and resource utilization in edge computing environments.

$$min \sum_{i \,\epsilon\, I} \left((1 - x_i) \left(\frac{C_i}{V_i^{loc}} + \alpha C_i \left(V_i^{loc} \right)^2 \right) + x_i \left(\left(\frac{D_i}{R_{nj}} \right) + \left(t^{off} * p_i^{tr} \right) \right) \right) \tag{11}$$

$$min \sum_{i \,\epsilon\, I} \left(\left(cost^{off} - cost^{loc} \right) x_i \right) + min \sum_{i \,\epsilon\, I} cost^{loc} \tag{12}$$

$$cost^{off\_total} < cost^{loc} \tag{13}$$

$$t^{off} + t^{edge} < t^{max} \tag{14}$$

In order for tasks to be offloaded effectively, several critical conditions must be met. First, as outlined in the equations, the cost associated with offloading computation to the edge server must be lower than the cost of local computing. This consideration is fundamental in ensuring cost efficiency and justifying the decision to leverage edge resources for computation. By conducting a cost–benefit analysis, we can determine the optimal allocation strategy that minimizes expenses while maximizing performance.

3.7. Lagrange Duality Resource Optimization Algorithm (LDROA)

The LDROA transforms optimization problems with constraints into unconstrained ones by introducing Lagrange multipliers which enforce the constraints and adjust the objective function to find the best solutions. In the context of MEC, this algorithm dynamically distributes resources for task execution while ensuring that QoS standards, resource availability, and limitations are met. It guarantees efficient task execution even in the face of system changes by adapting resource allocation accordingly. In our research, the LDROA plays a pivotal role in dynamically allocating resources for offloaded tasks, responding to fluctuations in available resources or changes in QoS requirements to optimize resource utilization and task execution. This involves formulating an objective function aimed at minimizing task execution time, integrating constraints, and introducing Lagrange multipliers to fine-tune resource allocation. Through iterative solving of primal and dual problems, the algorithm facilitates the identification of the most optimal resource allocation while adhering to the specified constraints. The solution for resource adjustment of our work

via the Lagrange Duality algorithm is presented in (15) following the constraints from (16) and (17). We can form the Lagrangian function by introducing a Lagrange multiplier and modifying the objective function with it as expressed in (18).

$$f(x) = min \sum_{i \in I_{off}} \left(\frac{C_i}{V_i^r} \right) \tag{15}$$

$$s.t.\ V_i^r > 0\ and \sum_{n \in N} (V_i^r) \leq V_m^r \tag{16}$$

$$g(x) = V_i^r - V_m^r \tag{17}$$

$$L(x, \lambda) = f(x) + \lambda(g(x) \tag{18}$$

where L is the Lagrangian optimization problem symbol, $f(x)$ is the objective function to be minimized, $g(x)$ is the constraint function (in this case, $(x) = V_i^r - V_m^r$), and λ is the Lagrange multiplier; therefore, after replacing it into the Lagrangian, we obtain (19) and (20). By taking the first derivative of $L(x, \lambda)$ with respect to V_i^r, we obtain

$$f(x) = \ min \sum_{i\ \epsilon\ I_{off}} \left(\frac{C_i}{V_i^r} \right) \tag{19}$$

$$min\ L(x,\ \lambda) = \sum_{i\ \epsilon\ I_{off}} \left(\frac{C_i}{V_i^r} \right) + \ \lambda \left(\sum_{i \in I_{off}} \ V_i^r - \ V_m^r \right) \tag{20}$$

where V_i^r is the minimum resource for task i, $\sum i \epsilon I_{off}$ are the total numbers of offloading tasks, and V_m^r is the total available resource of the edge server.

In our work, we propose a solution for dynamically adjusting resources using the Lagrange Duality algorithm, as specified in Equation (15), which is formulated based on the constraints provided in Equations (16) and (17). These constraints play a crucial role in defining the boundaries within which resource adjustments can be made effectively. To formalize this solution, we introduce the Lagrange multiplier into the Lagrangian function, as depicted in Equation (18). The Lagrangian function serves as a comprehensive framework for incorporating both the objective function and the constraints into a unified optimization problem. By integrating the Lagrange multiplier into the objective function, we effectively modify the optimization process to account for the constraints imposed by the system. The Lagrange multiplier acts as a scaling factor that influences the trade-off between optimizing the objective function and satisfying the constraints. The introduction enables us to transform the constrained optimization problem into an unconstrained one, facilitating the application of optimization techniques to find the optimal solution. We can introduce Lagrange multipliers, which are additional variables used to enforce the constraints. The Lagrange multipliers are used to adjust the objective function and find the optimal resource allocation for the task. The Lagrange Duality algorithm involves iteratively solving two optimization problems: the primal problem and the dual problem. The primal problem involves minimizing the objective function subject to constraints, while the dual problem involves maximizing the Lagrangian function, which is a modified version of the objective function that incorporates the Lagrange multipliers. By solving these two problems iteratively, we can find the optimal resource allocation that minimizes the execution time of the task subject to the constraints. The Lagrange Duality algorithm allows for dynamic adjustment of the resource allocation in response to changes in the system and ensures that the task is executed efficiently and effectively while meeting the QoS requirements. The derived Functions (21)–(24) are just the process of mathematical formulas to calculate the minimum resource for task i assigned by the edge server. We derivate Equation (21) first with respect to λ so that we can obtain the maximum value of λ. The value of λ, which is the Lagrange multiplier, is ranked between zero and one for the purpose of optimization. So, based on Equation (22), if we can find the maximum

value of λ, then we can find the minimum value of required resource (V_i^r) for task i, which is revealed in the mathematical formula in Equation (24).

$$\frac{\partial yL(v,\lambda)}{\partial V_i^r} = \lambda - \frac{C_i}{\left(V_i^r\right)^2} \tag{21}$$

$$\left(V_r^r\right)^2\lambda = C_i \rightarrow V_i^r = \sqrt{\frac{C_i}{\lambda}} \tag{22}$$

$$\min L(x,\lambda) = \sum_{i\in I_{off}} \frac{C_i}{V_i^r} + \lambda\left(\sum_{i\in I_{off}} V_i^r - V_m^r\right) \tag{23}$$

$$V_i^r = \sqrt{\frac{C_i}{\lambda}} \rightarrow V_i^r = \frac{V_i^r}{\sum i \in I_{off}} \tag{24}$$

3.8. Proposed Solution and Algorithm Design

The algorithms estimate the total latency and cost from the local model, and also the total latency and cost from the offloading model. In the offloading model, first, the process starts by initializing all the parameters needed for resource allocation, such as the available computational resources, the tasks that need to be performed, and the constraints on the allocation [45]. Second, is the resource allocation problem. This step is implemented to solve the resource allocation problem. It involves assigning tasks to the available computational resources at the edge server in a way that optimizes total latency and meets with the requirement of IoT time constraints. Third is the offloading decisions. Once the resource allocation problem is solved, the next step is to extract the offloading decisions from the solution, which is the local model or the offloading model. Finally, after the decision is selected, the optimal solution can be obtained in terms of energy and latency saving. Figure 2 shows more details of the flowchart of the algorithm in the proposed approach of LDROA. First, the process starts by initializing all the parameters needed for resource allocation, such as the available computational resources, the tasks that need to be performed, and the constraints on the allocation. The next step is to solve the resource allocation problem, which involves assigning tasks to the available computational resources in a way that optimizes some objective function, such as minimizing the total time or energy required to perform the tasks. Once the resource allocation problem is solved, the next step is to extract the offloading decisions from the solution. This involves determining which tasks should be offloaded to edge and which tasks should be performed locally on the device. In some cases, the resource allocation problem may be too complex to solve directly. In these cases, Lagrange Duality can be used to simplify the problem and find an approximate solution. This involves introducing Lagrange multipliers to convert the constraints of the problem into the objective function, which can then be optimized using standard methods. After the resource allocation problem is solved, the computational resources may need to be adjusted to ensure that the tasks are performed optimally. This involves iteratively adjusting the resources and evaluating their performance until the optimal resource allocation is reached. Once the optimal resource allocation is determined, the tasks can be mapped to the allocated resources, and the computation can be performed. If the adjusted resources do not converge to the minimum, the process will need to be repeated until the optimal solution is reached.

Algorithm 1 shows the flow of our proposed approach. In the described context, the algorithm focuses on predicting the total cost of local task computation and offloading it to the edge server, taking into account the time constraints of the IoT application.

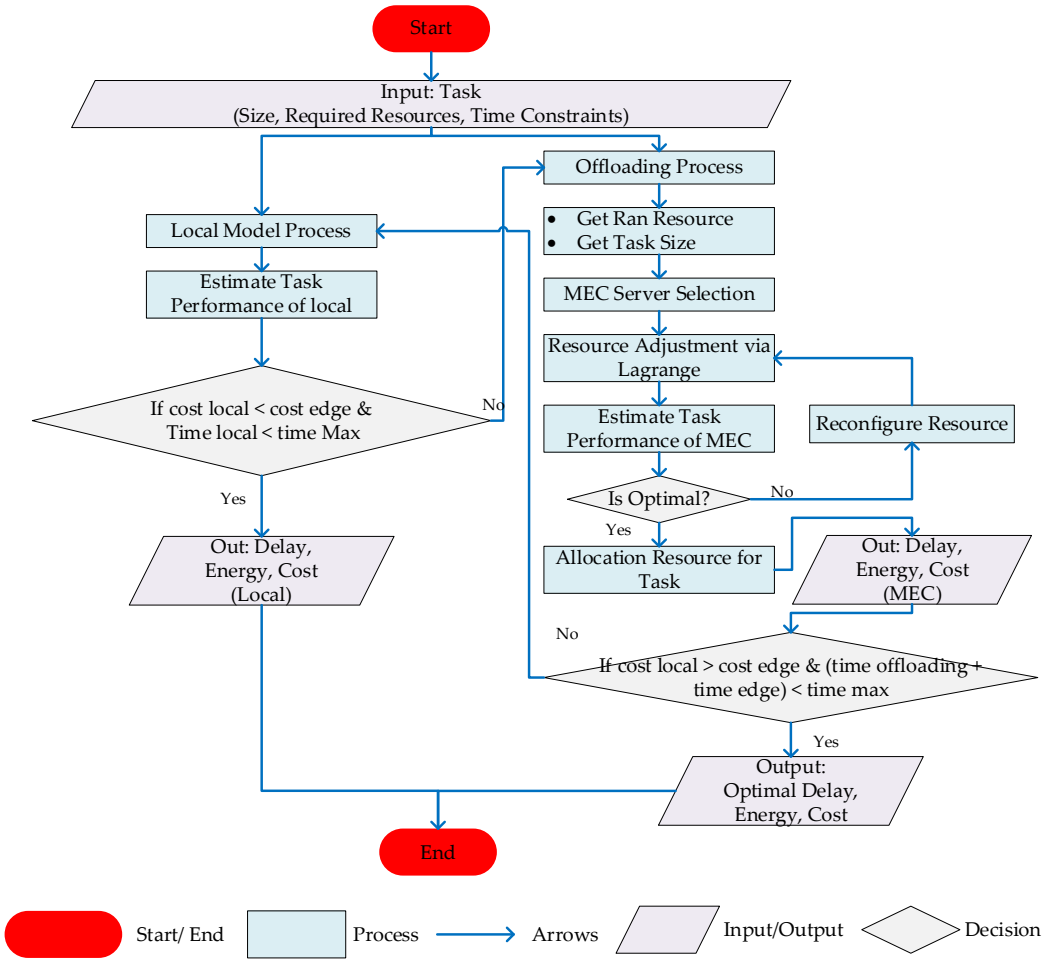

Figure 2. Flowchart of the algorithm.

The aim of the algorithm is the prediction of the total cost, which encompasses various factors such as computation expenses, data transmission costs, and resource utilization fees. By accurately estimating these costs, the algorithm can make informed decisions regarding task offloading, ensuring that the chosen strategy aligns with the overarching objectives of minimizing expenses while meeting performance requirements. Importantly, the algorithm considers the unique time constraints inherent in IoT applications, where timely data processing and response are paramount. By incorporating these constraints into the decision-making process, the algorithm ensures that offloaded tasks are executed within the stipulated time limits, thereby guaranteeing the timely delivery of services and maintaining the integrity of the application. The algorithm accounts for the dynamic nature of edge computing environments, where resource availability and network conditions may vary over time. By adapting to these changes and adjusting resource allocation accordingly, the algorithm optimizes the overall cost effectiveness of task offloading while maintaining reliability and performance. The requirements for the algorithm include the resources available, task size, and time constraints. To calculate the total cost, the algorithm considers both the energy and time required for task computation. For local task computation, the amount of energy and time needed to compute the task depends on the available resources of the user devices. If the devices have higher computational capacity or more resources, the local task computation will be faster. If the cost of computation is lower when offloading to the MEC system compared to performing the task locally on the user device and the total execution time at the MEC system is within the time constraints, the user device will offload the task to the MEC system. However, if the total latency at the edge exceeds the time constraints but the total execution time locally on the user device is shorter than the

time constraints, our algorithm will prioritize executing the task locally. According to Algorithm 1, I is the total number of computational tasks and i defines the iterations. The outputs are x, l, e, and c where x is the offloading decision, l is the total latency, e is the total energy, and , c is the computation cost. In the simulation, we set the algorithms to iterate 60 times, and the algorithms can converge while the number of 40 iterations is reached. The algorithm detail explanation is already included in the proposed flowchart.

Algorithm 1: Local and edge server computation time, energy, and cost.

Required $task_i = \{C_i, D_i, T^{max}, \forall i \in I\}, \{R_k, C_i, \forall k \in K \; and \; \forall i \in I\}$
Initialize $if \; x_i = 0, \quad \forall i \in I$
$I_{loc} = I$
Output: $\{t^{loc}, e^{loc}, cost^{loc}\}$
$if \; x_i = 1, \forall i \in I$
$I_{off} = I$
V_i^r adjustment for $task_i$ via **LDROA**
    While $V_i^r \; not \; converge$ **do**
        for $(iteration = 1; \; iteration \leq 60; \; iteration++)$
            if $cost^{edge} > cost^{loc}, \quad \forall I \in I_{off}$
            i $\leftarrow \arg\min_{I \in I_{off}}(x_i(cost^{loc} - cost^{edge}))$
            $I_{off} \leftarrow I_{off} - \{i\}$
            Update x by setting $x_i \leftarrow 0$
            Update V_m^r (available resource of MEC) according to resource allocation
            else if $cost^{edge} < cost^{loc}, \; \forall I \in I_{off}$, or $I_{off} = 0$ (empty)
            End if
        End
        While $(t^{off} + t^{edge} > T^{max})$ **do**
            i $\leftarrow \arg\min_{i \in I_{off}}(x_i(cost^{loc} - cost^{edge}))$
            $I_{off} \leftarrow I_{off} - \{i\}$
            Update x by setting $x_i \leftarrow 0$
            Update V_m^r (available resource of MEC) according to resource allocation V_i^r
        End
        $cost^{edge} < cost^{loc}, \; \forall i \in I_{off} \; or \; I_{off} = 0$
    End
    Output optimal solution $\{x, l, e, c\}$

The algorithmic flow of a proposed approach focuses on predicting the total cost of local task computation and offloading it to an edge server within the constraints of an IoT application. The algorithm considers various factors, including available resources, task size, and time constraints. It calculates the total cost by considering both the energy and time required for task computation. The algorithm provides a systematic way to decide whether to perform computation locally or offload it to an edge server based on resource availability, task size, and time constraints, optimizing for energy and time efficiency to achieve cost effectiveness in IoT applications. In summary, Algorithm 1 encapsulates a systematic approach to task offloading in IoT applications, emphasizing the prediction of total costs and the fulfillment of time constraints. Through its comprehensive framework, the algorithm facilitates efficient resource management and decision making, ultimately enhancing the scalability, reliability, and cost efficiency of edge computing systems in IoT environments.

4. Experimentation and Result Evaluations

4.1. Simulation Setting and Parameters

For our simulation, we modeled a heterogeneous network that is represented by 20 user devices and composed of five small base stations and one main base station, which represents multi-user and multi-MEC, and all small base stations are connected to the main base station. The power gain (g_{nj}) is set to 100 milliwatts (mw). Next, the noise power (n_0), which refers to the amount of random electrical energy that interferes with the transmitted signal during wireless communication in a RAN, is set to -100 dBL. Additionally, according to [41], we consider that there is no inner-cell interference in wireless fronthaul. The system bandwidth (W_{nj}) is set to 10 Megahertz (MHz). The size of the input tasks (D_i) is randomly distributed within the range of 1.5 KB to 2 KB. The computing resources of the MEC server (C_i) are set to 1 GHz, 1.5 GHz, 2 GHz, 2.5 GHz, and 3 GHz, respectively. The available computing resources of the user devices (V_i^{loc}) are randomly assigned from 10 to 100 MHz [14]. The number of the required CPU cycles per bit is C_i = 20 cycles. The time constraint (T^{max}) of mission-critical delay-sensitive signaling (e.g., MC-PTT signaling, MC Video signaling) IoT applications is set to less than 60 ms based on 3GPP TS Release V17.1.0 (2021-06). The set of transmission powers (p_n^{tr}) for each UE is set to 200 mw. This means that the UE is capable of transmitting signals at power levels within this range, which can be used for wireless communication with other devices or networks. We set the number of iterations (i) to 60, and our algorithm is able to obtain the optimal result after it reaches 40 iterations. The performance metric that is used is total latency in milliseconds (ms), total cost that includes both energy and latency.

4.2. Proposed and Reference Schemes

To comprehensively assess the efficacy of our proposed approach, we undertake a rigorous comparative analysis against a set of reference schemes in our study. This comparative evaluation provides valuable insights into the relative performance and advantages of our proposed idea in contrast to existing methods commonly employed in edge computing environments. The reference schemes considered in our evaluation encompass a diverse range of offloading strategies, each with its unique characteristics and optimization goals. First, the Random Offloading scheme serves as a baseline approach, where tasks are offloaded to edge servers randomly without considering any optimization criteria. This simplistic strategy provides a benchmark for assessing the performance improvements achieved by more sophisticated offloading techniques. Second, the Load Balancing scheme aims to evenly distribute computational tasks among available edge servers to minimize resource contention and maximize resource utilization. This approach prioritizes load balancing across the network, thereby enhancing system scalability and responsiveness. Third, the Greedy Latency Offloading scheme focuses on minimizing latency by offloading tasks to the nearest edge server with the lowest latency. This strategy prioritizes latency optimization, ensuring rapid response times and improved user experience, particularly in latency-sensitive applications. By comparing our proposed idea against these reference schemes, we can evaluate its performance across various dimensions, including latency reduction, cost efficiency, resource utilization, and scalability. Through extensive experimentation and analysis, we aim to demonstrate the superiority of our proposed approach in optimizing task offloading decisions and enhancing the overall effectiveness of edge computing systems. Furthermore, the comparative evaluation provides valuable insights into the strengths and limitations of different offloading strategies, thereby informing future research directions and advancements in the field of edge computing. By systematically evaluating and benchmarking our proposed idea against established reference schemes, we contribute to the ongoing discourse on optimizing resource allocation and improving the performance of edge computing environments. Based on [33], the Random Resource Offloading scheme tasks are randomly offloaded to available resources such as edge servers without considering factors such as resource availability, network conditions, or energy consumption. Indeed, the utilization of simplistic or Random Offloading approaches can lead

to several undesirable outcomes within edge computing environments. One notable consequence is inefficient resource utilization, where computational tasks may not be allocated optimally across available resources. As a result, certain edge servers may be underutilized while others are overwhelmed, leading to uneven resource distribution and suboptimal performance. This approach may result in inefficient resource utilization, longer latencies, and greater energy consumption. Overall, the use of Random Offloading strategies poses significant challenges in terms of resource management, latency optimization, and energy efficiency within edge computing environments.

The Load Balancing resource offloading scheme is a technique used to optimize the usage of resources on edge servers. The goal of this scheme is to distribute the workload evenly across multiple servers or nodes in the system to prevent overloading and ensure equal resource allocation for each task. A load balancer evaluates the available resources on the system equally and decides which node or server is best suited to handle the task. Implementing Load Balancing mechanisms adds complexity to the system architecture, introducing overhead in terms of computational and management resources. Additionally, reliance on centralized controllers or algorithms can create a single point of failure, undermining system reliability and fault tolerance. Moreover, Load Balancing may introduce network overhead, impacting latency and consuming bandwidth, particularly in distributed environments. Despite the goal of evenly distributing the workload, variations in task characteristics and resource capabilities may result in unequal resource utilization, affecting system performance. Furthermore, traditional Load Balancing approaches may struggle to adapt to dynamic changes in workload and resource availability, leading to suboptimal task allocation.

In the Greedy Latency offloading scheme, the offloading decision is made based on the estimated latency of executing the task locally on the user device or offloading the task to the edge server. The offloading decision is greedy in the sense that it always chooses the option with the lowest estimated latency without concerning too much about energy or cost consumption. One weakness of the Greedy Latency offloading scheme is that it does not consider the available resources on the edge server or the network conditions. The offloading decision is solely based on minimizing latency, and it may result in overloading some edge servers or congested networks. To summarize, the Greedy Latency offloading scheme, while effective in minimizing latency, may also present several drawbacks. Primarily, its locally optimal choices at each step without considering the available resource at edge server can lead to suboptimal resource utilization and cost efficiency. Tasks may be offloaded based solely on immediate latency, potentially resulting in resource imbalances among edge servers and inefficient utilization. Additionally, the scheme's reliance on network latency as the primary criterion for task allocation may make it sensitive to variable network conditions, risking network congestion and degraded performance. Moreover, Greedy Latency schemes may overlook energy consumption considerations, potentially leading to increased energy usage and operational costs. Overall, while effective in improving responsiveness, the Greedy Latency offloading scheme must be employed judiciously considering its potential drawbacks in resource management, network congestion, and computation cost.

Finally, the proposed LDROA algorithm can be used to consider on/offloading decisions and find the minimum execution time for a given task at edge node, subject to various constraints such as resource availability resource limits and the requirements of the IoT time constraint. By using Lagrange, the proposed approach can provide an effective way of adjusting the available resources for each task. This can result in a more optimal resource allocation scheme, leading to reduced energy consumption, lower latency costs, and decreased computation costs. For example, if a task requires a large amount of computing power but has a low-latency requirement, the Lagrange method can adjust the resource allocation to favor computing power over latency. The joint offloading decisions and resource allocation by Lagrange, which is called the LDROA, can provide a more effective way of minimizing both computation time and computation cost in the model.

The main purpose of our algorithms is to save energy on user devices and optimize resource allocation for offloading tasks to edge servers. This dual objective aims to enhance both the battery life of user devices and the overall latency for tasks executed at edge servers. That is why we consider the computation cost for comparison in our algorithm. First, LDROA outperforms Load Balancing because the goal of Load Balancing is to distribute the workload evenly across multiple servers or nodes in the system to prevent congestion and ensure equal resource allocation for each task. In this scheme, the workload is analyzed to determine the number of resources required to complete the task. The load balancer then evaluates the resources available in the system and decides which node or server is best suited to handle the task. For example, in our work, some tasks need to be offloaded to the same edge server, which provides the resources for the tasks to be executed equally by not considering local execution on the user device at all. Therefore, it is not as effective as our proposed approach. Load Balancing can save energy on user devices, but it can take much time to execute the task, leading to higher computation costs as well as computation latency. So, our proposed algorithm can make intelligent offloading decisions and optimize resources for tasks on edge servers, thereby reducing the total cost and total latency. This approach ensures that the total available MEC server resources meet the task requirements and time constraints while saving energy on user devices.

Second, the proposed LDROA can be superior to the method of Random Offloading, because in the random resource offloading scheme that is mentioned in references [32,33], tasks are assigned to available resources such as edge servers without considering factors such as resource availability, network conditions, or power consumption. This approach can result in inefficient resource utilization, longer latencies, and higher power consumption because tasks may be assigned to resources that are not optimized for their execution. This approach can result in the complex task being offloaded to the MEC server with congestion or with a very little number of available resources for each task. This leads to high congestion and increased execution time as well as computation costs. However, our algorithm can dynamically adjust with the minimum required resource for each task so that the remaining resource of the edge server can be well utilized for the next coming task. Our algorithms not only elastically adjust the resource to minimize the cost and latency, but also proactively estimating the total communication and computation cost of the tasks on the edge server and computation cost on the user device and make the optimal decision of whether the task should be uploaded or just computed locally by considering the maximum time constraint of the IoT application utilized in this work. According to this process of algorithm work, that is why we clarify that our algorithms are good.

Third, the proposed LDROA can be superior to the method of Greedy Latency offloading because the Greedy Latency offloading mentioned in [31,32] makes decisions based on the estimated latency of executing the task locally on the user device or offloading the task to the edge server. The offloading decision is greedy in the sense that it always chooses the option with the lowest estimated latency without worrying too much about energy or cost. If the latency of executing the task locally is lower than the latency of offloading the task to the edge server, the task will be executed locally on the user device, even if the power or cost is higher on the edge server. If the estimated latency of offloading the task to the edge server is lower than executing the task locally, the task is offloaded to the edge server for execution without regard for compute cost or energy. A weakness of the Greedy Latency offloading scheme is that it does not take into account the available resources on the edge server or network conditions. The offloading decision is based solely on minimizing latency, and it may result in overloading some edge servers or congesting networks. This means that all tasks appear to be offloaded to the edge server with the highest computing capacity rather than considering its available resources. In contrast, our proposed algorithms can dynamically adapt to each task with the minimum required resource, so that the remaining resource of the edge server can be well utilized for the next coming task.

5. Simulation Result and Discussion

5.1. Simulation Based on User Device Computing Capacity

The computing capacities of the user device within 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100 MHz are taken for the experiment. For the performance metric used in the result, we consider only the total latency in seconds and the total cost for explanation. However, for easy understanding, the result of total latency is converted to millisecond mapping to meet the time constraints of IoT applications. In total cost, the value comes from total latency in seconds and total energy in j. That is why there is a big range between latency and cost value.

In Figure 3 on the left side, we compare the performance of total latency on average in terms of the user device computing capacity. In numerical results, the proposed approach incorporates a blend of local computing and offloading to an edge server, yielding the following total latency values in milliseconds for each level of computing capacity of the user devices: 32.42, 17.22, 12.15, 9.62, 8.10, 7.09, 6.36, 5.82, 5.40, and 5.06. These findings demonstrate that as the computing capacity of the user device increases, the total latency decreases, indicating the effectiveness of the proposed approach in latency reduction. In comparison to the proposed approach, the conventional scheme comprises three methods: Greedy Latency, Load Balancing, and Random Offloading. Greedy Latency has a total latency in ms of 35.14, 25.54, 22.34, 20.74, 19.78, 19.14, 18.68, 18.34, 18.07, and 17.86 according to the computing capacity of the user device of 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100 MHz, respectively. Load Balancing can obtain a total latency of 35.54, 26.74, 23.81, 22.34, 21.46, 20.88, 20.46, 20.14, 19.90, and 19.70 according to the computing capacity of the user device of 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100 MHz, respectively. Finally, Random Offloading is less efficient than other approaches, with a total latency of 36.46, 28.46, 25.79, 24.46, 23.66, 23.13, 22.75, 22.46, 22.24, and 22.06 according to the computing capacity of the user device of 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100 MHz, respectively. If we compare the performance of total latency on average in terms of the user device computing capacity, when the computing capacity of the user device increases from 10 MHz to 100 MHz, the total latency of our proposed approach is 97.46% better than that of Greedy Latency, 111.51% better than that of Load Balancing, and 130.28% better than that of Random Offloading. Overall, the proposed approach outperforms all three conventional schemes in terms of latency reduction, and as the computing capacity of the user device increases, the proposed approach's latency reduction becomes even better.

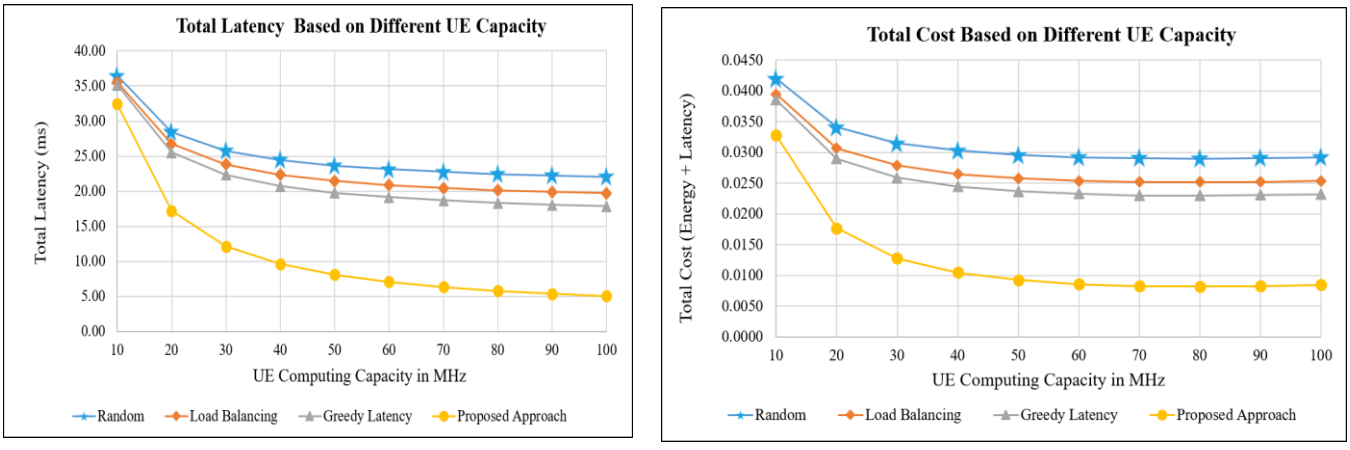

Figure 3. Total latency and cost based on different UE capacities.

In Figure 3 on the right side, we compare the performance of total cost on average in terms of the user device computing capacity. The proposed approach results in the lowest cost compared to other schemes. The Lagrange scheme can compute the total cost of 0.0328, 0.0177, 0.0128, 0.0105, 0.0093, 0.0086, 0.0083, 0.0082, 0.0083, and 0.0085 based on

user device computing capacity ranging from 10, 20, 30, 40, 50, 60, 70, 80, 90, to 100 MHz, respectively. The Lagrange scheme's total cost is based on user device computing capacities ranging from 10 MHz to 100 MHz, with the total cost decreasing from 0.0328 to 0.0085 as the device's computing capacity increases. The Greedy Latency scheme can also provide a low cost following the proposed approach, with total costs of 0.0386, 0.0290, 0.0259, 0.0245, 0.0237, 0.0233, 0.0230, 0.0230, 0.0231, and 0.0232 based on user device computing capacities ranging from 10 MHz to 100 MHz, respectively. The Greedy Latency scheme's total cost also decreases as the device computing capacity increases, ranging from 0.0386 to 0.0232. The Load Balancing scheme results in an average total cost per device of 0.0395, 0.0307, 0.0279, 0.0265, 0.0258, 0.0254, 0.0252, 0.0252, 0.0252, and 0.0254 based on device computing capacity ranking from 10 MHz to 100 MHz, respectively. Finally, Random Scheme can obtain a total cost of 0.0420, 0.0341, 0.0315, 0.0303, 0.0296, 0.0292, 0.0291, 0.0290, 0.0291, and 0.0292 based on device computing capacities ranging from 10 MHz to 100 MHz, respectively. Random Access Scheme's total cost decreases slightly as the device computing capacity increases, ranging from 0.0420 to 0.0292. If we compare the performance of total cost on average in terms of the user device computing capacity, when the computing capacity of the user device increases from 10 MHz to 100 MHz, the total cost of our proposed approach is 105.84% better than that of Greedy Latency, 121.44% better than that of Load Balancing, and 150.48% better than that of Random Offloading. Overall, the proposed approach is shown to be the most energy-efficient approach for offloading and executing tasks on user devices, with lower total costs than the other three schemes.

5.2. Simulation Based on Different Numbers of Tasks

In Figure 4 on the left side, we compare the performance of total latency on average in terms of different number of tasks. The number of tasks being offloaded to an edge server can affect the performance of latency, energy, and computation costs. Offloading a large number of tasks to an edge server can increase the overall latency of the system, as the edge server has to process all the tasks it receives. This can result in increased waiting times for users, which can negatively impact user experience. Offloading a large number of tasks to an edge server can also increase the energy consumption of the system as well as the computation cost of the system. This can result in increased power consumption and reduced battery life, which can be particularly important for user devices. The figure shows the total latency in ms obtained by each approach. The proposed Lagrange approach achieves a total latency of 6.40, 12.80, 24.45, 38.40, 51.20, and 64.00 ms according to the task numbers of 20, 40, 80, 120, 160, and 200, respectively. These results demonstrate that as the number of tasks increases, the proposed algorithm still performs well in reducing latency. In contrast, the conventional approaches show different results: Greedy Latency has total latencies of 35.14, 39.56, 77.96, 118.67, 158.23, and 197.79 ms according to the task numbers of 20, 40, 80, 120, 160, and 200, respectively. Load Balancing can obtain total latencies of 35.54, 42.93, 84.70, 128.77, 171.70, and 214.62 ms according to the task numbers of 20, 40, 80, 120, 160, and 200, respectively. Finally, Random Offloading has a total latency of 23.66, 47.32, 93.48, 141.95, 189.26, and 236.58 ms according to the task numbers of 20, 40, 80, 120, 160, and 200, respectively, which is less efficient than the other approaches. If we compare the performance of total latency on average in terms of different numbers of tasks, when the task number increases from 20 to 200, the total latency of our proposed approach is 218.05% better than that of Greedy Latency, 243.86% better than that of Load Balancing, and 271.23% better than that of Random Offloading. Overall, the results show that the proposed Lagrange approach still outperforms conventional approaches in terms of reducing latency, especially as the number of tasks increases.

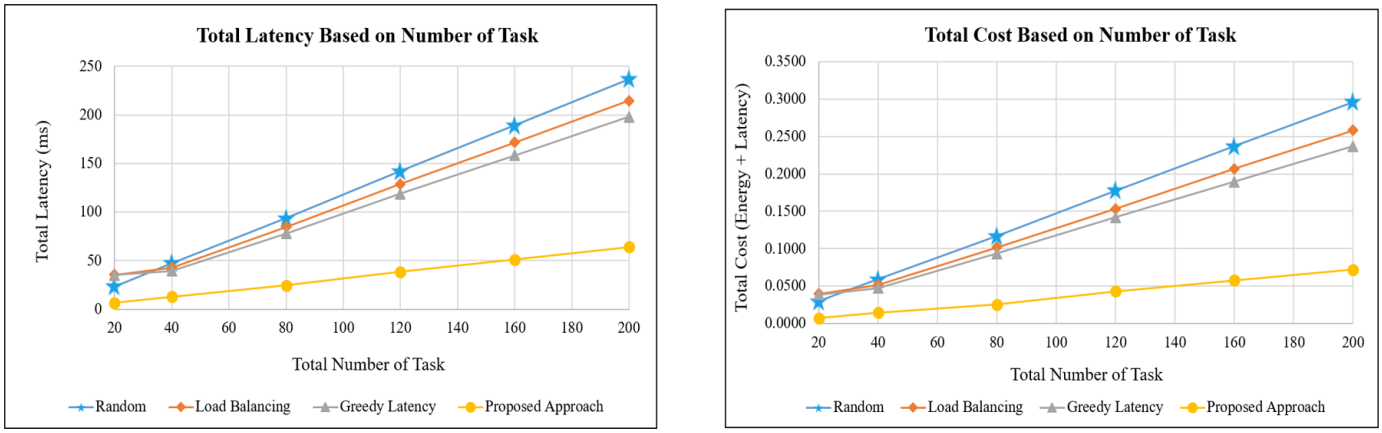

Figure 4. Total latency and cost based on number of tasks.

In Figure 4 on the right side, we compare the performance of total cost on average in terms of different numbers of tasks. The proposed Lagrange approach achieves total costs of 0.0072, 0.0144, 0.0255, 0.0432, 0.0576, and 0.0720 according to the task numbers of 20, 40, 80, 120, 160, and 200, respectively. These results indicate that the proposed algorithm effectively reduces computation costs even as the number of tasks increases. On the other hand, the conventional approaches yield different outcomes: Greedy Latency has total costs of 0.0386, 0.0474, 0.0934, 0.1421, 0.1894, and 0.2368 according to the task numbers of 20, 40, 80, 120, 160, and 200, respectively, while Load Balancing can obtain a total cost of 0.0395, 0.0516, 0.1020, 0.1534, 0.2066, and 0.2582. Lastly, Random Offloading yields a total cost of 0.0296, 0.0592, 0.1171, 0.1776, 0.2368, and 0.2959 as the number of tasks ranges from 20 to 200, respectively. If we compare the performance of total cost on average in terms of different numbers of tasks, when the task number increases from 20 to 200, the total cost of our proposed approach is 240.02% better than that of Greedy Latency, 268.94% better than that of Load Balancing, and 316.64% better than that of Random Offloading. Overall, these findings suggest that the proposed Lagrange approach outperforms conventional methods in terms of reducing costs, particularly as the number of tasks increases.

5.3. Simulation Based on Different Numbers of MEC Servers

The number of MEC servers can have a significant impact on offloading latency costs and energy consumption in the edge node. Generally, adding more MEC servers to the system can reduce the offloading latency cost, as tasks can be distributed across multiple servers and processed in parallel, leading to faster processing times. However, adding more servers can also increase energy consumption, as each server requires power to operate. The optimal number of MEC servers for a given MEC system depends on various factors, including the workload characteristics, the performance requirements, and the energy constraints. For instance, if the workload is highly parallelizable and energy consumption is a key concern, adding more servers can lead to better performance and energy efficiency. On the other hand, if the workload is less parallelizable and energy consumption is not a limiting factor, a smaller number of servers may suffice.

In Figure 5 on the left side, we compare the performance of total latency on average in terms of different numbers of MEC Servers. The given information presents the results of a comparison between the proposed scheme and three conventional offloading approaches: Greedy Latency, Load Balancing, and Random Offloading based on different numbers of MEC servers, including 2, 4, 6, 8, and 10. The proposed scheme achieves total latencies in ms of 62.88, 60.60, 58.49, 56.41, and 54.28 according to the MEC numbers of 2, 4, 6, 8, and 10, respectively. These results indicate that the proposed scheme performs well in reducing latency as the number of MEC servers increases. In comparison to the proposed scheme, the Greedy Latency approach achieves total latencies of 62.87, 61.60, 59.20, 57.89, and 56.67 according to the MEC numbers of 2, 4, 6, 8, and 10, respectively. Similarly, the Load

Balancing approach achieves total latencies of 62.92, 61.58, 60.39, 59.16, and 57.91 according to the MEC numbers of 2, 4, 6, 8, and 10, respectively. The Random Offloading approach obtains total latencies of 63.06, 62.06, 61.00, 59.87, and 58.67 according to the MEC numbers of 2, 4, 6, 8, and 10, respectively. If we compare the performance of total latency on average in terms of different numbers of edge servers, when the number of edge servers increases from 2 to 10, the total latency of our proposed approach is 1.90% better than that of Greedy Latency, 3.18% better than that of Load Balancing, and 4.10% better than that of Random Offloading. Overall, the results show that the proposed scheme and the conventional approaches demonstrate varying levels of performance in reducing latency based on the number of MEC servers. However, the proposed scheme outperforms conventional approaches in reducing latency, particularly as the number of MEC servers increases.

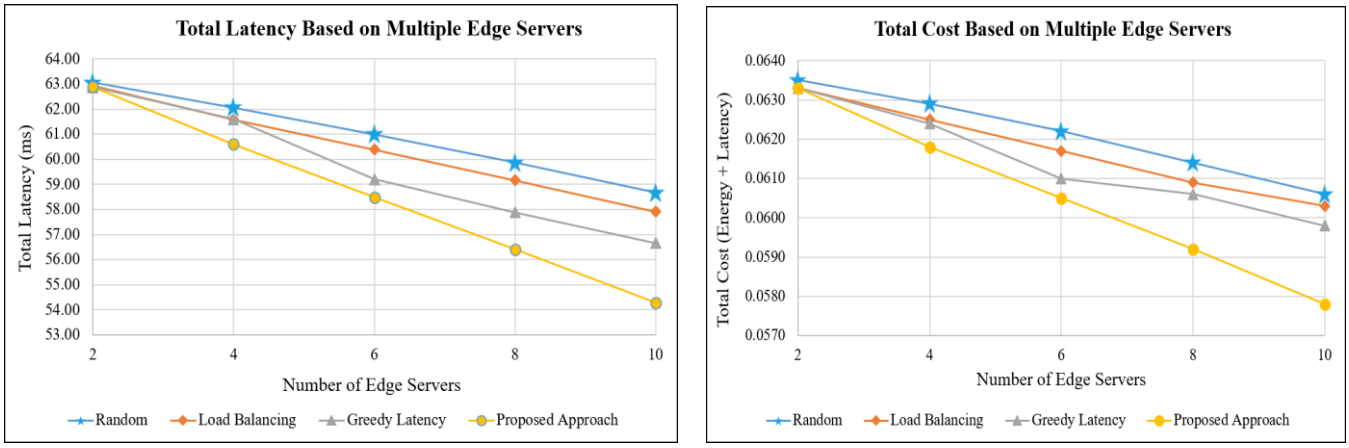

Figure 5. Total latency and cost based on different numbers of MEC servers.

In Figure 5 on the right side, we compare the performance of total cost on average in terms of different MEC servers. As the number of MEC servers varies from 2 to 10, the proposed Lagrange method achieves lower total costs of 0.0633, 0.0618, 0.0605, 0.0592, and 0.0578. These results indicate that the proposed algorithm is effective in reducing computation costs even when the number of tasks increases. In contrast, conventional approaches have different results. Greedy Latency yields total costs of 0.0633, 0.0624, 0.0610, 0.0606, and 0.0598, while Load Balancing results in a total cost of 0.0633, 0.0625, 0.0617, 0.0609, and 0.0603. Lastly, Random Offloading produces a total cost of 0.0635, 0.0629, 0.0622, 0.0614, and 0.0606 according to the MEC numbers of 2, 4, 6, 8, and 10, respectively. If we compare the performance of the total cost on average in terms of different numbers of edge servers, when the number of edge servers increases from 2 to 10, the total cost of our proposed approach is 1.49% better than that of Greedy Latency, 2.02% better than that of Load Balancing, and 2.64% better than that of Random Offloading. These findings indicate that the proposed Lagrange approach is slightly superior to conventional methods in reducing costs, particularly as the number of MEC servers increases.

5.4. Simulation Based on Congestion Condition

Congestion in mobile edge computing occurs when the network or resources in the infrastructure are overloaded with a high volume of data or requests. This overload leads to delays, increased latency, and degraded performance for applications and services. Congestion can result from factors like high user demand, limited network bandwidth, inadequate processing capacity, or improper resource allocation. Effective management of congestion is essential to maintain the efficient and reliable operation of mobile edge computing systems. By implementing strategies like Load Balancing, resource provisioning, and traffic prioritization, congestion can be alleviated, and the overall performance of mobile edge computing can be improved. However, our proposed approach based on Lagrange can perform better in congestion conditions.

In Figure 6 on the left side, we compare the performance of total latency on average in terms of different congestion conditions. The figure shows the total latency in seconds obtained by each approach. The proposed Lagrange approach achieves total latencies between 0.02 and 0.06 s under the no-congestion condition. Next, in low-congestion conditions, it can achieve a total latency between 2.56 and 3.20 s. In moderate congestion, the total latency is 32.00 s, and finally it is 64.00 s in high-congestion conditions. These results demonstrate that even in conditions of high congestion, the proposed algorithm still performs well in reducing latency. In contrast, the conventional approaches show different results: Greedy Latency achieves total latencies between 0.08 and 0.20 s under the no-congestion condition. Next, in low-congestion conditions, it can obtain a total latency between 7.91 and 9.89 s. In moderate congestion, the total latency is 98.89 s, and finally it is 197.79 s in high-congestion conditions. Load Balancing can obtain total latencies between 0.08 and 0.21 s under the no-congestion condition. Next, in low-congestion conditions, it can obtain a total latency between 8.58 and 10.73 s. In moderate congestion, the total latency is 107.31 s, and it is 214.62 s in high-congestion conditions. Finally, Random Offloading has a total latency between 0.09 and 0.24 s for the no-congestion condition. Next, in low-congestion conditions, it can obtain a total latency between 9.46 and 11.83 s. In moderate congestion, the total latency is 118.29 s, and finally it is 236.58 s in the high-congestion condition. If we compare the performance of the total latency on average in terms of all congestion conditions, the total latency of our proposed approach is 209.07% better than that of Greedy Latency, 235.36% better than that of Load Balancing, and 269.69% better than that of Random Offloading scheme.

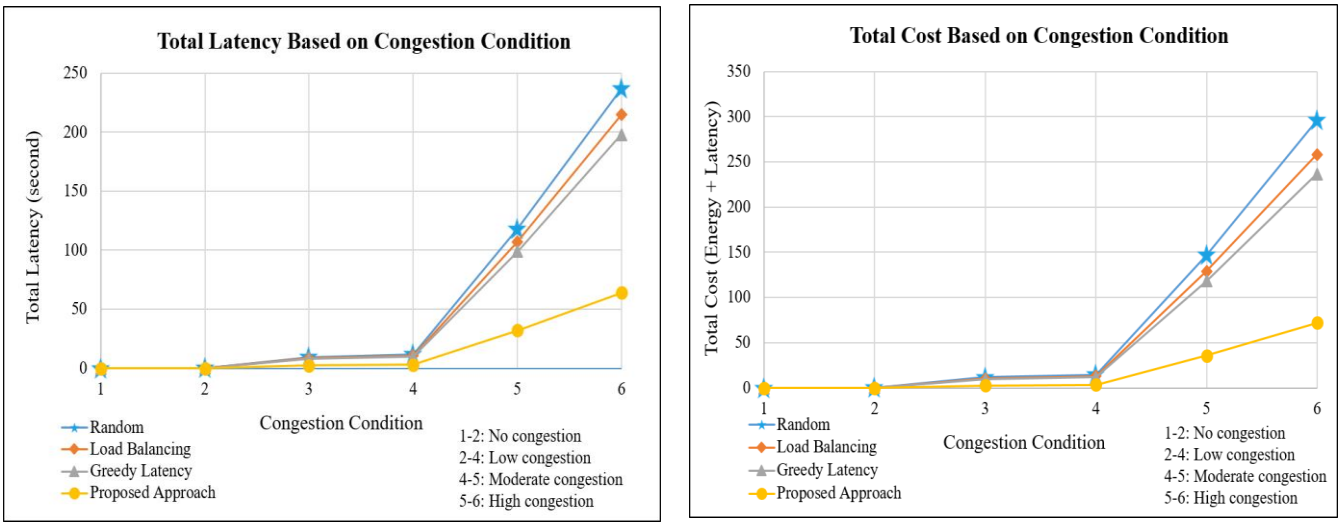

Figure 6. Total latency and cost based on congestion conditions.

In Figure 6 on the right side, we compare the performance of total cost on average in terms of different congestion conditions. This figure presents a comparison of total cost between the proposed Lagrange approach and three conventional task offloading schemes in congestion conditions such as no congestion, low congestion, moderate congestion, and high congestion. The figure shows the total cost obtained by each approach. The proposed Lagrange approach achieves a total cost between 0.0255 and 0.0720 under no-congestion conditions. Next, in low-congestion conditions, it can achieve a total cost between 2.8800 and 3.6000. In moderate congestion, the cost is 36.0000, and finally it is 72.0000 in high-congestion conditions. These results demonstrate that even in conditions of high congestion, the proposed algorithm still performs well in reducing computation costs. In contrast, the conventional approaches show different results: Greedy Latency achieves a total cost between 0.0934 and 0.02368 under the no-congestion condition. Next, in low-congestion conditions, it can obtain a total cost between 9.4721 and 11.8401. In

moderate congestion, the total cost is 118.4000, and finally it is 236.8000 in high-congestion conditions. Load Balancing can obtain total costs between 0.1020 and 0.2582 under the no-congestion condition. Next, in low-congestion conditions, it can obtain a total cost of between 10.3284 and 12.9104. In moderate congestion, the total cost is 129.1050, and it is 258.21 in high-congestion conditions. Finally, Random Offloading has a total cost between 0.1171 and 0.2989 for the no-congestion condition. Next, in low-congestion conditions, it can obtain total cost between 11.8376 and 14.7970. In moderate congestion, the total cost is 147.0000, and finally it is 295.94 in high-congestion conditions. If we compare the performance of the total cost on average in terms of all congestion conditions, the total cost of our proposed approach is 228.90% better than that od Greedy Latency, 258.63% better than that of Load Balancing, and 310.19% better than that of Random Offloading. Overall, the results show that the proposed Lagrange approach outperforms the conventional approaches very well in reducing energy, latency, and computation costs, especially in high-congestion conditions.

6. Conclusions and Future Works

In our work, we use the LDROA to jointly provide smart offloading decisions and optimize resources for each task being offloaded at an edge server. This approach ensures that the total available resource of the MEC server is not lesser than the that of the adjusted resource for each task and that the available resource is greater than zero, thus finding the minimum resource provided by the MEC server for computing the task. The execution task must meet the time constraint of mission-critical delay-sensitive signaling, for which the time constraint is 60 ms, while saving energy on user devices. We can see that the performance of our algorithms is much better than that of the models showcased in other existing work, such as Random Offloading, Load Balancing, and Greedy Latency schemes, in terms of total latency and total cost, which means that the proposed method can both save energy in user devices and obtain optimal total latency for IoT applications. The LDROA can make intelligent offloading decisions and optimize resources for tasks on edge servers. This approach ensures that the total available MEC server resources meet the task requirements and time constraints while saving energy on user devices. Our method achieves superior performance by reducing energy consumption in user devices and achieving optimal latency for IoT applications.

In our future work, we will extend our simulation based on the two main conditions of different system bandwidths and the transmission power of user devices. We will conduct simulations to evaluate the performance of Lagrange Duality optimization based on the system bandwidth provided by the base station for each user device. We will vary the transmitting power of user devices, the bandwidth capacities, and the congestion condition to analyze how these factors affect the system's overall performance. By extending the simulation to cover different scenarios, we can gain more insights into how the system performs under various conditions and identify potential problems that may affect system performance.

Author Contributions: Conceptualization, C.E. and S.K. (Seokhoon Kim); methodology, C.E. and S.R.; software, S.K. (Seungwoo Kang), I.S. and P.T.; validation, S.R., S.K. (Seungwoo Kang), I.S. and P.T.; formal analysis, C.E. and S.R.; investigation, S.K. (Seokhoon Kim); resources, S.K. (Seokhoon Kim); data curation, S.R., I.S., P.T. and S.M.; writing—original draft preparation, C.E. and S.R.; writing—review and editing, S.R. and S.K. (Seokhoon Kim); visualization, S.R., S.K. (Seungwoo Kang) and S.M.; supervision, S.K. (Seokhoon Kim); project administration, S.K. (Seokhoon Kim); funding acquisition, S.K. (Seokhoon Kim). All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by Institute of Information and Communications Technology Planning and Evaluation (IITP) grant funded by the Korean government (MSIT) (No. RS-2022-00167197, Development of Intelligent 5G/6G Infrastructure Technology for The Smart City), in part by the National Research Foundation of Korea (NRF), Ministry of Education, through Basic Science Research Program under Grant NRF-2020R1I1A3066543, in part by BK21 FOUR (Fostering Outstanding Universities for Research) under Grant 5199990914048, and in part by the Soonchunhyang University Research Fund.

Data Availability Statement: Derived data supporting the findings of this study are available from the corresponding author on request.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Brown, G. *Mobile Edge Computing Use Cases and Deployment Options*; Juniper White Paper; Heavy Reading: New York, NY, USA, 2016.
2. Dolezal, J.; Becvar, Z.; Zeman, T. Performance evaluation of computation offloading from mobile device to the edge of mobile network. In Proceedings of the 2016 IEEE Conference on Standards for Communications and Networking (CSCN), Berlin, Germany, 31 October–2 November 2016.
3. Kekki, S.; Featherstone, W.; Fang, Y.; Kuure, P.; Li, A.; Ranjan, A.; Purkayastha, D.; Jiangping, F.; Frydman, D.; Verin, G.; et al. *MEC in 5g Networks*; ETSI White Paper; ETSI: Valbonne, France, 2018.
4. Yadav, R.; Zhang, W.; Elgendy, I.A.; Dong, G.; Shafiq, M.; Laghari, A.A.; Prakash, S. Smart Healthcare: RL-Based Task Offloading Scheme for Edge-Enable Sensor Networks. *IEEE Sens. J.* **2021**, *21*, 24910–24918. [CrossRef]
5. Tam, P.; Math, S.; Kim, S. Adaptive partial task offloading and virtual resource placement in SDN/NFV-based network softwarization. *Comput. Syst. Sci. Eng.* **2023**, *45*, 2141–2154. [CrossRef]
6. Shahzadi, S.; Iqbal, M.; Dagiuklas, T.; Qayyum, Z.U. Multi-access edge computing: Open issues, challenges and future perspectives. *J. Cloud Comput.* **2022**, *6*, 30. [CrossRef]
7. Chen, Y.; Zhao, F.; Lu, Y.; Chen, X. Dynamic Task Offloading for Mobile Edge Computing with Hybrid Energy Supply. *Tsinghua Sci. Technol.* **2023**, *28*, 421–432. [CrossRef]
8. Medhat, A.M.; Taleb, T.; Elmangoush, A.; Carella, G.A.; Covaci, S.; Magedanz, T. Service Function Chaining in Next Generation Networks: State of the Art and Research Challenges. *IEEE Commun. Mag.* **2017**, *55*, 216–223. [CrossRef]
9. Ren, Y.; Guo, A.; Song, C. Multi-Slice Joint Task Offloading and Resource Allocation Scheme for Massive MIMO Enabled Network. *KSII Trans. Internet Inf. Syst.* **2023**, *17*, 794–815.
10. Zhang, K.; Mao, Y.; Leng, S.; He, Y.; Zhang, Y. Mobile-Edge Computing for Vehicular Networks: A Promising Network Paradigm with Predictive Off-Loading. *IEEE Veh. Technol. Mag.* **2017**, *12*, 36–44. [CrossRef]
11. Song, I.; Tam, P.; Kang, S.; Ros, S.; Kim, S. DRL-Based Backbone SDN Control Methods in UAV-Assisted Networks for Computational Resource Efficiency. *Electronics* **2023**, *12*, 2984. [CrossRef]
12. Hazarika, B.; Singh, K.; Biswas, S.; Li, C.-P. DRL-Based Resource Allocation for Computation Offloading in IoV Networks. *IEEE Trans. Ind. Inform.* **2022**, *18*, 8027–8038. [CrossRef]
13. Sha, Y.; Hu, J.; Hao, S.; Wang, D. Joint Relay Selection and Resource Allocation for Delay-Sensitive Traffic in Multi-Hop Relay Networks. *KSII Trans. Internet Inf. Syst.* **2022**, *16*, 3008–3028.
14. Tong, L.; Li, Y.; Gao, W. A hierarchical edge cloud architecture for mobile computing. In Proceedings of the IEEE INFOCOM 2016—The 35th Annual IEEE International Conference on Computer Communications, San Francisco, CA, USA, 10–14 April 2016.
15. Fatemi Moghaddam, F.; Rohani, M.B.; Ahmadi, M.; Khodadadi, T.; Madadipouya, K. Cloud computing: Vision, architecture and Characteristics. In Proceedings of the 2015 IEEE 6th Control and System Graduate Research Colloquium (ICSGRC), Shah Alam, Malaysia, 10–11 August 2015.
16. Chen, L.; Tang, H.; Zhao, Y.; You, W.; Wang, K. A Privacy-preserving and Energy-efficient Offloading Algorithm based on Lyapunov Optimization. *KSII Trans. Internet Inf. Syst.* **2022**, *16*, 2490–2506.
17. Pham, Q.-V.; Fang, F.; Ha, V.N.; Piran, M.J.; Le, M.; Le, L.B.; Hwang, W.-J.; Ding, Z. A Survey of Multi-Access Edge Computing in 5G and Beyond: Fundamentals, Technology Integration, and State-of-the-Art. *IEEE Access* **2020**, *8*, 116974–117017. [CrossRef]
18. Wang, Y.; Wan, X.; Du, X.; Chen, X.; Lu, Z. A Resource Allocation Strategy for Edge Services Based on Intelligent Prediction. In Proceedings of the 2021 IEEE 6th International Conference on Smart Cloud (SmartCloud), Virtual, 6–8 November 2021.
19. Ros, S.; Eang, C.; Tam, P.; Kim, S. ML/SDN-Based MEC Resource Management for QoS Assurances. In *Advances in ComputerScience and Ubiquitous Computing*; Springer: Singapore, 2023; Volume 1028, pp. 591–597.
20. Tam, P.; Corrado, R.; Eang, C.; Kim, S. Applicability of Deep Reinforcement Learning for Efficient Federated Learning in Massive IoT Communications. *Appl. Sci.* **2023**, *13*, 3083. [CrossRef]
21. Huynh, L.N.T.; Pham, Q.-V.; Pham, X.-Q.; Nguyen, T.D.T.; Hossain, M.D.; Huh, E.-N. Efficient Computation Offloading in Multi-Tier Multi-Access Edge Computing Systems: A Particle Swarm Optimization Approach. *Appl. Sci.* **2020**, *10*, 203. [CrossRef]
22. Ros, S.; Tam, P.; Kim, S. Modified Deep Reinforcement Learning Agent for Dynamic Resource Placement in IoT Network Slicing. *J. Internet Comput. Serv.* **2022**, *23*, 17–23.

23. Guo, H.; Liu, J.; Qin, H. Collaborative Mobile Edge Computation Offloading for IoT over Fiber-Wireless Networks. *IEEE Netw.* **2018**, *32*, 66–71. [CrossRef]
24. Chen, C.; Zeng, Y.; Li, H.; Liu, Y.; Wan, S. A Multi-hop Task Offloading Decision Model in MEC-enabled Internet of Vehicles. *IEEE Internet Things J.* **2022**, *10*, 3215–3230. [CrossRef]
25. Ryu, J.-W.; Pham, Q.-V.; Luan, H.N.T.; Hwang, W.-J.; Kim, J.-D.; Lee, J.-T. Multi-Access Edge Computing Empowered Heterogeneous Networks: A Novel Architecture and Potential Works. *Symmetry* **2019**, *11*, 842. [CrossRef]
26. Ren, J.; Mahfujul, K.M.; Lyu, F.; Yue, S.; Zhang, Y. Joint Channel Allocation and Resource Management for Stochastic Computation Offloading in MEC. *IEEE Trans. Veh. Technol.* **2021**, *69*, 8900–8913. [CrossRef]
27. Tam, P.; Math, S.; Nam, C.; Kim, S. Adaptive Resource Optimized Edge Federated Learning in Real-Time Image Sensing Classifications. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2021**, *14*, 10929–10940. [CrossRef]
28. Li, J.; Lv, T. Deep Neural Network Based Computational Resource Allocation for Mobile Edge Computing. In Proceedings of the 2018 IEEE Globecom Workshops (GC Wkshps), Abu Dhabi, United Arab Emirates, 9–13 December 2018.
29. Naouri, Y.A.; Wu, H.; Nouri, N.A.; Dhelim, S.; Ning, H. A Novel Framework for Mobile-Edge Computing by Optimizing Task Offloading. *IEEE Internet Things J.* **2021**, *8*, 13065–13076. [CrossRef]
30. Chen, Y.; Zhang, N.; Zhang, Y.; Chen, X.; Wu, W.; Shen, X. Energy Efficient Dynamic Offloading in Mobile Edge Computing for Internet of Things. *IEEE Trans. Cloud Comput.* **2021**, *9*, 1050–1060. [CrossRef]
31. Wang, Z.; Li, P.; Shen, S.; Yang, K. Task Offloading Scheduling in Mobile Edge Computing Networks. *Procedia Comput. Sci.* **2021**, *184*, 322–329. [CrossRef]
32. Zhang, H.; Wu, W.; Wang, C.; Li, M.; Yang, R. Deep Reinforcement Learning-Based Offloading Decision Optimization in Mobile Edge Computing. In Proceedings of the 2019 IEEE Wireless Communications and Networking Conference (WCNC), Marrakesh, Morocco, 15–18 April 2019.
33. Lakew, D.S.; Tuong, V.D.; Dao, N.-N.; Cho, S. Adaptive Partial Offloading and Resource Harmonization in Wireless Edge Computing-Assisted IoE Networks. *IEEE Trans. Netw. Sci. Eng.* **2022**, *9*, 3028–3044. [CrossRef]
34. Wan, Z.; Xu, D.; Xu, D.; Ahmad, İ. Joint Computation Offloading and Resource Allocation for NOMA-Based Multi-Access Mobile Edge Computing Systems. *Comput. Netw.* **2021**, *196*, 108256. [CrossRef]
35. Xu, D.; Xu, D. Cooperative Task Offloading and Resource Allocation for UAV-Enabled Mobile Edge Computing Systems. *Comput. Netw.* **2023**, *223*, 109574. [CrossRef]
36. Xu, D. Device Scheduling and Computation Offloading in Mobile Edge Computing Networks: A Novel NOMA Scheme. *IEEE Trans. Veh. Technol.* **2024**, 1–6. [CrossRef]
37. Abkenar, F.S.; Iranmanesh, S.; Bouguettaya, A.; Raad, R.; Jamalipour, A. ENERGENT: An Energy-Efficient UAV-Assisted Fog-IoT Framework for Disaster Management. *J. Commun. Netw.* **2022**, *24*, 698–709. [CrossRef]
38. Abkenar, F.S.; Ramezani, P.; Iranmanesh, S.; Murali, S.; Chulerttiyawong, D.; Wan, X.; Jamalipour, A.; Raad, R. A Survey on Mobility of Edge Computing Networks in IoT: State-of-The-Art, Architectures, and Challenges. *IEEE Commun. Surv. Tutor.* **2022**, *24*, 2329–2365. [CrossRef]
39. Jan, T.; Iranmanesh, S.; Sajeev, A.S.M. Ensemble of Semi-Parametric Models for IoT Fog Modeling. In Proceedings of the 2019 IEEE Symposium Series on Computational Intelligence (SSCI), Xiamen, China, 6–9 December 2019.
40. Pham, Q.-V.; Leanh, T.; Tran, N.H.; Park, B.J.; Hong, C.S. Decentralized Computation Offloading and Resource Allocation for Mobile-Edge Computing: A Matching Game Approach. *IEEE Access* **2018**, *6*, 75868–75885. [CrossRef]
41. Zhang, K.; Mao, Y.; Leng, S.; Zhao, Q.; Li, L.; Peng, X.; Pan, L.; Maharjan, S.; Zhang, Y. Energy-Efficient Offloading for Mobile Edge Computing in 5G Heterogeneous Networks. *IEEE Access* **2016**, *4*, 5896–5907. [CrossRef]
42. Wang, K.; Yang, K. Power-Minimization Computing Resource Allocation in Mobile Cloud-Radio Access Network. In Proceedings of the 2016 IEEE International Conference on Computer and Information Technology (CIT), Nadi, Fiji, 8–10 December 2016.
43. Tang, J.; Tay, W.P.; Wen, Y. Dynamic Request Redirection and Elastic Service Scaling in Cloud-Centric Media Networks. *IEEE Trans. Multimed.* **2014**, *16*, 1434–1445. [CrossRef]
44. *3GPP TS 23.203, v. 17.1.0*; Technical Specification Group Services and System Aspects. Policy and Charging Control Architecture; ETSI: Valbonne, France, 2021.
45. Fan, X.; Cui, T.; Cao, C.; Chen, Q.; Kwak, K.S. Minimum-Cost Offloading for Collaborative Task Execution of MEC-Assisted Platooning. *Sensors* **2019**, *19*, 847. [CrossRef] [PubMed]