Energy-Latency Computation Offloading and Approximate Computing in Mobile-Edge Computing Networks

Ayman Younis, *Member, IEEE*, Sumit Maheshwari, *Member, IEEE*, and Dario Pompili, *Fellow, IEEE*

***Abstract*—Task offloading with Mobile-Edge Computing (MEC) is envisioned as a promising technique to prolong battery lifetime and enhance the computational capacity of mobile devices. In this paper, we consider a multi-user MEC system with a Base Station (BS) equipped with a computation server that assists users in executing computation-intensive tasks via offloading. Exploiting approximate computing in MEC, we can trade the output accuracy over a subset of offloading data instead of the entire dataset. We formulate the Energy-Latency-aware Task Offloading and Approximate Computing (ETORS) problem, aiming to optimize the trade-off between energy consumption and latency. Due to the mixed-integer nature of this problem, we employ the Dual-Decomposition Method (DDM) to decompose the original problem into three subproblems—namely the Task-Offloading Decision (TOD), the CPU Frequency Scaling (CFS), and the Quality of Computation Control (QoCC). Our approach consists of two iterative layers: in the outer layer, we adopt the duality technique to find the optimal value of the Lagrangian multiplier associated with the primal problem; and in the inner layer, we formulate the subproblems that can be solved efficiently using convex optimization techniques. Simulation results coupled with real-time experiments on a small-scale MEC testbed show the effectiveness of our proposed resource allocation scheme and its advantages over existing approaches.**

***Index Terms*—Mobile Edge Computing; Resource Allocation; Testbed; Computer-vision applications; Convex Optimizations.**

I. Introduction

A. Motivation

In the past decade, there has been a significant surge in the adoption of mobile devices, including mobile phones, tablets, and wearable devices. These devices, characterized by their compact sizes, have experienced substantial growth worldwide. However, their computational capabilities and battery capacities are inherently limited, which poses challenges in effectively handling computationally demanding tasks [2]. Applications that involve complex functionalities, such as Virtual Reality (VR), Augmented Reality (AR), and object tracking and recognition, require greater computing power, memory, and battery longevity on mobile devices [3]. At the same time, with the development of wireless communication technologies such as Wi-Fi and Beyond 5G (B5G), Mobile Edge Computing (MEC) has been proposed as a promising framework design to tackle such challenges. Specifically, MEC servers are installed directly at the Base Stations (BSs) or at the local wireless Access Points (APs) and managed by the mobile network operator. Hence, MEC provides a cloud computing store for the resource-limited end devices to offload their computation tasks to the BSs/APs, where the offloaded tasks can be executed efficiently. This substantially helps reduce the task completion delay and releases the burden on the backhaul networks [4]. The MEC can improve the performance of mobile devices by: (i) selectively offloading tasks of an application (e.g., object/gesture recognition, image/video editing, and natural language processing [5]) onto the cloud, and (ii) carefully scheduling local task executions on the mobile devices with the anticipation of remote task executions in the cloud while taking into account the task-precedence requirements.

Enabling task offloading technology at MEC significantly improves the performance of mobile devices, as the edge servers in MEC are equipped with much higher computation resources compared to end users' devices. In general, the component responsible for making the task offloading decision at each mobile device is called the *task scheduler*. To achieve the optimal decision, the task scheduler should consider the service latency and the transmit power consumption demanded to establish uplink wireless connections between the end users and the edge servers. Besides, in a complex application with multiple tasks, the reduction of computation workload at the edge server has a significant impact on the task execution latency [6]. For instance, real-time gaming applications have a preferred response time between 45 to 75 ms latency to enjoy a higher Quality of Experience (QoE) [7]. This might be achieved by trading off between game's graphical quality and performance.

In general, in some scenarios, processing the arrived data at MEC servers would require more than the accessible computing resources to satisfy the preferred metrics (e.g., latency and throughput). For instance, the traditional processing and analysis of a massive amount of data—collected from IoT devices and transmitted an MEC server—would be prohibitively expensive for handling real-time stream analytics. Moreover, the distributed MEC servers usually are restricted by computing capacity and service latency. Therefore, deploying approximate computing, in which the collected raw data at

A. Younis and D. Pompili are with the Dept. of Electrical and Computer Engineering, Rutgers University–New Brunswick, NJ, 08901, USA. S. Maheshwari is with Microsoft. Emails: {a.younis, pompili}@rutgers.edu, sumitece87@gmail.com;

A preliminary version of this work appeared in the Proc. of the IEEE International Conference on Mobile Ad-Hoc and Smart Systems (MASS), Nov'19 [1].

This work was supported in part by the National Science Foundation (NSF) Grant No. ECCS-2030101.

MEC servers is processed over a representative sample instead of the entire dataset, to meet the MEC constraints would be an efficient method. For example, applying approximate computing to machine learning-based applications such as object detection has emerged as an effective approach to minimize hardware complexity, service latency, and computation energy [8], [9].

B. Our Vision

One of the key challenges is how to fulfill an energy-latency computation offloading policy. To achieve the potential benefits of MEC in terms of service latency and energy saving for end users, the following key aspects need to be considered: (i) end-user computation task type, (i.e., which computation tasks of a users' application require offloading to the MEC server?) (ii) computing capacity (i.e., how much CPU resources should be assigned to each computation task in end-user devices?) and (iii) energy-latency system awareness (i.e., how much processing, transmitting, and management can be saved through approximate computing, with an acceptable loss in accuracy?) *In this paper, we seek to answer the above-mentioned questions by studying the Energy-Latency-aware Task Offloading and Approximate Computing (ETORS) problem.*

Generally, there are two main types of offloading schemes in the MEC systems: binary and partial offloading [10]. In binary offloading, each computation task is either computed locally or offloaded remotely to the MEC server. On the other side, in partial offloading, each computation task can be partially executed both locally and remotely at the MEC servers. *In this article, we consider binary computational offloading since it is widely adopted in real-time low-latency B5G applications [11].* However, in both of these offloading schemes, the computational cost and latency constraints present additional challenges when designing task offloading algorithms in MEC systems. Furthermore, there is a limited understanding of the relationship among approximate computing, computing latency, and energy consumption. Such understanding is crucial for designing cost-efficient computation offloading in MEC networks—indeed the target of this article.

Hence, in this context, the approximate computing technique could be leveraged to maximize the utilization of the available computing resources at the MEC servers by balancing the trade-off between result accuracy and overall performance. Specifically, our aim is to optimize the trade-off between the total energy consumption in a mobile device for executing an application, which can be improved by employing the Dynamic Voltage and Frequency Scaling (DVFS) technique [12], and the application completion time resulting from uploading and downloading data to and from the cloud. We design a task-offloading assignment and approximate-computing framework and implement a MEC testbed to demonstrate our vision for an energy-latency offloading and approximate computing scheme. Specifically, We implemented a programmable 5G testbed that consists of a local server acting as a user and an MEC server functioning as an edge cloud server to create a practical environment for modeling the ETORS problem. Specifically, we generated a real-time Traffic Lane Detection (TLD) application, which involves video streaming and computer vision computation tasks. We then executed several real-time experiments under various scenarios to analyze and evaluate aspects such as data framework size, storage utilization, network latency, and the impact of different quality computation levels.

C. Contributions

This article is an extended version of our earlier work in [1]. Our article presents a substantial enhancement through the implementation of a practical 5G testbed, effectively overcoming the sophisticated system implementation challenges outlined in [1]. The comprehensive analysis presented in this article, Specifically in Sect. V, not only reaffirms the findings from the short version but also provides novel perspectives on the setups of real-time performance evaluation and the nuanced exploration of approximate computing. With a detailed examination of functional splitting, data input profiling, memory utilization, and average processing, this work significantly augments the scope and depth of understanding presented in [1], thus contributing to the evolving landscape of 5G network research. The main objective of this paper is to design the ETORS algorithm, optimizing the trade-off between energy consumption and application completion time. The main contributions of this paper can be summarized as follows:

- Taking into account task dependencies and completion processing time, we formulate the offloading utility of each user by considering the weighted sum of energy consumption and computation latency in both local and remote computing schemes.
- We formulate the ETORS problem as a Mixed-Integer Programming (MIP) problem, which aims to jointly optimize the task offloading decisions, local CPU frequency, and approximate computing accuracy. The objective is to minimize the system offloading utility while considering the trade-off between these variables.
- To solve the optimization problem, we propose a distributed ETORS algorithm consisting of three subproblems—namely the Task Offloading Decision (TOD), CPU Frequency Scaling (CFS), and Quality of Computation Control (QoCC). The ETORS algorithm effectively addresses the optimization challenge by tackling these subproblems in a coordinated manner. More importantly, we have identified that CPU CFS is primarily dependent on several factors, including the computing workload, latency, CPU frequency, and transmission power of the user device. These factors play a crucial role in determining the optimal CPU frequency scaling strategy within the context of the ETORS algorithm.
- We implement our MEC network with a local server that can be Commercial Off-The-Shelf (COTS) user equipment, such as a mobile phone, while the MEC server acts as an edge cloud server and has the computing capacity to perform the ETORS algorithm. Since we assume the radio link between the end user and the

MEC is a 5G connection, we have implemented the 5G programmable testbed. Our testbed can be deployed in the MEC server and is "programmable" since all the PHY and MAC layers are generated by virtual software using OpenAirInterface (OAI) [13], Docker, and USRP Software Defined Radio (SDR) boards.
- Using the real-time Traffic Lane Detection (TLD) application, which involves video streaming and computer vision computation tasks, we conducted several real-time experiments under various scenarios. These experiments allowed us to analyze and evaluate aspects such as data framework size, storage utilization, network latency, and the impact of different quality of computation levels.
- We conducted numerical simulations to evaluate the performance of the proposed distributed ETORS algorithm. In comparison to existing baseline methods, the ETORS method effectively minimizes system cost-utility. This cost-utility is defined by the trade-off between energy consumption and network latency.

The remainder of this article is organized as follows. We present the related previous works in Sect. II. In Sect. III, we introduce the system model; the ETORS problem is formulated in Sect. IV, followed by the decomposition of the problem itself; in Sect. V, we present the experiment results as well as numerical simulation; finally, we conclude the article in Sect. VI.

II. Related Work

Cloud-based wireless networks have garnered significant interest in academia and industry in recent years. As an example, in 2013, Nokia introduced the first real-time MEC framework [14]—a notable development that has further emphasized the growing importance of cloud-based wireless networks. Recently, CloudLab [15] was introduced as a cloud-based research infrastructure that provides a platform for conducting large-scale experiments in the field of computer science and networking.

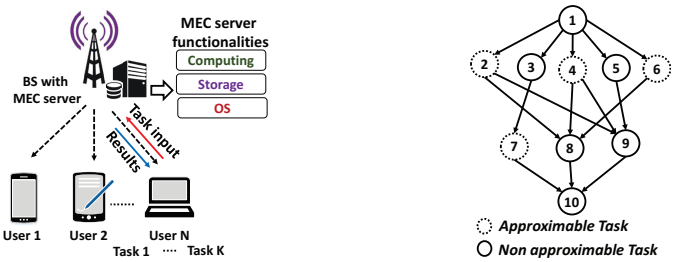
The cloud framework, as well as the radio sections, were fully realized with the Flexi Multiradio BS. In recent times, AT&T MEC has introduced MEC to bring cloud computing capabilities to end-users [16]. The integration of AT&T MEC offers various potential benefits, including low latency, improved connectivity and coverage, and enhanced data security.

Meanwhile, several academic efforts have been directed towards MEC architectures and their applications, such as single-user single MEC server [17] and multi-user single-MEC server [18]. Simultaneously, the problem of computation task offloading has emerged as a common topic and has been extensively investigated in the field of cloud computing. In recent years, researchers have made significant efforts to develop full and/or binary computation offloading schemes aimed at minimizing execution costs in cloud computing. For instance, in a study conducted by Chen et al. [19], an online Task Offloading and Frequency Scaling for Energy Efficiency (TOFFEE) approach was introduced. The TOFFEE framework aimed to minimize energy consumption while ensuring a bounded queue length for applications. In the work in [20], the objective was to minimize execution delay using a one-dimensional search algorithm. This algorithm aimed to find an optimal offloading decision policy based on the application buffer queuing state and the characteristics of the channel between the user and the MEC server. In the work by Wei et al. [21], the authors formulated the optimization problem for task offloading in edge servers, focusing on saving energy on mobile devices. They specifically addressed divisible tasks and employed a greedy approach to tackle the problem.

In [22], the authors proposed a low-complexity Lyapunov Optimization based Dynamic Computation Offloading (LODCO) algorithm, which aimed to reduce the latency for offloaded applications. The problem of computation offloading decisions that minimize energy consumption at the user while satisfying the application's execution delay has been addressed in prior works [23], [24]. The optimization problem in [23] was formulated as a Constrained Markov Decision Process (CMDP). On the other hand, in [24], the decision on computation offloading was made periodically in each time slot. During this process, all users were divided into two groups: the first group was permitted to offload computation to the MEC server, while the second group had to perform computation locally due to the unavailability of computation resources at the MEC server.

Furthermore, in terms of MEC and IoTs, the work in [25] introduced an efficient and secure multi-user multi-task computation offloading model to minimize the weighted sum of energy consumption for mobile IoT devices. The authors in [26] presented an innovative framework that utilizes blockchain technology to ensure secure task offloading in MEC systems. The proposed framework guarantees performance in terms of execution delay and energy consumption. By introducing blockchain technology as a platform, the authors aim to achieve data confidentiality, integrity, authentication, and privacy for task offloading in the MEC environment. Besides, in [27], the authors offered a novel approach to offloading decisions by combining parallel Deep Neural Networks (DNNs) with Q-learning. This integration allows for fine-grained decision-making based on the deep learning models' perception, reinforcement learning's decision-making capabilities, and meta-learning's ability to quickly adapt to changing environments. As a result, the algorithm enables the rapid and flexible identification of the most optimal offloading strategy in dynamic settings. The objective of the study in [28] was to tackle the issue of dynamic slice scaling and task offloading in a multi-tenant edge computing system, focusing on the profit aspect for service providers. The proposed approach involved a Deep Q-learning-based network slicing framework, which enables the dynamic reconfiguration of radio and computing resources allocated to a specific slice reserved for a target service provider.

On the other hand, several existing works [29]–[31] have investigated the trade-off between energy consumption and execution delay for users offloading their applications to the cloud. In [32], the authors proposed an energy-efficient offloading-decision algorithm based on Lyapunov optimization. This algorithm aimed to solve the task offloading as-



(a) (b)

Fig. 1. (a) Illustration of MEC system with multiple mobile devices; (b) A directed acyclic task graph of a task graph.

signment problem, wherein the application's tasks would be executed either locally or remotely based on the available wireless network resources. The authors in [30] considered the computation offloading decision in a multi-user multi-channel environment, taking into account the weighting parameter. The authors in [31] formulated the trade-off between energy saving and computing performance problem as as a non-convex Quadratically Constrained Quadratic Program (QCQP). Subsequently, they proposed an efficient three-step algorithm, which consisted of semidefinite relaxation, alternating optimization, and sequential tuning.

In terms of machine learning and task offloading, several recent works have demonstrated significant potential in addressing task offloading in MEC systems. The authors in [33] introduced an optimal computation offloading policy for MEC in an ultradense system. The proposed policy was based on a Deep Q-network (DQN) and did not require prior knowledge of dynamic statistics. The work presented in in [34] proposed a deep reinforcement learning approach that leveraged a DNN technique to train and improve task offloading schemes based on experimental outcomes. This approach effectively addressed the challenge of dealing with high complexity in optimization problem analyses, as it eliminated the need for intricate optimization procedures. However, these previous works make the assumption of non-adjustable processing capabilities of the CPUs in mobile devices. This assumption is not energy-efficient, as it disregards the fact that CPU energy consumption increases super-linearly with the CPU-cycle frequency [35]. By overlooking the potential energy-saving benefits of adjusting the CPU-cycle frequency, these works may not fully optimize the energy consumption aspect of task offloading strategies. Furthermore, the previous works on energy and time-aware mobile cloud offloading did not propose dynamic offloading algorithms that take into account resource scheduling and approximate computing in the context of real-time applications. The lack of such dynamic offloading algorithms limits the ability to effectively optimize resource allocation and leverage approximate computing techniques to meet the requirements of real-time applications. To address this gap, this paper proposes a novel approach that incorporates dynamic offloading, resource scheduling, and approximate computing to optimize the energy and time efficiency of mobile cloud offloading for real-time application scenarios.

TABLE I
SUMMARY OF KEY NOTATIONS.

Symbol	Description
$\mathcal{N}$	set of mobile users
$\mathcal{K}$	set of computational tasks
r_{ik}	achievable offloading rate for task k of user i
B_{ik}	bandwidth available for user i of offloading task k
p_{ik}	transmission power for user i of offloading task k
h_{ik}	channel power gain from user i transmitting task k to the BS
d_{ik}	data input size for task k of user i
c_{ik}	computing workload required for completing task k of user i
f_{ik}	CPU-clock frequency of mobile user i on task k
T_{ik}^{l}	local execution time for task k of mobile user i
E_{ik}^{l}	local energy consumption for task k of user i
$T_{ik}^{c,tr}$	transmission time of user i offloading task k
$E_{ik}^{c,tr}$	transmitted energy for user i of offloading task k
$T_{ik}^{c,exe}$	computation execution time of task k of user i on the MEC serve
CT_{im}^{l}	completion time of local execution of task m of user i
CT_{im}^{t}	wireless task transmission time from user i to the MEC serve
CT_{im}^{c}	MEC server execution time of task m of user i
CT_{im}^{r}	wireless reception time of task m from the MEC server to user i
RT_{ik}^{l}	locally completed execution time for task k of user i
δ_{ik}^{c}	computation time weight for user i of task k
δ_{ik}^{e}	energy weight for user i of task k

III. System Model

In this section, we first describe the network setting, communication scheme, and scheduling model; then, we present the task-precedence requirements in the MEC network. Table I summarizes the key notations used in this article.

A. Network Description

We consider a general MEC network consisting of one BS/AP equipped with a MEC server, which provides computation offloading services to a set of $\mathcal{N} = \{1, 2, ..., N\}$ resource-constrained mobile users. Each user has a computation-intensive application that needs to be executed. We assume that the end-user application can be divided into a set of K computation tasks, denoted by $\mathcal{K} = \{1, 2, ..., K\}$. Furthermore, each computation task can be carried out either on the end-user device or offloaded to the MEC server, as illustrated in Fig. 1(a). In addition, we utilize a directed acyclic graph $G(V, E)$ to depict the relationship among computation tasks within an application. Each node $i \in V$ in G represents a task and a directed edge $e(i, j) \in E$ indicates the precedence constraint such that task (node) i should complete its execution before the task (node) j starts execution. There are a total number of K tasks (nodes) in the task graph (application). Typically, K is smaller than 100, e.g., in computer-vision applications, K often falls within the range of $10 \sim 30$ [36]. In our assumption, we classify tasks into two different categories: (i) *approximable*, tasks that can be approximated to achieve

significant savings in energy and/or execution time. However, this approximation may result in a potential loss of accuracy in the final outcome; and (ii) *non-approximable*, tasks require exact execution without any form of approximation for the application to succeed. In other words, if any approximation technique were applied to these tasks, the application would not produce meaningful results.

We refer interested readers to our recent work [6], which introduces a lightweight online algorithm that selects between these tasks to enable real-time distributed applications on resource-limited devices. The random Layer-by-Layer relations among 10 computation tasks of an application are depicted in Fig. 1(b). In this illustration, tasks 2, 4, and 6 are the immediate predecessors of task 8, while task 10 is the successor/descendant of task 8.

B. Wireless Link Model

We consider that each user can establish a cellular link with the BS. Moreover, we assume that the locations of users remain unchanged and that wireless channels are invariant during each decision-making procedure. We denote $a_{ik} \in \{0, 1\}$ to represent the offloading indicator of task k of end-user i. Thus, $a_{ik} = 1$ indicates that user i decides to upload the computation of task k to the MEC server via wireless network channel, while $a_{ik} = 0$ denotes that user i chooses to execute task k on its local device. The achievable offloading rate $r_{ik}[\text{bps}]$ for task k of user i is given by,

$$r_{ik} = B_{ik} \log_2 \left(1 + \frac{p_{ik} h_{ik}}{\Upsilon \sigma^2}\right), \forall i \in \mathcal{N}, k \in \mathcal{K}, \quad (1)$$

where B_{ik} and p_{ik} denote the transmission bandwidth and power of user i offloading task k to the BS, respectively, h_{ik} indicates the effective channel power gain from user i to the BS when transmitting task k, σ^2 is the noise power at the receiver of the BS, and $\Upsilon \geq 1$ is a constant accounting for the gap from channel capacity due to a practical coding and modulation scheme. For simplicity, we assume $\Upsilon = 1$.

C. Computation Model

We denote d_{ik} as the size of input data (including system configurations, program codes, and profiling parameters) of user i with task k, while the c_{ik} represents the workload, i.e., the amount of CPU computation demanded for performing the task.

1) Local Computing: We denote f_{ik} as the CPU-clock frequency of mobile user i on task k. In addition, we assume that user devices have several computation capacities, where each task of mobile users requires a different CPU-clock frequency to be executed. The local execution time of task k of mobile user i can be expressed as,

$$T^l_{ik} = c_{ik}/f_{ik}, \forall i \in \mathcal{N}, k \in \mathcal{K}, \quad (2)$$

while the related local computing energy can be determined by,

$$E^l_{ik} = \epsilon_i c_{ik} f^2_{ik}, \forall i \in \mathcal{N}, k \in \mathcal{K}, \quad (3)$$

where ϵ_i is the energy coefficient depending on the chip architecture. From (3), we conclude that the local computing energy significantly increases as CPU frequency increases. Therefore, applying the optimization to the DVFS technique, in which the CPU core is enabled to perform at several frequency values, can efficiently reduce the overall computing cost.

2) Approximate Computing: We denote q_{ik} as the approximate computing level assigned to task k of user i. In our proposed framework, we allow each user i to select different q_{ik} values to exploit the trade-off between processing cost and latency. We consider y^*_{ik} and y_{ik} as the highest achievable accuracy and the actual accuracy, respectively, that can be attained through exact and approximate computing for task k of user i.

Then, $q_{ik} = 1 - \frac{y^*_{ik} - y_{ik}}{y^*_{ik}} \in [0, 1]$, where $q_{ik} = 0$ when the the result from task i is fully approximated, whereas $q_{ik} = 1$ when the result from task i is exactly processed. In practice, the set of measuring metric q_{ik} for an application is calculated for each application domain. For instance, let task k of mobile user i be a part of an object recognition application with identifying exact accuracy being 98% (i.e. on average only 1 object is falsely identified out of the 50 objects by the exact algorithm). With approximate computing, if the accuracy is 90% (i.e., on average 45 out of the 50 objects are correctly identified), then q_{ik} is measured by normalized accuracy as $90\%/98\% = 0.9184$.

3) Cloud Computing: In case user i offloads its task k to the MEC server, the execution process of task k in the MEC server comprises three sequential time periods: (i) transmitting time to transmit the configuration and input workload of task k to the MEC server via uplink cellular channel, (ii) task execution time in the MEC server, and (iii) receiving time to return the processing workload from the MEC server to end-users via downlink cellular channel. We can compute the transmission time of user i offloading task k by,

$$T^{c,tr}_{ik} = (q_{ik} d_{ik})/r_{ik}, \quad (4)$$

while the related cloud computing energy can be calculated by,

$$E^{c,tr}_{ik} = p_{ik} T^{c,tr}_{ik}. \quad (5)$$

Moreover, we can model the computation execution delay of task k of user i on the MEC server as linearly increasing function of q_{ik}, expressed as,

$$T^{c,exe}_{ik} = (c_{ik} \varphi(q_{ik}))/f_c, \quad (6)$$

where $\varphi(q_{ik})$ is defined in detail in Theorem 1 and Sect. V-A. f_c denotes the CPU-clock frequency assigned on the MEC server. In this work, f_c is assumed to be given and static throughout the execution cycle. In practice, the MEC servers can exchange multiple key parameters, such as CPU clock frequency of the servers and network settings, by Request-To-Send (RTS)/Clear-To-Send (CTS) packages [37].

D. Task-Precedence Requirements in MEC

We use CT^l_{im}, CT^t_{im}, CT^c_{im}, CT^r_{im}, $\forall m \in \mathbf{prd}(k)$, where $\mathbf{prd}(k)$ represents the sequence of immediate predecessors of task k, to denote the local completion time, the task

transmission time from user's device to the MEC server via the wireless channel, the task execution time in the MEC server, and the reception time of the executed result of task m from the MEC server to end-user i via the wireless channel, respectively. If the task m of user i is scheduled locally, we set $CT^t_{im} = CT^c_{im} = CT^r_{im} = 0$; otherwise, if the task m of user i is offloaded onto the MEC server, we set $CT^l_{im} = 0$. Before we schedule a task k, all its immediate predecessors must have already been scheduled.

1) Local Scheduling: Suppose that task k is to be scheduled locally. Then the *ready time*, earliest time when all immediate predecessors of the task have completed execution, of task k can be calculated as,

$$RT^l_{ik} = \max_m \max\{CT^l_{im}, CT^r_{im}\}, \forall i \in \mathcal{N}, m \in \mathbf{prd}(k). \quad (7)$$

From (7), it can be noted that if an immediate precedent task m of task k is locally performed, then $\max\{CT^l_{im}, CT^r_{im}\} = CT^l_{im}$. In this case we have $RT^l_{ik} \geq CT^l_{im}$, which means that task k will begin local performance only after the local processing of task m has finished. Otherwise, task m can be executed remotely onto the MEC server and, then $\max\{CT^l_{im}, CT^r_{im}\} = CT^r_{im}$. In this case, we have $RT^l_{ik} \geq CT^r_{im}$, which means that task k can begin local performance only after the output result of task m completely transmitted to user i via the downlink wireless channel. Therefore, RT^l_{ik} can be reformulated as,

$$RT^l_{ik} \geq (1 - a_{im})CT^l_{im} + a_{im}CT^r_{im}, \forall m \in \mathbf{prd}(k). \quad (8)$$

Since the size of the output is generally much smaller than the input, and the downlink data rate is much higher than that of the uplink, we omit the time and energy consumption for transferring the output in our computation, as also done in [38], [39]. Accordingly, (8) can be rewritten as,

$$RT^l_{ik} \geq (1 - a_{im})CT^l_{im} + a_{im}CT^c_{im}, \forall m \in \mathbf{prd}(k), \quad (9)$$

From (9), it can be observed that if not accounting for the data receiving time of task m, task k can begin performance only after task m has finished its process. The local completion time to carry out task k on user device i can be modeled as the sum of the local computation time and the ready time in local processing as,

$$CT^l_{ik} = T^l_{ik} + RT^l_{ik}. \quad (10)$$

2) Cloud Scheduling: If task k is offloaded to the MEC server, the *ready time* of task k of user i on the server can be calculated as similar to Sect. III-D1, which can be modeled as,

$$RT^c_{ik} = \max\{CT^t_{ik}, \max_m CT^c_{im}\}, \forall i \in \mathcal{N}, m \in \mathbf{prd}(k), \quad (11)$$

where $\max_m CT^c_{im}$ is the required time for all immediate predecessors of task k offloaded to the MEC server to be completely executed remotely.

Besides, it can be noted that the MEC server can begin performing task k only after the task has been completely uploaded remotely or all the immediate predecessors of task m have been completely performed on the server, i.e., $RT^c_{ik} \geq CT^t_{ik}, RT^c_{ik} \geq \max_m CT^c_{im}$. Basically, in case of disregarding the receiving time of data output of task k, the completion time of task k of user i in the MEC server can be formulated as the sum of the execution time of task k of user i and its ready time on the server, i.e.,

$$CT^c_{ik} = T^{c,exe}_{ik} + RT^c_{ik}. \quad (12)$$

IV. Problem Formulation

We present here the completed details about the ETORS problem $(\mathcal{P}1)$, which is cast as a Mixed-Integer Programming (MIP) problem to optimize the trade-off between energy consumption and the task completion delay while offloading tasks in the MEC server. Due to the intractability of the problem and the need for a practical solution, we then present a step-by-step distributed solution based on the Dual-Decomposition Method (DDM) [40], which is employed to decouple the original problem into three subproblems. Our framework can be summarized as follows.

- In Sects. IV-A and IV-B, we formulate the ETORS problem $\mathcal{P}1$ as a MIP, which is NP-hard and complex to solve.
- In Sect. IV-C, to deal with the complexity in $\mathcal{P}1$, we use the features of the DDM method, in which the primal problem can be divided into equivalent optimization subproblems—namely the Task-Offloading Decision (TOD), the CPU Frequency Scaling (CFS), and the Quality of Computation Control (QoCC).
- Finally, in Sect. IV-D, we introduce a dual form-based algorithm to solve each subproblem and update the relevant Lagrangian multipliers.

A. Energy-Completion Time Utility

Given the offloading decision profile $\mathcal{A} = \{a_{ik} | i \in \mathcal{N}, k \in \mathcal{K}\}$, the local CPU frequency $\mathcal{F} = \{f_{ik} | i \in \mathcal{N}, k \in \mathcal{K}\}$, and the approximate computing level $\mathcal{Q} = \{q_{ik} | i \in \mathcal{N}, k \in \mathcal{K}\}$, the energy-completion time utility of user i is defined as,

$$\begin{aligned} U_{ik}(\mathcal{A}, \mathcal{F}, \mathcal{Q}) = \delta^e_{ik} \left[(1 - a_{ik})E^l_{ik} + a_{ik}E^{c,tr}_{ik}\right] \\ + \delta^c_{ik} \left[(1 - a_{ik})CT^l_{ik} + a_{ik}CT^c_{ik}\right], \end{aligned} \quad (13)$$

where $0 \leq \delta^e_{ik} \leq 1$ and $0 \leq \delta^c_{ik} \leq 1$ represent the weights of energy consumption and execution completion delay for device user i carrying out task k, respectively.

Note that we define the energy-completion time utility of user i as a linear combination of the two metrics because both of them can concurrently reflect the energy-latency trade-off of performing a computation task. In other words, increasing network energy consumption and computation completion delay produce prohibitive total costs. To satisfy the mobile device requirements, we model the utility function to let each end-user choose various weights, which are denoted by δ^e_{ik} and δ^c_{ik}, in the ETORS problem. For instance, a mobile device with low energy capability would prefer selecting a high value of δ^e_{ik} to save more energy. Otherwise, the mobile device would select a high value of δ^c_{ik} to reduce the latency, when it is running a latency-aware application.

B. The ETORS Problem

We define the total system cost as the weighted sum of total network energy, the cost to process a task in local computing and the cost to offload a task in the MEC server, and the execution completion time of running a computation task locally or remotely. Then, we aim to formulate our problem to assign the computation tasks required to be offloaded on the MEC server, as well as determine the CPU clock frequency of each computation task in the case of performing locally on a mobile device, such that the total system cost is minimized. Accordingly, the main optimization problem can be formulated as follows,

$$\mathcal{P}1: \min_{\mathcal{A},\mathcal{F},\mathcal{Q}} \sum\nolimits_{i\in\mathcal{N}}\sum\nolimits_{k\in\mathcal{K}} U_{ik}(\mathcal{A},\mathcal{F},\mathcal{Q}) \tag{14a}$$

$$\text{s.t}: \forall i\in\mathcal{N}, k\in\mathcal{K}, m\in \mathbf{prd}(k),$$

$$\sum\nolimits_{k\in\mathcal{K}}\left[(1-a_{ik})CT^l_{ik}+a_{ik}CT^c_{ik}\right]\le T_i^{max}, \tag{14b}$$

$$(1-a_{im})CT^l_{im}+a_{im}CT^c_{im}\le RT^l_{ik}, \tag{14c}$$

$$CT^t_{ik}\le RT^c_{ik}, \tag{14d}$$

$$\max_m\ CT^c_{im}\le RT^c_{ik}, \tag{14e}$$

$$a_{ik}\in\{0,1\}, f_{ik}\in[0,F_i^{max}], q_{ik}\in[Q_i^{min},1], \tag{14f}$$

where F_i^{max} and Q_i^{min} denote the maximum CPU frequency, and the minimum approximate computing level that is acceptable to mobile user i, respectively. In practice, both CPU frequency f_{ik} and approximate computing level q_{ik} can only be integers chosen from finite sets. However, such integer variables may make the design problem a mixed-integer one, which is NP-hard in general. To avoid this, we model f_{ik} and q_{ik} as continuous variables to provide an upper-bound performance for the practical cases with discrete CPU frequencies and approximate computing levels. The constraints of optimization problem $\mathcal{P}1$ can be explained as follows: constraint (14b) guarantees that the total completion time of all the tasks of an application of user i cannot exceed required maximum time deadline, T_i^{max}; Constraint (14c) implies that task k can only begin execution after all its immediate predecessors have been completed.

constraints (14d) and (14e) represent the MEC server task-precedence specific constraints, i.e., task k can only begin execution on the MEC server after the computation task has been completely uploaded to the server.

Next, we propose a distributed approach to solve $\mathcal{P}1$ based on DDM, as described in $\mathcal{P}2$-$(a\sim c)$. The complete process of transforming the original ETORS problem to obtain its suboptimal solution is shown in Fig. 2.

C. Our Distributed Solution

From Equation (14), it can be observed that the key challenge to achieve the optimal solution of $\mathcal{P}1$ is the variable indicator $a_{ik}\in\{0,1\}$, which makes the problem a non-convex MIP problem and NP-complete [41].

To this end, as in [12], [42], we first relax the binary computation offloading indicator a_{ik} to a real number between 0 and 1, i.e., $0\le a_{ik}\le 1$. Further discussion regarding the convexity of $\mathcal{P}1$ with the relaxed variable a_{ik} will be presented later in the next sections. The Lagrangian associated with $\mathcal{P}1$ is given as,

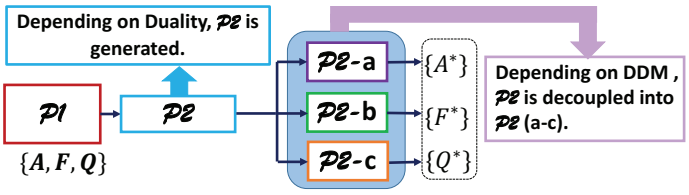


Fig. 2. The complete process to solve the ETORS optimization problem $\mathcal{P}1$ via decoupling into $\mathcal{P}2(a\sim c)$.

$$\begin{aligned}\mathcal{L}(\mathcal{A},\mathcal{F},\mathcal{Q},\boldsymbol{\lambda},\boldsymbol{\mu}) &= \sum\nolimits_{i\in\mathcal{N}}\sum\nolimits_{k\in\mathcal{K}} U_{ik}(\mathcal{A},\mathcal{F},\mathcal{Q})\\ &+\sum\nolimits_{i\in\mathcal{N}}\lambda_i\sum\nolimits_{k\in\mathcal{K}}\left[(1-a_{ik})CT^l_{ik}+a_{ik}CT^c_{ik}\right]\\ &-\sum_{i\in\mathcal{N}}\lambda_i T_i^{max}+\sum_{i\in\mathcal{N}}\sum_{k\in\mathcal{K}}\mu_{ik}[CT^t_{ik}-RT^c_{ik}],\end{aligned} \tag{15}$$

where $\boldsymbol{\lambda}=\{\lambda_i|\forall i\in\mathcal{N}\}$ and $\boldsymbol{\mu}=\{\mu_{ik}|\forall i\in\mathcal{N},k\in\mathcal{K}\}$ are the Lagrangian multipliers.

Accordingly, the primal problem $\mathcal{P}1$ can be addressed through two layers of optimization, an outer layer for updating dual variables and an inner layer for determining the optimization variables [43]. Specifically, the dual problem is expressed as,

$$\mathcal{P}2: \max_{\boldsymbol{\lambda},\boldsymbol{\mu}}\min_{\mathcal{A},\mathcal{F},\mathcal{Q}}\mathcal{L}(\mathcal{A},\mathcal{F},\mathcal{Q},\boldsymbol{\lambda},\boldsymbol{\mu}) \tag{16a}$$

$$\text{s.t. (14c), (14e), (14f).} \tag{16b}$$

In the inner minimization layer, we will solve three subproblems to achieve the optimal variables $(\mathcal{A}^*,\mathcal{F}^*,\mathcal{Q}^*)$ by following the Karush-Kuhn-Tucker (KKT) conditions [44]. In the outer maximization layer, we will update the dual parameters using the subgradient method. In the following, we describe the three distributed subproblems—namely the Task Offloading Decision (TOD), CPU Frequency Scaling (CFS), and Quality of Computation control (QoCC).

1) Task Offloading Decision (TOD): In this subproblem, our objective is to determine the computation tasks of an application that should be executed at the MEC server. The goal is to reduce the energy-latency cost of the application process while satisfying the task-precedence constraint. Consequently, the TOD optimization problem can be formulated as follows:

$$\mathcal{P}2\text{-}a: \min_{\mathcal{A}}\ (1-a_{ik})\Psi^l_{ik}+a_{ik}\Psi^c_{ik} \tag{17a}$$

$$\text{s.t. (14c), (14e), (14f)} \tag{17b}$$

where $\Psi^l_{ik}=\delta^e_{ik}E^l_{ik}+(\delta^c_{ik}+\lambda_i)CT^l_{ik}$ and $\Psi^c_{ik}=\delta^e_{ik}E^{c,tr}_{ik}+(\delta^c_{ik}+\lambda_i)CT^c_{ik}$ can be considered local and the MEC server cost, respectively. Obviously, the objective function in $\mathcal{P}2$-a can be cast as a linear function on indicator a_{ik}. Hence, we can formulate the task-offloading selecting policy as,

$$a_{ik}=\begin{cases}1 & \text{if } \Psi^c_{ik}<\Psi^l_{ik},\\ 0 & \text{otherwise.}\end{cases}\ \forall i\in\mathcal{N},k\in\mathcal{K}. \tag{18}$$

Eq. (18) implies that when the energy-latency cost on the MEC server is less than on the local device, it is profitable to execute task k remotely; otherwise, it should be processed locally.

2) CPU Frequency Scaling (CFS): In this subsection, we aim to formulate the CFS subproblem to optimally adjust the CPU clock frequency of the user device so that the local energy consumption of performing an application can be minimized. Obviously, the CFS subproblem is active when it is decided to locally execute the computation task (i.e., $a_{ik} = 0$). Thus, the CFS subproblem is formulated as,

$$\mathcal{P}2\text{-}b : \min_{\mathcal{F}} \delta^e_{ik} E^l_{ik} + \delta^c_{ik} CT^l_{ik} + \lambda_i (CT^l_{ik} - T^{max}_i) + \mu_{ik}(CT^t_{ik} - RT^c_{ik}) \quad (19a)$$

$$\text{s.t. } (14c), (14e) \quad (19b)$$

Proposition 1. *The optimization problem $\mathcal{P}2$-b is convex with respect to the optimization variable set $\mathcal{F}$.*

Proof. Since the $\frac{\partial^2}{\partial f^2_{ik}} = 2\left(\delta^e_{ik}\epsilon_i + \frac{(\delta^e_{ik}+\lambda_i)}{f^3_{ik}}\right) > 0, \forall i = k$, and $\frac{\partial^2}{\partial f^2_{ik}} = 0, \forall i \neq k$. Hence, the Hessian matrix is positive semidefinite, and thus the objective function of $\mathcal{P}2$-b w.r.t optimization variable f_{ik} is convex [44]. Since constraints (14c) and (14e) are affine, we can state that $\mathcal{P}2$-b is convex. The proof is complete. ■

It is known that T^{max}_i, RT^l_{ik}, CT^t_{ik}, and RT^c_{ik} are independent of f_{ik}. Then, the solution of $\mathcal{P}2$-b can be achieved by exploiting standard convex techniques and the KKT conditions [44]. Hence, the CPU frequency scaling policy can be calculated as,

$$f_{ik} = \sqrt[3]{(\delta^c_{ik} + \lambda_i)/2\epsilon_i \delta^e_{ik}} \qquad , \forall i \in \mathcal{N}, k \in \mathcal{K}. \quad (20)$$

We can see from (20) that the CPU frequency of the mobile device i executing task k depends on the weight parameters, δ^e_{ik} and δ^c_{ik}, and the price factor for the required application completion deadline, λ_i.

3) Quality of Computation Control (QoCC): The objective of this subproblem is to determine the optimal setting of the approximate computing level, q_{ik}, for the MEC server. This optimization aims to minimize the energy cost of the MEC server and the response time required for task execution. It is important to note that the QoCC subproblem is applicable only when the mobile device decides to execute the computation task at the MEC server (i.e. $a_{ik} = 1$).

Hence, it can be formulated as,

$$\mathcal{P}2\text{-}c : \min_{\mathcal{Q}} \delta^e_{ik}\frac{p_{ik} q_{ik} d_{ik}}{r_{ik}} + \delta^c_{ik}\left(\frac{c_{ik}}{f_c}\varphi(q_{ik}) + RT^c_{ik}\right) + \lambda_i(CT^c_{ik} - T^{max}_i) + \mu_{ik}(CT^t_{ik} - RT^c_{ik}) \quad (21a)$$

$$\text{s.t. } (14c), (14e). \quad (21b)$$

We denote $J(q_{ik})$ as the objective function in $\mathcal{P}2$-c. According to (4)–(6), (11), and (12), we can model $J(q_{ik})$ of two forms depend on two various values of RT^c_{ik} as follows.

Case 1: $RT^c_{ik} = CT^t_{ik}$, i.e., $CT^t_{ik} > \max\limits_{m \in \mathbf{prd}(k)} CT^c_{im}$. After assigning $CT^t_{ik} = CT^{c,tr}_{ik} + RT^{tr}_{ik}$, where RT^{tr}_{ik} represents the ready time of task k being offloaded on the wireless channel to ensure the task-precedence requirement, which can be given as constant for task k. Then, $J(q_{ik})$ can be formulated as,

$$J(q_{ik}) = \xi^c_{ik}\left(\frac{c_{ik}}{f_c}\varphi(q_{ik})\right) + \xi^e_{ik}\frac{q_{ik} d_{ik}}{r_{ik}}, \quad (22)$$

where $\xi^c_{ik} = \delta^c_{ik} + \lambda_i$, $\xi^e_{ik} = \xi^c_{ik} + p_{ik}\delta^e_{ik}$, and T^{max}_i is constant.

Case 2: $RT^c_{ik} = \max\limits_{m \in \mathbf{prd}(k)} CT^c_{im}$, i.e., $CT^t_{ik} \leq \max\limits_{m \in \mathbf{prd}(k)} CT^c_{im}$. Since $\max\limits_{m \in \mathbf{prd}(k)} CT^c_{im}$ is the time when all the immediate predecessors of task k that are transmitted to the MEC server have accomplished their performance on the server, the RT^c_{ik} can be considered as a constant. Hence, $J(q_{ik})$ can be written as,

$$J(q_{ik}) = \eta^c_{ik}\left(\frac{c_{ik}}{f_c}\varphi(q_{ik})\right) + \eta^e_{ik}\frac{q_{ik} d_{ik}}{r_{ik}}, \quad (23)$$

where $\eta^c_{ik} = \delta^c_{ik} + \lambda_i$ and $\eta^e_{ik} = \delta^e_{ik} p_{ik} + \mu_{ik}$.

It can be observed from (22) and (23) that the objective function $J(q_{ik})$ has the same form for the two cases, but with different weight factors. Next, we give the approximate computing policy for these cases.

Theorem 1. *In the two cases (1 and 2), the objective function $J(q_{ik})$ is convex with respect to q_{ik} when $\varphi(q_{ik}) = \tau^c_{ik} q_{ik} + \tau^e_{ik}$ for some constants $\tau^c_{ik}, \tau^e_{ik}, \forall i \in \mathcal{N}, k \in \mathcal{K}$. The parameters τ^c_{ik} and τ^e_{ik} can be estimated by offline profiling of the MEC testbed, as detailed in Sect. V-A, Table II.*

Proof. Based on the experimental simulations from Figs Figs. 5(a), 5(b), and 5(c), significant variations in memory, CPU, and latency can be observed corresponding to different levels of computation quality. We can conclude that the function $\varphi(q_{ik})$ can be modeled as a linear equation, $\varphi(q_{ik}) = \tau^c_{ik} q_{ik} + \tau^e_{ik}$, where the constant parameters τ^c_{ik} and τ^e_{ik} can be estimated by fitting the data from Figs. 5(a), (b), and (c) for memory and CPU utilization, as well as latency time. The values used for estimation are provided in Table II. To prove that the function $J(q_{ik})$ in (23) is convex, we need to show that the second derivative of $J(q_{ik})$ with respect to q_{ik} is non-negative for all q_{ik}. First, let's find the second derivative of $J(q_{ik})$ with respect to q_{ik}. Using the power rule and chain rule, we have:

$$\frac{d^2 J}{dq^2_{ik}} = \frac{d}{dq_{ik}}\left(\eta^c_{ik}\left(\frac{c_{ik}}{f_c}\varphi(q_{ik})\right) + \eta^e_{ik}\frac{q_{ik} d_{ik}}{r_{ik}}\right) \quad (24)$$

Simplifying the expression, we get:

$$\frac{d^2 J}{dq^2_{ik}} = \eta^c_{ik}\left(\frac{c_{ik}}{f_c}\frac{d\varphi}{dq_{ik}}\right) + \eta^e_{ik}\frac{d_{ik}}{r_{ik}} \quad (25)$$

Since c_{ik}, f_c, $\varphi(q_{ik})$, η^c_{ik}, d_{ik}, r_{ik}, and η^e_{ik} are all non-negative constants, the only term that may vary with q_{ik} is $\frac{d\varphi}{dq_{ik}}$. And since $\frac{d\varphi}{dq_{ik}} > 0 \forall q_{ik}$. Hence $J(q_{ik})$ is convex. Since constraints (14c) and (14e) are affine, we can state that $\mathcal{P}2$-c is convex. The proof is complete.

■

Since $\mathcal{P}2$-c is convex, then we can use a standard optimization solver such as MOSEK to solve the problem.

D. Lagrangian Multiplier Update

We can employ the subgradient method to solve the master problem in (16). Specifically, a subgradient of the problem for λ_i can be written as,

$$z_i^{\lambda} = \sum_{k \in \mathcal{K}} \left[(1 - a_{ik}) CT_{ik}^{l} + a_{ik} CT_{ik}^{c}\right] - T_i^{max}, \tag{26}$$

and for μ_{ik} is,

$$z_{ik}^{\mu} = \sum_{k \in \mathcal{K}} [CT_{ik}^{t} - RT_{ik}^{c}]. \tag{27}$$

Then, we can update λ_i and μ_{ik} as,

$$\lambda_i^{[t+1]} = \left[\lambda_i^{[t]} - \tau_{\lambda}^{[t]} z_i^{\lambda}\right]^{+}, \ \mu_{ik}^{[t+1]} = \left[\mu_{ik}^{[t]} - \tau_{\mu}^{[t]} z_{ik}^{\mu}\right]^{+}, \tag{28}$$

where $\tau_{\lambda}^{[t]}$, $\tau_{\mu}^{[t]}$ are the step length at iteration step [t], and $[x]^{+} = max\{0, x\}$ denotes the projection function to the nonnegative orthant. Then, we can utilize the new Lagrange multipliers in (28) for updating the computation offloading policy in $\mathcal{P}2(a \sim c)$. The proposed ETORS algorithm is summarized in Algorithm 1, in which the time complexity of the algorithm for user i is $\mathcal{O}(KI_{max}I_{ac})$, where I_{max} denotes the maximum number of iterations of the algorithm for a task, and I_{ac} indicates the number of iterations for approximate computing convergence. Subsequently, the updated Lagrange multipliers as specified in (26), (27), and (28) can be effectively utilized to iteratively optimize the resource allocation problems (18), (20), (22), and (23).

Theorem 2. *The convergence of the ETORS algorithm is guaranteed, when the iteration step sizes, $\tau_{\lambda}^{[t]}$ and $\tau_{\mu}^{[t]}$, satisfy,*

$$\begin{aligned} &\tau_{\lambda}^{[t]} \to 0, \sum_{t=1}^{\infty} \tau_{\lambda}^{[t]} \to \infty, \sum_{t=1}^{\infty} (\tau_{\lambda}^{[t]})^2 < \infty \\ &\tau_{\mu}^{[t]} \to 0, \sum_{t=1}^{\infty} \tau_{\mu}^{[t]} \to \infty, \sum_{t=1}^{\infty} (\tau_{\mu}^{[t]})^2 < \infty \end{aligned} \tag{29}$$

Proof. To prove that the update iterations of a_{ik}, f_{ik} and $J(q_{ik})$ will converge to the optimal solution under the given conditions on the step sizes and Lagrange multipliers, we can apply the method of Lagrangian multipliers and the subgradient method for optimization. The Lagrangian function for the optimization problem is defined in (15). The update equations for $a_{ik}^{[t+1]}$, $f_{ik}^{[t+1]}$ and $J(q_{ik})^{[t+1]}$ can be calculated by (18), (20), (22), and (23), respectively. To establish convergence, we need to show that the $a_{ik}^{[t+1]}$, $f_{ik}^{[t+1]}$, and $J(q_{ik})$ are monotonically decreasing. From the update equations for $a_{ik}^{[t+1]}$ and $f_{ik}^{[t+1]}$, by observing the update equation for $\lambda_i^{[t+1]}$ in (28), we can see that $\lambda_i^{[t+1]}$ is decreasing. Thus, Ψ_{ik}^{l} and Ψ_{ik}^{c} in (18) are also decreasing. Similarity, $\eta_{ik}^{c}{}^{[t]}$ in (20) are also decreasing. As a result, $a_{ik}^{[t+1]}$ and $f_{ik}^{[t+1]}$ is monotonically decreasing. Similarly, we can prove that, since $J(q_{ik})^{[t+1]}$ is bounded below and monotonically decreasing, it converges to a limit, denoted as J^*. By following the KKT conditions [44]. Therefore, under the conditions of diminishing step sizes and the convergence of Lagrange multipliers, we can see that the update iterations of a_{ik}, f_{ik} and $J(q_{ik})$ will converge to the optimal solution. ■

Algorithm 1 Iterative ETORS algorithm for mobile user i

Initialize: $\mathcal{K}$, $\mathbf{prd}(k)$, I_{max}, d_{ik}, c_{ik}, δ_{ik}^{e}, δ_{ik}^{c}, λ_i, μ_{ik},$\tau_{\lambda}^{[t]}$,$\tau_{\mu}^{[t]}$, $\mathcal{F}$, $\mathcal{Q}$, infinitesimal number ω and iteration index $t \leftarrow 1$, $\forall i \in \mathcal{N}, k \in \mathcal{K}$
for $k = 1 : K$ **do**
  while $t \leq I_{max}$, and $|\mu_{ik}(t+1) - \mu_{ik}(t)| \leq \omega$ **do**
    Compute r_{ik}, T_{ik}^{l}, E_{ik}^{l} by (1)$\sim$ (3), respectively, CT_{ik}^{l} by (10), $T_{ik}^{c,tr}$, $E_{ik}^{c,tr}$, $T_{ik}^{c,exe}$ by (4)$\sim$(6), respectively, $CT_{ik}^{t} = T_{ik}^{c,tr} + RT_{ik}^{tr}$
    if $\mathbf{prd}(k) == \emptyset$ **then**
      $RT_{ik}^{l} = 0$, $RT_{ik}^{tr} = 0$, $RT_{ik}^{c} = T_{ik}^{c,tr}$
    else
      Compute RT_{ik}^{l} by (7), $RT_{ik}^{tr} = \max_{m \in \mathbf{prd}(k)} CT_{im}^{l}$
    end if
    Compute a_{ik} by (18)
    if $a_{ik} == 0$ **then**
      Compute f_{ik} by (20)
    else
      Compute ξ_{ik}^{c}, ξ_{ik}^{e}, η_{ik}^{c}, and η_{ik}^{e}
      if $RT_{ik}^{c} = CT_{ik}^{t}$ **then** Solve problem (22)
      else Solve problem (23) **end if**
    end if
    Update Lagrangian multipliers λ_i and μ_{ik} by (28)
    $t = t + 1$
  end while
end for

V. Performance Evaluation

In this section, we present the experimental performance and numerical results to demonstrate the efficiency of the ETORS algorithm.

A. Testbed Experiment

We introduce our 5G MEC testbed, which encompasses the architecture, setup, and experimental performances. Additionally, we record the quality of computation patterns in terms of CPU processing time and service latency.

1) Testbed Setup: In this part, we establish the Non-standalone (NSA) 5G RAN testbed to implement a realistic end-to-end 5G infrastructure at the MEC. The MEC testbed, along with the necessary software, is described in Fig. 3. Our proposed testbed primarily consists of the following main components:

- *Commercial Off-The-Shelf (COTS) User equipment (UE):* For end-users, we utilize a Samsung Galaxy S8 smartphone operating on Android 9.0 a representative COTS UE. This smartphone is capable of exchanging data via WiFi or cellular connections within the RAN network.
- *Remote servers:* We employ a Dell Laptop running Ubuntu 18.04 as the edge server in our setup. On the other hand, the remote cloud is deployed using a desktop PC equipped with an Intel Xeon E5-1650 processor, featuring 12 cores operating at 3.5 GHz, and 32 GB of RAM.
- *RAN Network:* As a component of the 5G RAN network, the 3rd Generation Partnership Project (3GPP) has put

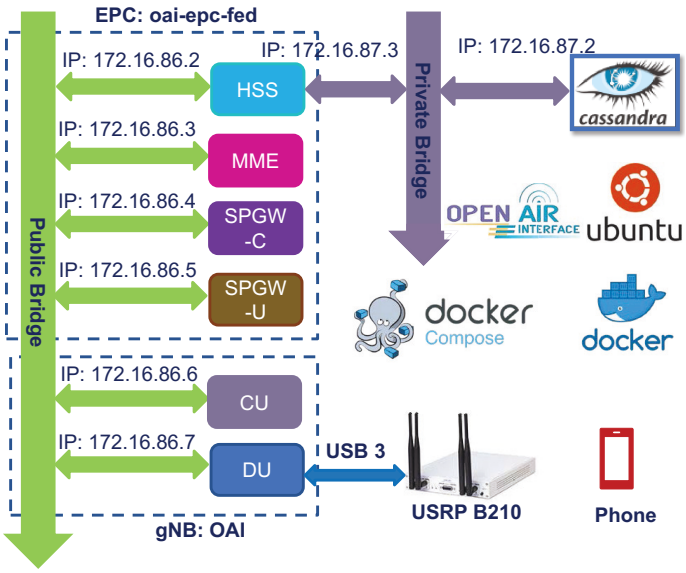


Fig. 3. 5G OAI-based testbed setup and configuration.

forward eight different functional split options, outlined in 3GPP TR 38.801, which specify the division of functionalities between the Distributed Unit (DU) and the Central Unit (CU) [45]. Therefore, in our testbed implementation, we assume that the functional split technique is integrated into the gNodeB (gNB).

Enabling a functional split in 5G RAN brings several significant benefits. These include traffic load balancing and minimizing latency and fronthaul costs. Furthermore, to establish our 5G NSA MEC testbed, we utilize OpenAir-Interface (OAI), an open-source platform developed by EURECOM. OAI is specifically designed for deploying the MAC/PHY layers at high levels in real-time 5G RAN systems.

As illustrated in Fig. 3, we have implemented a RAN, gNB, consisting of two containers: CU and DU. These containers are deployed using Docker, a widely-used containerization platform [46]. Additionally, in our MEC testbed, we employ the *openair-epc-fed* software to emulate the core network functionality. This software enables us to simulate the operations of the core network within the MEC environment.

The main components of *openair-epc-fed* are Mobility Management Entity (MME), Home Subscriber Server (HSS), and PDN Gateway (SPGW). As shown in Fig. 3, to establish the radio transmit links between the gNB and the UE, we use USRP B210, which is an SDR board that offers a frequency range from 70 MHz to 6 GHz.

2) Real-time Profiling: To perform the approximate computing, we select a real-time traffic lane detection (TLD) application. The main reason for choosing such an application in our case is that it has computing-intensive processing tasks, which can be performed at different levels of quality. Besides, applying the TLD application at MEC could be appropriate to exhibit the experimental results that can lead to improving the system model since the application composes several computation and communication tasks, including the capturing, extracting, and matching tasks. TLD first captures a real-time image of the road at different required frames. Then, the image is converted to grayscale, smooths, detects edges and selects a region of interest (ROI) to generate sloped lines on the traffic lanes. We perform the TLD application on two Dell Workstations each with two Xeon E6-1650 processors, 32 GB of RAM, and running on Ubuntu 18.04. In our case, we stream a one-minute video between the two workstations by using *ffmpeg*, a software tool used to process multimedia content to various resolutions. On the other end, we utilize a *ffplay* as a media player to receive and process the video stream.

Testbed Performance. In this experiment, we record the CPU utilization percentage by employing the *docker stats* command in Ubuntu. This command provides a real-time data stream that captures the CPU utilization of the running containers.

To simulate the experiment, we initiate the transmission of downlink UDP traffic from the SPGWU located in the core network to the UE. We conduct this transmission using different Physical Resource Block (PRB) values within two functional split settings: F1 and IF4.5 [45].

The percentage of the CPU usage has been measured for functional split Options F1, and IF4.5 in Fig. 4(a), and Fig. 4(b), respectively.

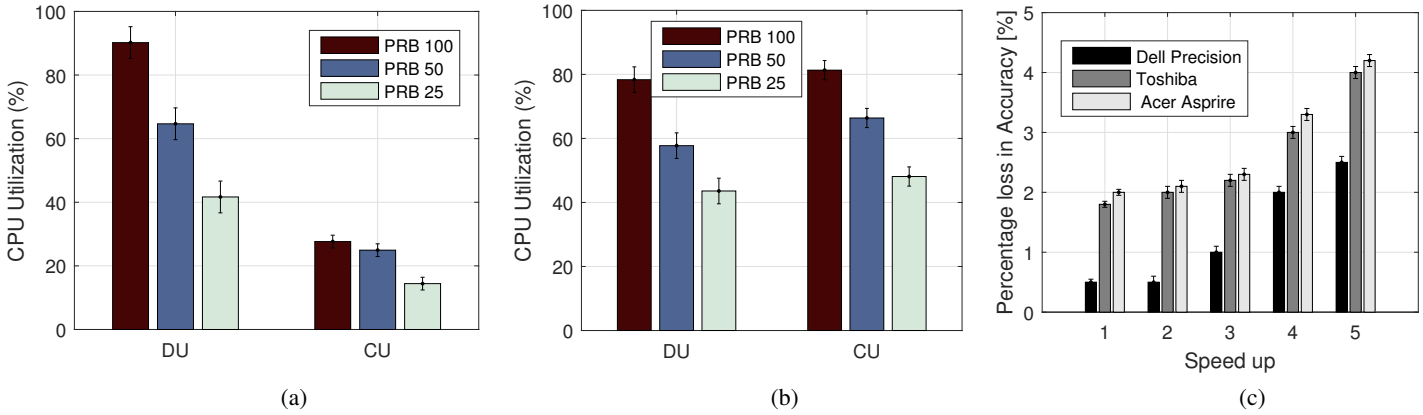
Besides, we motivate and study the performance of approximate computing via a widely recognized and extensively applied recognition algorithm known as Scale Invariant Feature Transform (SIFT) [47]. Fig. 4(c) shows the results of percentage loss in accuracy obtained when different levels of speed up are achieved by applying approximation transformation to the application. The percentage loss in accuracy of the output when applying approximation with respect to exact computation is calculated as $T = \frac{q-\hat{q}}{q} \times 100$, where q represents the accuracy of the output obtained through exact implementation of the application, while $\hat{q}$ represents the accuracy of the output obtained through approximate implementation.

The speed-up obtained from the task's approximate version is calculated by dividing the makespan (i.e., the time required to execute the workflow) of the task's approximate version by the makespan associated with its exact implementation.

It is observed that the speedup is 5 times when the percentage accuracy loss is 2.6% for the Dell Precision machine, equipped with a 2.66 GHz Intel i5 processor and 8 GB of RAM. A similar trend is observed for the other devices as well. Specifically, for the Toshiba device with a 2.13 GHz Intel i3 processor and 4 GB of RAM, and the Acer Aspire device with a 1.60 GHz Intel processor and 2 GB of RAM.

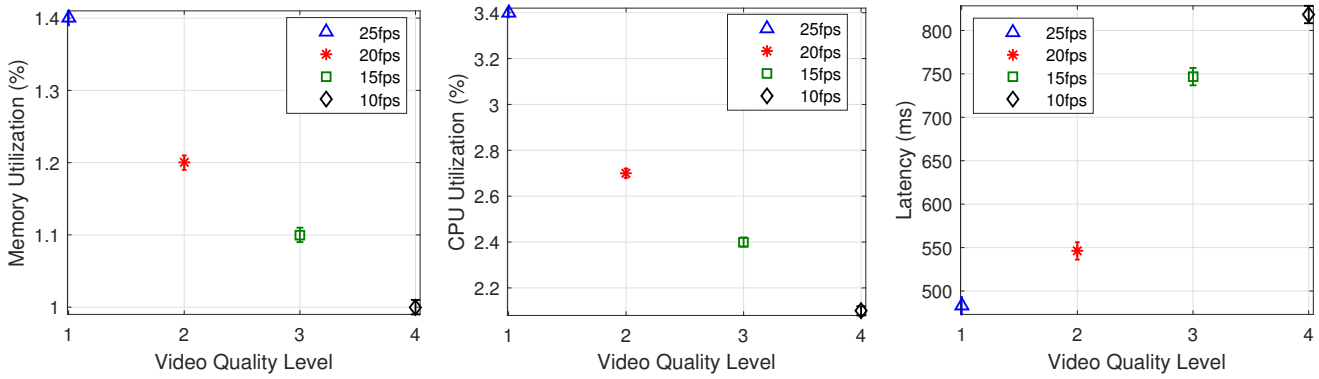
Computation Resources and Quality of Computation tradeoff.

To propose an efficient design of resource allocation algorithms in MEC, it is crucial to comprehend the computational resources available on MEC servers, such as CPU, memory, and processing time, in order to handle real-time task offloading requests.



(a) (b) (c)

Fig. 4. Fully containerized 5G OAI-based testbed experimental results for different configurations; (a) CPU utilization of functional split Option F1 for downlink traffic; (b) CPU utilization of functional split Option IF4.5 for downlink traffic, (c) Percentage loss in accuracy versus speed-ups achieved by applying approximate computing techniques on SIFT algorithm.



(a) (b) (c)

Fig. 5. (c) Average Memory Utilization for various quality of computation levels in video streaming; (b) Average CPU utilization for various quality of computation levels; and (c) Average processing time for various quality of computation levels.

In our case, we conducted multiple real-time experiments on the MEC testbed with various setups to establish the relationship between CPU, memory, processing time, and the quality of computation. As depicted in Figs. 5(a), 5(b), and 5(c), we collected data on CPU and memory usage, as well as network latency, for various quality levels after executing the TLD application between two nodes, namely the UE and the gNode B (gNB).

By utilizing the "top" tool with 1-second intervals, we measured the CPU and memory usage, along with network latency, of the TLD application, which includes the video processing layer, on the cloud server.

Remark: *Based on Figs. 5(a), 5(b), and 5(c), significant variations in memory, CPU, and latency can be observed corresponding to different levels of computation quality. Consequently, the function* $\varphi(q_{ik})$ *can be modeled as a linear equation,* $\varphi(q_{ik}) = \tau^c_{ik} q_{ik} + \tau^e_{ik}$*, where the constant parameters* τ^c_{ik} *and* τ^e_{ik} *can be estimated by fitting the data from Figs. 5(a), (b), and (c) for memory and CPU utilization, as well as latency time. The values used for estimation are provided in Table II.*

TABLE II
VALUES OF PARAMETERS τ^c_{ik} AND τ^e_{ik}.

$\varphi(q_{ik})$	τ^c_{ik}	τ^e_{ik}
Memory Utilization (%)	−0.13	1.5
CPU Utilization (%)	−0.42	3.7
Processing Cost (ms)	120.6	347

B. Numerical Simulations

We present simulation results to evaluate the performance of our proposed ETORS algorithm. The simulations are conducted using a MATLAB implementation with optimization solvers (e.g., MOSEK).

1) Setup: We consider a MEC system consisting of 100 m × 100 m cell with a BS in the center. The mobile devices, $N = 10$, are randomly located inside the cell. The channel gains are generated using a distance-dependent path-loss model given as $L[\text{dB}] = 140.7 + 36.7 \log_{10} \text{d}_{[\text{km}]}$, where d is the distance between the mobile device and the BS, and the log-normal shadowing variance is set to 8 dB. In most of the simulations, unless otherwise specified, we assume that the users' maximum transmit power is set to $p_{ik} = 20$ dBm, and the initial decision weights are $\delta^e_{ik} = \delta^c_{ik} = 0.5$. Furthermore, the network transmit bandwidth is chosen as $B_{ik} = B = 20$ MHz, while the additional noise power is set to $\sigma^2 = -100$ dBm. For the computing resources

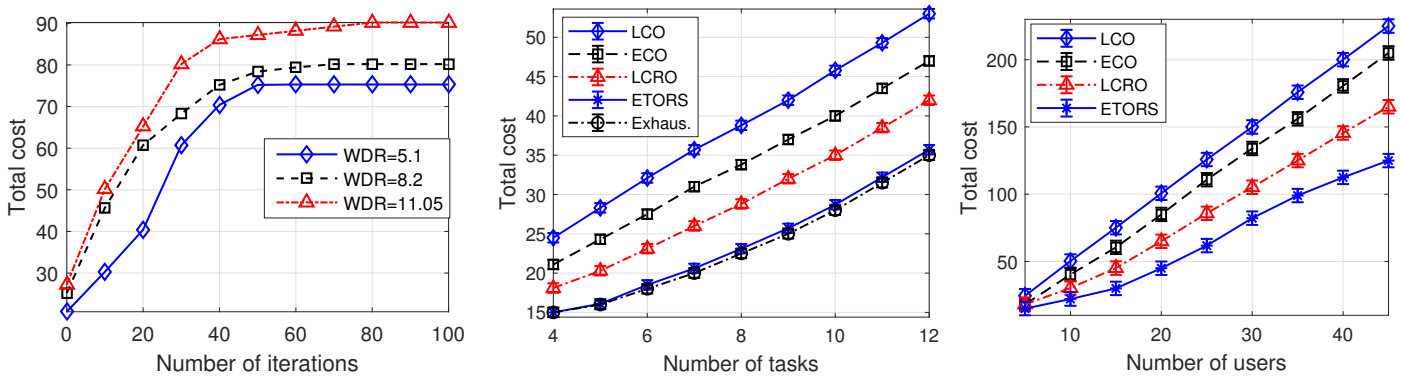


Fig. 6. Comparison of total cost against: (a) Different number of WDRs, evaluated with ten-user scenarios; (b) Different number of tasks, evaluated with two-user scenarios; and (c) Different number of users, evaluated with two-task scenarios.

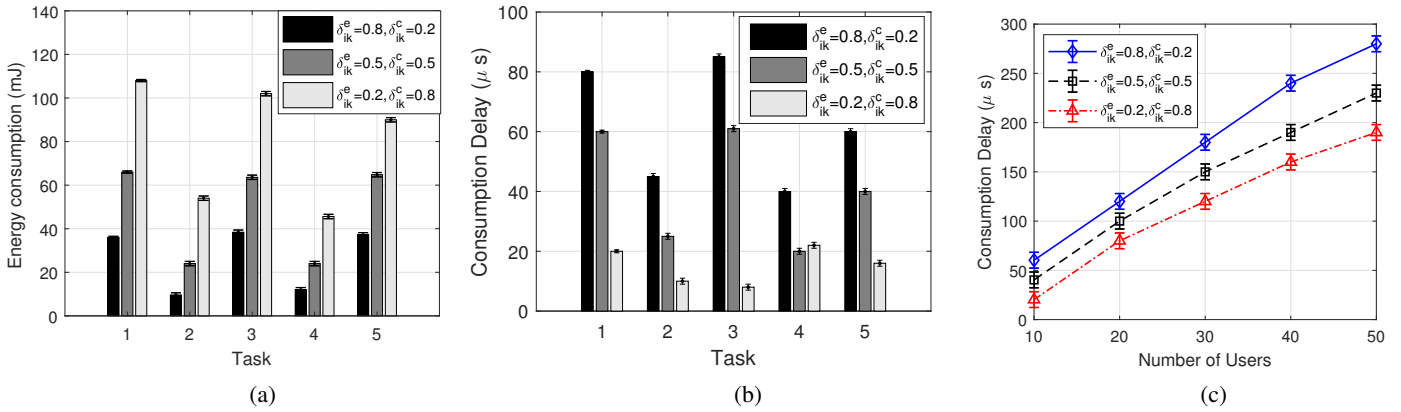


Fig. 7. Simulation result comparison for different values of δ^e_{ik} and δ^c_{ik} for (a) Energy consumption vs. number of tasks; (b) Computation delay vs. number of tasks; and (c) Computation delay vs. number of users.

configuration, the CPU clock frequency of the MEC server is set to $f_c = 20$ GHz, and the energy coefficient is set to $\epsilon_i = 10^{-11}$ based on empirical measurements presented in [48]. For the computation task, we consider the TLD application for security and surveillance

This particular application stands to significantly benefit from the collaborative capabilities offered by the integration of mobile devices and the MEC platform.

Unless otherwise specified, we adopt the default values for the settings: $d_{ik} = 420$ KB and $c_{ik} = 1000$ Megacycles, $\forall$i $\in \mathcal{N}$, k $\in \mathcal{K}$. We evaluate the system utility performance of the ETORS strategy by comparing it with the following four approaches.

i) *Exhaustive*: This approach utilizes a brute-force method to identify the optimal offloading scheduling solution by exhaustively searching through 2^{NK} possible decisions. Due to the significantly high computational complexity associated with this method, we limit the evaluation of its performance to a small-scale network.

ii) *Local Computing Only (LCO)*: Each user $i \in \mathcal{N}$ performs its computation task exclusively through local computing. This scheme corresponds to solving $\mathcal{P}1$ with $a_{ik} = 0$ for all $i \in N$ and $k \in \mathcal{K}$.

iii) *Edge Computing Only (ECO)*: Each user $i \in \mathcal{N}$ completes its computation task by fully offloading the computation input to the MEC server. This scheme corresponds to solving $\mathcal{P}1$ with $a_{ik} = 1$ for all $i \in \mathcal{N}$ and $k \in \mathcal{K}$.

iv) *Local-Cloud Random Offloading (LCRO)*: Each task is processed either at the local device or the edge cloud server with equal probability. The offloading decision is made independently by each user [49].

2) Complexity Analysis: In this section, we examine the convergence of the proposed ETORS algorithm concerning the number of iterations, computation tasks, and users. In terms of total cost, we evaluate our algorithm by numerical simulations conducted on a local workstation, *Intel Xeon E5-1650*. We define the Workload-input Data Ratio (WDR) to quantify the computational cost of user tasks, i.e., WDR $= \frac{c_{ik}}{d_{ik}}$. We consider three scenarios where the values of the three tasks are selected as $\{5.1, 8.2, 11.05\}$. For comparison purposes, we assume that the computation tasks can be executed both locally and remotely. After running the ETORS algorithm using these task values, we monitor the convergence pattern and record the total cost for different values of the WDR.

From Figure 6(a), several key points can be concluded: (i) The proposed ETORS algorithm converges after approximately $70 - 80$ iterations for all tasks.

TABLE III
AVERAGE RUN-TIME OVER MULTIPLE RUNS FOR DIFFERENT SCHEMES.

	LCO	**ECO**	**LCRO**	**ETORS**	**Exhaustive**
Runtimes [ms]	3.1	3.5	15.5	23	2,300

(ii) The computation task with a higher WDR value exhibits a lower number of iterations. This is due to the increased probability of offloading the task to the remote server as the complexity of the computation task rises, resulting in a higher WDR value. (iii) The Computation task with a higher WDR value tends to have higher total costs. This is due to the increased complexity associated with higher WDR values, leading to higher total costs. Consequently, it can be inferred that selecting an appropriate WDR setting for tasks can enhance complexity while reducing the total cost.

3) Algorithm Comparison: In this part of the section, we compare the system utility performance of our proposed ETORS strategy with two scenarios: task offloading baselines and several existing state-of-the-art heuristic joint task offloading schemes.

i) ETORS algorithm versus task offloading baseline schemes. To characterize the suboptimality of our proposed ETORS solution, we compare its performance with the best solution obtained by the *Exhaustive* method, and then with the three other described baselines. Since the *Exhaustive* method searches through all possible offloading scheduling decisions, its runtime becomes significantly long when dealing with a large number of variables.

Hence, we conduct the comparison in a small network setting where $N = 5$ users are uniformly distributed within the coverage area of a single BS.

Figure 6(b) presents a comparison of the performance of different schemes as the number of tasks K varies. In this figure, the parameter c_{ik} follows a controlled uniform distribution. It is evident that the proposed ETORS algorithm exhibits performance closely aligned with that of the Exhaustive method, while significantly outperforming the other baselines.

Additionally, Table III provides the average run-time per simulation iteration for different algorithms, executed on a Windows 7 desktop equipped with a 3.6 GHz CPU and 16 GB RAM.

It can be seen that the Exhaustive method takes a very long time, about $100\times$ longer than the ETORS algorithm for such a small network. The LCRO algorithm runs slightly faster than ETORS while ECO and LCO require the lowest runtimes. Figure 6(c) assesses the total cost performance as it varies with the number of users seeking to offload their tasks. It can be observed that ETORS consistently delivers the best performance, while the cost of all schemes noticeably increases as the number of users rises.

4) Impact of Parameters δ^e_{ik} and δ^c_{ik}: In Fig. 7 (a), and (b), we examine the impact of computation weights, δ^e_{ik} and δ^c_{ik}, on the computation energy and execution latency of five tasks with varying data sizes and CPU cycles.

The setting value of the data size and the CPU cycles of 5 tasks are given by $[91.05, 189.95, 96.12, 266.76, 130.05]$ KB and $[890, 985, 1090, 791, 1150]$ Megacycles, respectively. It can be noticed from Fig. 7(a) that, for a given task, the computation energy increases as δ^e_{ik} decreases. On the other hand, the trend of computing delay increases as δ^c_{ik} decreases. This is reasonable since a large δ^e_{ik} puts more emphasis on the energy consumption. As a consequence, all tasks are more likely to be processed remotely. Furthermore, as shown in Fig. 7(b), when δ^e_{ik} decreases, the delay time cost becomes significant, and all tasks become more likely to be processed locally. The comparison between latency and the number of UEs for different settings of δ^e_{ik} and δ^c_{ik} is illustrated in Fig. 7 (c). It can be observed that selecting appropriate values for these parameters positively affects the overall latency of the task. In the case of latency-sensitive applications, a larger value of δ^c_{ik} is advantageous for enhancing the QoS for the users.

ii) ETORS algorithm versus existing task offloading schemes.

- *Task Offloading and Resource Allocation based Greedy technique (TORSG):* Similar to the approach in [21], we consider that all computation tasks (up to the maximum number that can be admitted by the BS) are offloaded to the edge server. In each BS, the offloading tasks are greedily assigned to sub-bands with the highest channel gains until all tasks are admitted or all the sub-bands are occupied. We then apply joint resource allocation and across the BSs as proposed in Section IV-2.
- *Task offloading and resource allocation based Lyapunov optimization technique (TORSL)*: It is used in several existing work such as [19], [22]. According to Lyapunov optimization theory, we transform the objective function in Problem (14a) into the deterministic optimization problem in each slot. Then, similar to the approach presented in [19], the Lyapunov function can be defined as follows,

$$L_i(t) = \frac{1}{2}\sum\nolimits_{k\in\mathcal{K}} Q^2_{ik}(t), \forall i \in \mathcal{N}. \tag{30}$$

where $Q(t)$ denotes the queue function of the unaccomplished tasks and can be defined as, $Q_{ik}(t+1) = \max[Q_{ik}(t) - (\zeta_{ik} r_{ik}), 0] + d_{ik}$, where ζ_{ik} is represent slot length of each task k of user i.

The total cost system utility of the three competing schemes are plotted in Fig. 8 It can be observed that the total costs of all schemes increase with the number of computational tasks. This implies that applications with high workloads benefit more from offloading than those with low workloads. Furthermore, we observe that the performance gains of the TORSG and TORSL algorithms also follow a similar trend, indicating an increase in task workloads. However, our proposed algorithm outperforms the others because it is designed based on the DDM approach.

VI. CONCLUSION

We proposed a holistic strategy for the Energy- Latency-aware Task Offloading and Approximate Computing (ETORS) problem, which aims to optimize the trade-off between energy consumption and application completion time in a Mobile-Edge Computing (MEC) network. The underlying optimization problem is formulated as a Mixed-Integer Program-

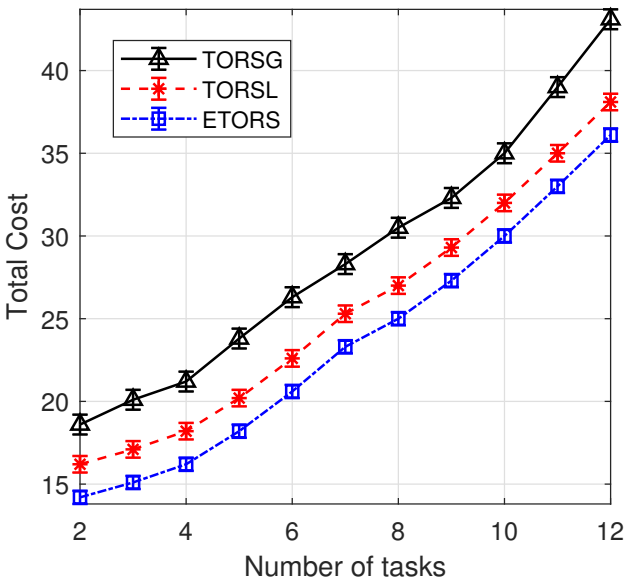


Fig. 8. Comparison of the total cost for various task offloading algorithms across different numbers of tasks, assessed using two-user scenarios.

ming (MIP) problem, which is NP-hard. Our approach decomposed the original problem into Task-Offloading Decision (TOD), the CPU Frequency Scaling (CFS), and the Quality of Computation Control (QoCC) problems; we then addressed these subproblems using convex optimization techniques.

Furthermore, we designed and implemented a programmable MEC testbed comprising a client-server and a MEC server. Finally, we implemented a real-time Traffic Detection (TLD) to study the benefits of computation offloading in an MEC server in terms of quality of computation and speed trade-off, system latency, and energy consumption. Simulation results show our algorithm approaches the optimal solution and significantly improves the total system offloading utility over traditional approaches.

REFERENCES

[1] A. Younis, T. X. Tran, and D. Pompili, "Energy-Latency-aware Task Offloading and Approximate Computing at the Mobile Edge," in *Proc. IEEE MASS*, pp. 1–9, 2019.
[2] Y. Siriwardhana, P. Porambage, M. Liyanage, and M. Ylianttila, "A survey on mobile augmented reality with 5G mobile edge computing: Architectures, applications, and technical aspects," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 2, pp. 1160–1192, 2021.
[3] F. Spinelli and V. Mancuso, "Toward enabled industrial verticals in 5G: A survey on MEC-based approaches to provisioning and flexibility," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 1, pp. 596–630, 2021.
[4] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili, "Collaborative mobile edge computing in 5G networks: New paradigms, scenarios, and challenges," *IEEE Commun. Mag.*, vol. 55, no. 4, pp. 54–61, 2017.
[5] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Netw. Appl.*, vol. 18, no. 1, pp. 129–140, 2013.
[6] P. Pandey and D. Pompili, "Exploiting the untapped potential of mobile distributed computing via approximation," *Pervasive and Mobile Computing*, vol. 38, pp. 381–395, 2017.
[7] Y. W. Bernier, "Latency compensating methods in client/server in-game protocol design and optimization," in *Technical report, Valve*, 2001.
[8] A. Ibrahim, M. Osta, M. Alameh, M. Saleh, H. Chible, and M. Valle, "Approximate computing methods for embedded machine learning," in *Proc ICECS*, pp. 845–848, 2018.
[9] H. J. Damsgaard, A. Ometov, and J. Nurmi, "Approximation opportunities in edge computing Hardware: A systematic literature review," *ACM Comput. Surveys*, vol. 55, no. 12, pp. 1–49, 2023.
[10] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, 2017.
[11] A. Younis, B. Qiu, and D. Pompili, "Latency-aware hybrid edge cloud framework for mobile augmented reality applications," in *Proc. IEEE SECON*, pp. 1–9, 2020.
[12] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Trans. Commun.*, vol. 65, no. 8, pp. 3571–3584, 2017.
[13] EURECOM, "OAI." Available: http://www.openairinterface.org/, 2018.
[14] Intel and Nokia Siemens Networks, "Increasing mobile operators' value proposition with edge computing," *Technical Brief*, 2013.
[15] "CloudLab: A Platform for Large-Scale Experimentation." https://www.cloudlab.us/.
[16] AT&T, "AT&T MEC." Available: https://www.business.att.com, 2021.
[17] X. Cao, F. Wang, J. Xu, R. Zhang, and S. Cui, "Joint computation and communication cooperation for energy-efficient mobile edge computing," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4188–4200, 2018.
[18] M.-H. Chen, M. Dong, and B. Liang, "Resource sharing of a computing access point for multi-user mobile cloud offloading with delay constraints," *IEEE Transactions on Mobile Computing*, vol. 17, no. 12, pp. 2868–2881, 2018.
[19] Y. Chen, N. Zhang, Y. Zhang, X. Chen, W. Wu, and X. S. Shen, "TOFFEE: Task offloading and frequency scaling for energy efficiency of mobile devices in mobile edge computing," *IEEE Trans. Cloud Comput., DOI: 10.1109/TCC.2019.2923692*, 2019.
[20] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *Proc. IEEE ISIT*, pp. 1451–1455, 2016.
[21] F. Wei, S. Chen, and W. Zou, "A greedy algorithm for task offloading in mobile edge computing system," *China Commun.*, vol. 15, no. 11, pp. 149–157, 2018.
[22] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3590–3605, 2016.
[23] M. Kamoun, W. Labidi, and M. Sarkiss, "Joint resource allocation and offloading strategies in cloud enabled cellular networks," in *Proc. IEEE ICC*, pp. 5529–5534, 2015.
[24] S. Barbarossa, S. Sardellitti, and P. Di Lorenzo, "Joint allocation of computation and communication resources in multiuser mobile cloud computing," in *Proc. IEEE Workshop SPAWC*, pp. 26–30, 2013.
[25] I. A. Elgendy, W.-Z. Zhang, Y. Zeng, H. He, Y.-C. Tian, and Y. Yang, "Efficient and secure multi-user multi-task computation offloading for mobile-edge computing in mobile IoT networks," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 4, pp. 2410–2422, 2020.
[26] A. Samy, I. A. Elgendy, H. Yu, W. Zhang, and H. Zhang, "Secure task offloading in blockchain-enabled mobile edge computing with deep reinforcement learning," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 4, pp. 4872–4887, 2022.
[27] G. Qu, H. Wu, R. Li, and P. Jiao, "DMRO: A deep meta reinforcement learning-based task offloading framework for edge-cloud computing," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 3, pp. 3448–3459, 2021.
[28] Y. Chiang, C.-H. Hsu, G.-H. Chen, and H.-Y. Wei, "Deep Q-learning-based dynamic network slicing and task offloading in edge network," *IEEE Trans. Netw. Service Manag.*, vol. 20, no. 1, pp. 369–384, 2023.
[29] O. Munoz, A. Pascual-Iserte, and J. Vidal, "Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading," *IEEE Trans. Veh. Technol.*, vol. 64, no. 10, pp. 4738–4755, 2015.
[30] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, 2016.
[31] M.-H. Chen, B. Liang, and M. Dong, "Joint offloading and resource allocation for computation and communication in mobile cloud with computing access point," in *Proc. IEEE INFOCOM*, pp. 1–9, 2017.
[32] H. Wu, Y. Sun, and K. Wolter, "Energy-efficient decision making for mobile cloud offloading," *IEEE Trans. Cloud Comput.*, vol. 8, no. 2, pp. 570–584, 2020.
[33] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Performance optimization in mobile-edge computing via deep reinforcement learning," in *Proc. IEEE VTC*, pp. 1–6, 2018.
[34] J. Yan, S. Bi, and Y. J. A. Zhang, "Offloading and resource allocation with general task graph in mobile edge computing: A deep reinforcement learning approach," *IEEE Trans. Wireless Commun.*, vol. 19, no. 8, pp. 5404–5419, 2020.

[35] T. D. Burd and R. W. Brodersen, "Processor design for portable systems," *VLSI Sign. Process. Syst.*, vol. 13, no. 2-3, pp. 203–221, 1996.
[36] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: Enabling interactive perception applications on mobile devices," in *Proc. ACM Mobile Syst., Appl. Serv.*, pp. 43–56, 2011.
[37] G. Bianchi, "Performance analysis of the IEEE 802.11 distributed coordination function," *IEEE J. Sel. Areas Commun.*, vol. 18, no. 3, pp. 535–547, 2000.
[38] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 4, pp. 974–983, 2015.
[39] X. Lyu, H. Tian, C. Sengul, and P. Zhang, "Multiuser joint task offloading and resource optimization in proximate clouds," *IEEE Trans. Veh. Technol.*, vol. 66, no. 4, pp. 3435–3447, 2017.
[40] M. Chiang, S. H. Low, A. R. Calderbank, and J. C. Doyle, "Layering as optimization decomposition: A mathematical theory of network architectures," *Proc. IEEE*, vol. 95, no. 1, pp. 255–312, 2007.
[41] E. D. Andersen and K. D. Andersen, "Presolving in linear programming," *Springer Mathematical Programming*, vol. 71, no. 2, pp. 221–245, 1995.
[42] S. Guo and Y. Yang, "A distributed optimal framework for mobile data gathering with concurrent data uploading in wireless sensor networks," in *Proc. IEEE INFOCOM*, pp. 1305–1313, 2012.
[43] D. P. Palomar and M. Chiang, "A tutorial on decomposition methods for network utility maximization," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 8, pp. 1439–1451, 2006.
[44] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.
[45] 3GPP TR 38.801, "Study of new radio access technology: Radio access architecture and interfaces," *Release 14*, 2017.
[46] "Docker." https://docs.docker.com/, 2022.
[47] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Springer Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, 2004.
[48] A. P. Miettinen and J. K. Nurminen, "Energy efficiency of mobile clients in cloud computing," *HotCloud*, vol. 10, pp. 4–4, 2010.
[49] W. Zhang, Y. Wen, and D. O. Wu, "Collaborative task execution in mobile cloud computing under a stochastic wireless channel," *IEEE Trans. Wireless Commun.*, vol. 14, no. 1, pp. 81–93, 2015.

Ayman Younis received his Ph.D. in Electrical and Computer Engineering (ECE) from Rutgers U., NJ, USA, in October 2022, under the guidance of Dr. Pompili. He had previously received the B.Eng. and M.Sc. degrees in Electrical Engineering from the U. of Basrah, Iraq, in 2008 and 2011, respectively. His research interests include wireless communications and mobile cloud computing, with an emphasis on software-defined radio testbeds, cloud radio access networks, mobile-edge computing, and next-generation radio access networks. He received the Best Paper Award at the IEEE/IFIP Wireless On-demand Network Systems and Services Conference in 2021.

Sumit Maheshwari is a Software Engineer at Microsoft. Sumit also previously held various positions at Affirmed Networks, Nokia Bell Labs, AT&T, and Samsung. He completed his PhD from WINLAB, Electrical and Computer Engineering at Rutgers University. He has a Masters in Wireless Communications from IIT Kharagpur, India, and a Bachelors in Electronics and Communication Engineering from Dr. MGR University, India. His research interests span across computer networks, AI and wireless communications with a specific focus on edge clouds and virtual networks.

Dario Pompili is a professor with the Dept. of ECE at Rutgers University. Since joining Rutgers in 2007, he has been the director of the CPS Lab, which focuses on mobile edge computing, wireless communications and networking, acoustic communications, and sensor networks. He received his PhD in ECE from the Georgia Institute of Technology in 2007. He had previously received his 'Laurea' (combined BS and MS) and Doctorate degrees in Telecommunications and System Engineering from the U. of Rome "La Sapienza," Italy, in 2001 and 2004, respectively. He has received a number of awards in his career including the NSF CAREER'11, ONR Young Investigator Program'12, and DARPA Young Faculty'12 awards. In 2015, he was nominated Rutgers-New Brunswick Chancellor's Scholar. He served on many international conference committees, taking on various leading roles. He published about 200 refereed scholar publications, some of which received best paper awards: with about 16K citations, Dr. Pompili has an h-index of 50 and an i10-index of 138 (Google Scholar, Jan'24). He is a Fellow of the IEEE Communications Society (2021) and a Distinguished Member of the ACM (2019). He serves as Associate Editor for IEEE Transactions on Mobile Computing (TMC) and Area Editor of IEEE INFOCOM.