# A Multi-Layered Distributed Computing Framework for Enhanced Edge Computing

Ke Ma, *Student Member, IEEE,* Junfei Xie, *Senior Member, IEEE*

***Abstract*—The rise of the Internet of Things and edge computing has shifted computing resources closer to end-users, benefiting numerous delay-sensitive, computation-intensive applications. To speed up computation, distributed computing is a promising technique that allows parallel execution of tasks across multiple compute nodes. However, current research predominantly revolves around the master-worker paradigm, limiting resource sharing within one-hop neighborhoods. This limitation can render distributed computing ineffective in scenarios with limited nearby resources or constrained connectivity. In this paper, we address this limitation by introducing a new distributed computing framework that extends resource sharing beyond one-hop neighborhoods through exploring multi-hop offloading and layered network structures. Our framework involves transforming the network graph into a shortest-path tree and formulating a joint optimization problem based on the layered tree structure for task allocation and scheduling. To solve this problem, we propose two exact methods that find optimal solutions and three heuristic strategies to improve efficiency and scalability. The performances of these methods are analyzed and evaluated through theoretical analyses and comprehensive simulation studies. The results demonstrate their promising performances over the traditional distributed computing and computation offloading strategies.**

***Index Terms*—Edge computing, distributed computing, multi-hop offloading.**

## I. Introduction

THE proliferation of the Internet of Things (IoT) devices has enabled a multitude of delay-sensitive yet computation-intensive applications, such as face recognition, environment monitoring, and augmented/virtual reality [1], [2]. The surge of these applications drives the migration of computing resources from the remote cloud to the network edge closer to end-users [3]. Various architectures have been proposed to enable edge computing [4], such as cloudlet [5], mobile edge computing (MEC) [6], fog computing [7], networked airborne computing (NAC) [8]–[10], and vehicle edge computing (VEC) [11]. These architectures typically consist of four key elements: 1) edge devices or end users (e.g., IoT sensors, smartphones) that generate data for processing; 2) edge servers (e.g., cloudlets, MEC servers) that provide localized computing resources; 3) network infrastructure (e.g., base stations, Wi-Fi routers) that enables communication and data transmission, and 4) an optional cloud backend for handling heavy computational tasks or global coordination. The edge servers are often deployed at base stations, cell aggregation sites (e.g., malls, airports, and stadiums) or at the edge of the core network (e.g., access points and gateways) [6]. In VEC, edge servers are commonly deployed at roadside units (RSUs) or directly on vehicles. In NAC, edge servers are deployed on the drones. While edge computing offers compelling benefits, such as low latency, cost effectiveness, and improved data control and security, it also presents notable challenges. The distributed nature of edge servers, along with their inherent constraints in computing power, memory capacity, and available bandwidth when compared to the cloud, pose significant challenges to achieving high-performance edge computing [12] [13].

To speed up computation at the edge, distributed computing can be employed. Existing distributed computing strategies typically adopt a *master-worker* paradigm [14] [3], where a single master node partitions and distributes the task to multiple worker nodes that are directly connected to it. Although the master-worker paradigm is simple to implement, it restricts resource sharing within one-hop neighborhoods. However, in edge networks, situations may arise where nearby servers have very limited or no residual resources available or where the master can connect to only a few neighboring servers, rendering such master-worker based distributed computing ineffective. Such situations are particularly common in loosely deployed edge environments, regions with high user demand, and networks with unstable or disrupted connectivity. For instance, in high-demand hotspots with dense user populations (e.g., transportation hubs and concerts), nearby edge resources may be insufficient to meet computational needs. In remote industrial IoT deployments (e.g., oil fields, power grids, and mining sites), access to nearby edge servers is often limited or unreliable due to sparse infrastructure. In disaster scenarios, links to powerful nodes may be severed due to the destruction of communication infrastructure. Furthermore, in vehicular or drone-assisted edge computing networks, many devices may lack direct connectivity to compute-capable nodes due to mobility or communication range constraints.

In this paper, we overcome these challenges by exploring resources at distant servers located multiple hops away. While a similar idea has been explored in the MEC [15]–[21] [22], [23] and Internet of Vehicle [23]–[26] [27] domains, where computation offloading is proposed to address users' and vehicles' computing demands, most existing studies focus on offloading tasks to a single edge server located one or multiple hops away, with intermediate servers serving solely as relays. A few recent works [28]–[31] have considered offloading tasks to multiple servers, but they overlook the task scheduling issue addressed in this work and offer only heuristic solutions. To the best of our knowledge, this is the first systematic investi-

Manuscript received October 13, 2024.

Ke Ma is with the Department of Electrical and Computer Engineering, University of California, San Diego, 92037, USA, and also with the Department of Electrical and Computer Engineering, San Diego State University, 92115, USA. (e-mail:kem006@ucsd.edu)

Junfei Xie is with the Department of Electrical and Computer Engineering, San Diego State University, 92115, USA. (e-mail:jxie4@sdsu.edu)

gation into computation offloading to multiple servers across multiple hops and into the benefits of layered structures for improving distributed computing performance. Additionally, we present efficient exact solutions for optimal task allocation and scheduling, which are applicable not only to diverse edge computing architectures but also to broader classes of networked computing systems involving collaborative task execution among multiple computing servers.

The main contributions are summarized as follows:

- *A new multi-layered distributed computing framework:* The proposed framework explores layered network structures to fully utilize the capacity of the entire edge computing network for enhanced system performance. By transforming the network graph into a shortest-path tree, the resulting layered structure simplifies the analysis of task allocation and scheduling. Building upon this tree structure, we formulate a mixed-integer programming (MIP) problem to jointly optimize task allocation and scheduling. To the best of our knowledge, this is the first study to employ layered tree structures for investigating multi-hop distributed computing. Notably, the proposed framework is not limited to edge computing networks but is broadly applicable to any networked computing system.
- *Two exact methods that find optimal solutions:* To solve the optimization problem, we first develop a centralized method that guarantees optimality. Leveraging the layered network structure, which captures scheduling dependencies, we then introduce a parallel enhancement that preserves optimality while significantly reducing execution cost by harnessing the inherent parallelism in task distribution across different subtrees. We also present an offline-online computation scheme that allows both methods to execute in real-time. How these methods generalize existing solutions is also discussed.
- *Three heuristic methods for improving efficiency and scalability:* While the two exact methods offer optimality, their execution time grows rapidly as the network expands. To mitigate this challenge, we introduce two worker selection methods, as well as a genetic algorithm for efficiently finding near-optimal solutions.
- *Comprehensive simulation studies:* To evaluate the performance of the proposed approaches, we conduct extensive comparison studies. Our results demonstrate that enabling resource sharing within the entire network leads to better solutions compared to those found by the traditional distributed computing and computation offloading strategies. Additionally, simulations are conducted to assess the time efficiency of the proposed approaches and the impact of their key parameters.

It should be noted that the multi-layered distributed computing framework was initially introduced in a short conference version [3], which presented a different heuristic method for solving the MIP problem. This paper provides a more comprehensive and systematic investigation, featuring a set of new and rigorously analyzed methods.

The rest of the paper is organized as follows. Sec. II discusses related works. Sec. III describes the system model and the problem to be solved. Sec. IV introduces the proposed multi-layered distributed computing framework. The two exact methods and the three heuristic methods are introduced in Sec. V and Sec. VI, respectively. Sec. VII presents the results of simulation studies. Finally, Sec. VIII concludes the paper and discusses the future works.

## II. Related Works

This section reviews existing studies related to this work.

### A. Distributed Computing

In the field of distributed computing, the master-worker paradigm has been widely used to implement parallel applications [9], [32]–[34]. In this paradigm, multiple workers share the workload assigned by the master and communicate directly with the master. To determine the optimal task allocation, various distributed computing strategies have been proposed. For instance, traditional server-based distributed computing systems often divide the workload among workers equally or proportionally according to workers' computing power [32]. In heterogeneous systems or those with mobile compute nodes, stragglers, which are nodes with long response times, are common and can significantly degrade system performance. To mitigate the impact of stragglers, coded distributed computing techniques [9], [32], [33] have recently become increasingly popular. These techniques leverage coding theory to introduce computational redundancies, thereby enhancing system robustness against stragglers. While effective, they increase the computational load and, consequently, the overall energy consumption of the system. Additionally, they require greater storage resources to handle the added redundancies.

Another popular paradigm is the hierarchical master-worker paradigm [35], which involves a supervisor process managing multiple sets of processes, each consisting of a master process and multiple worker processes. It offers several advantages over the traditional master-worker paradigm, including improved scalability and fault tolerance [35]. However, it requires careful coordination and synchronization among the supervisor, sub-masters, and workers, which can complicate system maintenance and debugging. Differing from these paradigms, we investigate a multi-layer master-worker paradigm that is composed of a single master and multiple workers operating at different layers. By making the master directly access resources across all workers, this centralized structure simplifies the control logic and facilitates more efficient optimization of resource allocation.

### B. Computation Offloading

In the computation offloading domain, most existing studies consider a single-hop single-server offloading paradigm, where tasks are offloaded from users to a single edge server within their communication range [15]–[21].

The tasks can be offloaded as a whole or partially, known as *binary offloading* and *partial offloading*, respectively. Under this paradigm, many algorithms have been designed to make

the optimal offloading decisions. For instance, studies in [15], [16] examine the scenario where tasks are offloaded from a single user to a single nearby server. [17] extends this analysis by taking server mobility and task dependency into account. There have also been studies that explore tasks from multiple users, optimizing not only the offloading decisions (whether to offload a task or determining the offloading ratio) but also the allocation of resources to each user. For instance, [21] considers the allocation of computing resources, while [18] addresses the allocation of both computing and transmission power resources. The allocation of communication resources, including time slots under the Time Division Multiple Access protocol and sub-channels under the Orthogonal Frequency-Division Multiple Access (OFDMA) protocol, is considered in [19]. These problems are typically solved using numerical approaches [19], [20]. Reinforcement learning has also emerged as a promising tool for computation offloading [18], [21]. Although this single-hop single-server offloading paradigm is easy to implement, it limits the amount of resources users can access to a single nearby server.

In practical scenarios, it is possible that there are no (powerful) edge servers nearby for the end users. To address this limitation, researchers have started to explore multi-hop offloading, enabling the offloading of tasks from users to remote servers multiple hops away. Along this direction, existing studies mostly consider offloading tasks to a single server, as seen in [23]–[26], [36], [37]. Additionally, the three-tier network topology comprising end users, edge servers and cloud servers [38]–[41] has garnered considerable interest. In their approach, tasks are offloaded either to a nearby edge server one hop away or to a cloud server two hops away, with edge servers acting as relays. In contrast, we consider all servers reachable by users and aim to facilitate collaborative computing among them.

There are also several works that investigate partitioning tasks into multiple parts and offloading these parts to multiple servers, which are most relevant to our study [28]–[31]. For instance, [28] investigates the joint routing and multi-part offloading for both data and result. It employs a flow model to capture data/result traffic and introduces a distributed algorithm that finds solutions in polynomial time. [29] formulates the multi-hop offloading problem as a potential game. By dividing tasks into subtasks of equal size, each device independently decides the number of subtasks to forward or compute based on its economic utility. The study in [31] addresses the distribution of a set of tasks, partitioned from a complex application, to multiple cooperative servers that may be multiple hops away. This problem is formulated as a task assignment problem and solved by an iterative algorithm. Another relevant work is presented in [30], which considers a joint user association, channel allocation, and task offloading problem. It solves this problem by combining the genetic algorithm and deep deterministic policy gradient algorithm. Although a similar offloading paradigm is considered in these studies, they overlook the task scheduling issue addressed in this work and provide only heuristic solutions.

Distinct from previous research, we delve into the essential benefits of layered network structures while investigating how network properties like topology and server resources affect system performance. We also address the task scheduling problem that arises when transmissions of subtasks share channels or relays, which has been overlooked by existing works. Moreover, we propose both exact and heuristic methods to solve the problem, and introduce an offline-online computation scheme to enable real-time implementation and enhance their ability to adapt to dynamic and mobile networks.

## III. System Model and Problem Description

In this section, we first present the system model and then describe the problem to be solved.

### A. System Model

Consider an edge computing system (see Fig. 1 for an illustration) formed by $N+1$ edge servers, each with its own unique set of computing and communication capabilities. In this study, we focus on the common scenario where servers are statically deployed at locations such as base stations, cell aggregation sites, access points, gateways, or RSUs. The servers can share resources with their neighbors either through cables in wired networks or wirelessly when they are within communication range. Additionally, a server can communicate with its one-hop neighbors simultaneously using techniques like Orthogonal Frequency Division Multiplexing [42]. For simplicity, interference among the servers is not considered in this study. This network scenario arises in many real-world deployments, such as in smart city applications where edge servers are deployed at various locations (e.g., at base stations, airports and buildings) to process data from IoT devices [43]; in drone-assisted MEC systems where hovering drones are deployed as edge servers to process computation tasks sent from ground users [44]; and in VEC networks, where roadside units act as edge servers to handle tasks offloaded from vehicles [45].

Suppose one of the servers, referred to as *master*, has a computation-intensive task that is arbitrarily decomposable for parallel processing, such as matrix operations, image processing, and Monte Carlo simulations. The primary focus of this work is to investigate how such a task can be efficiently processed by a network of servers. Where and how the master receives this task is not within our interest. It could be generated locally by the server itself, requested by a nearby user, or assigned by a central orchestrator (e.g., a software-defined networking controller [46] or a cloud service [47]). To complete the task in a timely and energy-efficient manner, the task is decomposed into subtasks and distributed to the other servers, referred to as *workers*. The master (highlighted with red) can transmit subtasks simultaneously to their neighboring servers. However, for workers farther away, multi-hop offloading is required, which means that each server in the network can act as a worker, a relay, or both. When a subtask arrives at a relay, it is added to a queue and processed in a first-in-first-out order. A worker will not start executing the assigned subtask until it receives the complete subtask package. When a server acts as both a worker and a relay, it can perform the relay process and execute the assigned task simultaneously.

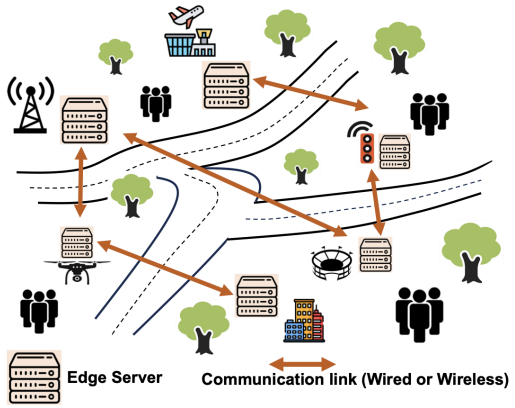



Figure 1. Network scenario.

In this preliminary study, we adopt several common assumptions made in existing studies [48], [49] to simplify our analysis. In particular, we assume that the network is stable with no package losses or retransmissions. Additionally, we assume that the computation result is relatively small, and hence the delay incurred in transmitting the result from workers back to the master is negligible. Under these assumptions, we model the network as follows.

*1) Network Model:* The network is modeled as a directed graph $\mathcal{G} = \{\mathcal{N}, \mathcal{E}\}$, where $\mathcal{N} = \{i | 0 \leq i \leq N\}$ is the set of edge servers and $\mathcal{E} = \{(i,j) | i,j \in \mathcal{N}, i \neq j\}$ is the set of server-to-server communication links that connect servers that can communicate directly.

*2) Computing model:* Let $f_i$ denote the computing capacity of server $i$, i.e., CPU-cycle frequency (GHz). Given a task of size $y$, let $b$ denote the total number of CPU cycles required to process one task size unit. The time required for server $i$ to process this task can then be expressed by [50]:

$$T_i^{comp} = \frac{yb}{f_i} \tag{1}$$

*3) Communication Model:* Let $R_{i,j}$ denote the data transmission rate from server $i$ to server $j$, which characterizes the communication property between two servers. For instance, for wired connections, $R_{i,j}$ can be approximated using Shannon's Theory [10] as $R_{i,j} = B_{i,j} log_2(1 + \frac{s_{i,j}}{n_{i,j}})$, where $B_{i,j}$ is the channel bandwidth, and $s_{i,j}$ and $n_{i,j}$ represent the signal power and noise power, respectively. For wireless connections, a feasible model [51] to approximate the transmission rate is $R_{i,j} = B_{i,j} log_{10}(1 + \frac{P_{i,j}\phi_{i,j}h^2}{n_{i,j}})$, where $P_{i,j}$ denotes the transmission power, $\phi_{i,j}$ is the path loss factor, and $h$ represents the distance between the two servers. In this work, we assume that $R_{i,j}$ is known to simplify the analysis, while its determination depends on the specific communication technologies used and is considered outside the scope of this study.

*4) Energy Consumption Model:* The energy consumed for executing a task mainly constitutes two components: energy consumed for computing and energy consumed for communication. The energy consumed for server $i$ to compute a task of size $y$ is given by [52]–[55]:

$$E_i^{comp} = \gamma_i y b (f_i)^2 \tag{2}$$

where $\gamma_i$ is the effective switched capacitance that depends on the chip architecture of server $i$. The energy consumed for server $i$ to transmit a task of size $y$ to server $j$ is given by [52]–[55]:

$$E_{i,j}^{comm} = \frac{e_i y}{R_{i,j}} \tag{3}$$

where $e_i$ represents the transmission power of server $i$.

### B. Problem Description and Analysis

Without loss of generality, suppose the master receives a task of size $Y \in \mathrm{R}^+$ to complete. Given the computing and communication characteristics of the entire network, i.e., $\mathcal{G}$, $\{f_i, R_{i,j}, e_i, \gamma_i\}, \forall i,j \in \mathcal{N}$, which are assumed to be known and can be collected, for example, by a central controller such as a software defined networking controller [46] or through message exchanges among the servers, we aim to jointly minimize the task completion time and energy consumption by partitioning the task into small subtasks and distributing them to other servers in the network.

Finding the optimal solution to this problem is nontrivial and challenging since it requires making decisions on several aspects, including identifying which servers the master should assign subtasks to, determining the amount of workload to be assigned to each worker, and selecting the transmission route for sending the subtask. Moreover, the order in which the subtasks should be sent by the master is also a crucial decision to make.

## IV. Multi-Layered Distributed Computing Framework

In this section, we present a multi-layered distributed computing framework to solve the problem described in the previous section. The framework first transforms the network graph into a shortest-path tree and then exploits this structure to derive the optimal task allocation and scheduling solutions.

### A. Transforming Graph into a Shortest-Path Tree

Our framework first identifies all compute nodes that are reachable from the master, along with the shortest paths to these nodes. Each path represents the most communication-efficient route that minimizes the transmission time for a single bit of data. Notably, if multiple shortest paths exist for a given node, one is selected at random. Conversely, if no path exists, the node is deemed unreachable and excluded from the graph. To determine these paths, we define the weight of each edge $(i,j) \in \mathcal{E}$ as the inverse of the associated data transmission rate, i.e., $1/R_{i,j}$, and apply Dijkstra's algorithm [56]. By unifying the shortest paths, we construct a shortest-path tree, denoted as $\mathcal{T}$, which is rooted at the master, and each node in the tree has a unique path from the master that is the most communication-efficient. This tree structure offers an intuitive and systematic representation of the distributed computing system, capturing task distribution paths and node hierarchy. This not only simplifies problem formulation but also facilitates the analysis of task allocation, functional role (relay or worker) of each node in task execution, and scheduling dependencies as explained below.

To facilitate subsequent analysis, we re-label the nodes in the tree $\mathcal{T}$ level-by-level from the root downward, and from left to right within each level (see Fig. 2). Consequently, nodes in lower levels have larger indices. Let $\mathcal{I}_l$ denote the set of indices of nodes in level $l \in \{0, 1, \ldots, H\}$, where $H$ is the height of the tree. Then, $\cup_{l=0}^{H} \mathcal{I}_l = \mathcal{N}$.

Notably, the master (root) can transmit subtasks to its one-hop neighbors, i.e., nodes in Level 1, simultaneously. However, if any one-hop neighbor has children, the subtasks assigned to them, including the one-hop neighbor, have to be transmitted one by one. This is because they share the same channel between the master and the one-hop neighbor, and the data arriving at the one-hop neighbor is processed in a first-in-first-out manner.

This sequential transmission makes the offloading order of the subtasks critical to the overall task completion time. For example, consider the scenario illustrated in Fig. 2, where each subtask (represented by a file icon) is labeled with the index of the server to which it is assigned. Focusing on nodes 1 and 3 in the left subtree, the order in which their subtasks are offloaded from the master (node 0) will lead to different overall task completion times. Specifically, if the subtask for node 1 is offloaded first, then node 1 only needs to wait for its subtask to be transmitted from node 0 before it can start execution, resulting in a waiting time denoted as $t_1^{0\to1}$. However, node 3 must wait for the subtask for node 1 to be transmitted from node 0 to node 1, and then for its own subtask to be transmitted from node 0 to node 1 and subsequently from node 1 to node 3, resulting in a total waiting time of $t_1^{0\to1} + t_3^{0\to1} + t_3^{1\to3}$. On the other hand, if the subtask for node 3 is offloaded first, node 1 must wait for the subtask for node 3 to be transmitted from node 0 to node 1, and then for its own subtask to be transmitted, resulting in a waiting time of $t_1^{0\to1} + t_3^{0\to1}$. In this case, node 3 only needs to wait for its subtask to be transmitted from node 0 to node 1 and then from node 1 to node 3, resulting in a waiting time of $t_3^{0\to1} + t_3^{1\to3}$.

Therefore, different offloading orders lead to different overall task completion times due to the variations in waiting times. Notably, the offloading order for the subtask computed locally at the master does not influence computation performance, as the master can execute its subtask immediately. Additionally, the offloading orders of servers belonging to one subtree of the master do not impact the waiting times of servers in a different subtree, since subtasks for servers in different subtrees can be offloaded in parallel.

Based on above analyses, we next formulate a joint task allocation and scheduling problem as a mixed integer programming (MIP) model.

### B. Mixed Integer Programming Model

*1) Decision Variables:* To specify the computation workload allocated to each node $i \in \mathcal{N}$, we introduce decision variables $\boldsymbol{y} = \{y_0, y_1, \ldots, y_N\}$, where $y_i \in [0, Y]$ represents the size of the subtask assigned to node $i$. If $y_i = 0$, it implies that node $i$ is not assigned any workload. Note that the master may choose to execute (part of) the task locally, in which case $y_0$ would be nonzero, i.e., $y_0 > 0$.

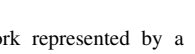



Figure 2. An example network represented by a layered tree structure. Servers' indices are highlighted in red.

To describe the offloading order for subtasks transmitted from the master to the other nodes, we introduce decision variables $\boldsymbol{o} = \{o_1, o_2, \ldots, o_N\}$, where $o_i \in \mathcal{N} \setminus \{0\}, \forall i \in \mathcal{N} \setminus \{0\}$ and $o_i \neq o_j, \forall i, j \in \mathcal{N} \setminus \{0\}, i \neq j$. When $o_i > o_j$, node $i$ has a higher priority than node $j$ to receive its subtask, where $i, j \in \mathcal{N} \setminus \{0\}$ and $i \neq j$.

*2) Objective Function:* We aim to achieve two objectives simultaneously: minimize the time spent and minimize the energy consumed by each node for executing the task. By employing a weighted sum method, we define the objective function as follows:

$$\begin{aligned} \mathcal{J}(\boldsymbol{y}, \boldsymbol{o}) &= \max_{i \in \mathcal{N}} w_1 T_i^{total} + w_2 E_i^{total} \\ &= \max_{i \in \mathcal{N}} J_i(\boldsymbol{y}, \boldsymbol{o}) \end{aligned} \tag{4}$$

where $w_1, w_2 \geq 0$ are the weights, representing the relative importance of the two objectives. $T_i^{total}$ is the total time required for node $i$ to receive its subtask from the master and complete the assigned subtask. Note that the time required for completing the whole task is $\max T_i^{total}$, $i \in \mathcal{N}$. $E_i^{total}$ is the total energy consumed by node $i$ during task execution. $J_i$ is introduced to denote the cost associated with node $i$. In the following sections, parentheses or subscripts may be omitted for simplicity when there is no confusion.

Next, we derive the formulas for $T_i^{total}$ and $E_i^{total}$.

*3) Time Consumption:* The task completion time for node $i$, $T_i^{total}$, is comprised of three components: 1) time taken to transmit subtask of size $y_i$ from the master to node $i$, denoted as $T_i^{tran}$; 2) time spent waiting in the queues of relays along the path to node $i$ if any, denoted as $T_i^{wait}$; and 3) time to execute the subtask, i.e., $T_i^{comp}$. It is noted that the waiting time $T_i^{wait}$ is impacted by the task sizes assigned to other nodes and the offloading order, which complicates the optimization problem considered in this study.

To obtain the transmission time $T_i^{tran}$, we introduce the notation $\boldsymbol{p}_i$ to denote the sequence of nodes that lie on the path from the master to node $i$, and the notation $p_{ik}$ to denote the $k$-th node in the sequence, where $1 \leq k \leq |\boldsymbol{p}_i|$, $p_{i1} = 0$ and $p_{i|\boldsymbol{p}_i|} = i$. $|\cdot|$ finds the cardinality of a set. $T_i^{tran}$ can then be expressed by:

$$T_i^{tran} = \begin{cases} 0, & \text{if } i = 0 \\ \sum_{k=1}^{|\boldsymbol{p}_i|-1} \frac{y_i}{R_{p_{ik}, p_{i(k+1)}}}, & \text{else} \end{cases} \tag{5}$$

Let's now consider the waiting time $T_i^{wait}$. Let $\mathcal{A}_t$ denote the full set of nodes in the $t$-th subtree of the master, where $t \in \mathcal{I}_1$, and $\cup_{t \in \mathcal{I}_1} \mathcal{A}_t = \mathcal{N} \setminus \{0\}$. Additionally, define $\mathcal{B}_i = \{j | o_j > o_i, i, j \in \mathcal{A}_t, i \neq j\}$ as the set of nodes whose subtasks will be transmitted before node $i$. Note that if nodes $i$ and $j$ belong to different subtrees, i.e., $i \in \mathcal{A}_t$ while $j \notin \mathcal{A}_t$, the subtask for node $j$ is transmitted using a different channel that is orthogonal to the one used for node $i$, and hence node $i$ does not need to wait for node $j$'s subtask to

be transmitted even if $o_j > o_i$. Based on these definitions, we can then express the waiting time as follows:

$$T_i^{wait} = \begin{cases} 0, & \text{if } i = 0 \text{ or } \mathcal{B}_i = \emptyset \\ \sum_{j \in \mathcal{B}_i} \sum_{k=1}^{|\boldsymbol{p}_s|-1} \frac{y_j}{R_{p_{sk}, p_{s(k+1)}}}, & \text{else} \end{cases} \tag{6}$$

where $\boldsymbol{p}_s = \boldsymbol{p}_i \cap \boldsymbol{p}_j$.

Based on (1), (5), and (6), we then have

$$T_i^{total} = T_i^{trans} + T_i^{wait} + T_i^{comp} \tag{7}$$

*4) Energy Consumption:* With $T_i^{total}$ and (2)-(3), the energy consumption $E_i^{total}$ can then be expressed by:

$$E_i^{total} = E_i^{comp} + \sum_{j \in \mathcal{C}_i} E_{i,j}^{comm} \tag{8}$$

In the above equation, $\mathcal{C}_i$ is the set of children of node $i$, whose subtasks will be relayed by node $i$.

### C. Problem Formulation

Mathematically, the multi-objective optimization problem can be formulated as follows:

$$\begin{aligned} \mathcal{P}_0: \min_{\boldsymbol{y},\boldsymbol{o}} \ & \mathcal{J}(\boldsymbol{y}, \boldsymbol{o}) \\ s.t. \ & \sum_{i=0}^{N} y_i = Y & C1 \\ & 0 \le y_i \le Y, \forall i \in \mathcal{N} & C2 \\ & o_i \in \mathcal{N} \setminus \{0\}, \forall i \in \mathcal{N} \setminus \{0\} & C3 \\ & o_i \ne o_j, \forall i, j \in \mathcal{N} \setminus \{0\}, i \ne j & C4 \end{aligned}$$

Constraint $C1$ ensures that the total assigned workload sums up to the total task size $Y$. Constraints $C2$-$C4$ guarantee that each decision variable takes on valid values.

## V. Joint Optimization of Task Allocation and Scheduling

In this section, we introduce two exact methods to find the optimal solution to the joint task allocation and scheduling problem $\mathcal{P}_0$.

### A. Centralized MILP-based Optimization (CMO)

*1) Algorithm Description:* It is noted that $\mathcal{P}_0$, which aims to minimize the maximum cost of individual nodes, is a min-max optimization problem. Hence, we can convert it into an equivalent mixed integer linear programming (MILP) problem by introducing an auxiliary variable $z$ as follows:

$$\begin{aligned} \mathcal{P}_1: \min_{\boldsymbol{y},\boldsymbol{o},z} \ & z \\ s.t. \ & z \ge J_i(\boldsymbol{y}, \boldsymbol{o}), \forall i \in \mathcal{N} \\ & C1 - C4 \end{aligned} \tag{9}$$

Then, the minimum cost $\mathcal{J}^\star = z^\star$, where $z^\star$ is the minimum value of $z$ found by solving $\mathcal{P}_1$.

Problem $\mathcal{P}_1$ can be further decomposed into two subproblems. The first subproblem aims to optimize the task allocation $\boldsymbol{y}$, given a particular offloading order denoted as $\boldsymbol{o} = \boldsymbol{o}_k$:

$$\begin{aligned} \mathcal{P}_1^{(a)}: \min_{\boldsymbol{y},z} \ & z \\ s.t. \ & z \ge J_i(\boldsymbol{y}, \boldsymbol{o}_k), \forall i \in \mathcal{N} \\ & C1 - C2 \end{aligned}$$

Denote the optimal solution to problem $\mathcal{P}_1^{(a)}$ at $\boldsymbol{o} = \boldsymbol{o}_k$ as $\{\boldsymbol{y}^*(\boldsymbol{o}_k), z^*(\boldsymbol{o}_k)\}$. The second subproblem aims to optimize the offloading order $\boldsymbol{o}$:

$$\mathcal{P}_1^{(b)}: \min_{\boldsymbol{o}_k} \ z^*(\boldsymbol{o}_k)$$

Now let's consider subproblem $\mathcal{P}_1^{(a)}$, which can be solved using Lagrange multipliers [57]. Particularly, the Lagrangian function can be defined as follows:

$$\mathcal{L}(\boldsymbol{y}, z, \boldsymbol{\lambda}, \mu) = z + \sum_{i=0}^{N} \lambda_i \left[ J_i(\boldsymbol{y}, \boldsymbol{o}_k) - z \right] + \mu \left( \sum_{i=0}^{N} y_i - Y \right),$$

where $\boldsymbol{\lambda} = [\lambda_0, \lambda_1, \ldots, \lambda_N]$ and $\mu$ are Lagrangian multipliers. $\lambda_i \ge 0$, $\forall i \in \mathcal{N}$. Define

$$g(\boldsymbol{\lambda}, \mu) = \min_{\boldsymbol{y},z} \mathcal{L}(\boldsymbol{y}, z, \boldsymbol{\lambda}, \mu).$$

The dual optimization problem is then constructed as follows:

$$\begin{aligned} \max_{\boldsymbol{\lambda},\mu} \ & g(\boldsymbol{\lambda}, \mu) \\ s.t. \ & \boldsymbol{\lambda} \ge 0 \end{aligned} \tag{10}$$

As the objective function and the inequality constraints in our problem are convex, and the equality constraints are affine and strictly feasible, Slater's condition [58] is satisfied and the strong duality holds. That means the optimal value of the primal problem $\mathcal{P}_1^{(a)}$ is equal to the optimal value of its dual problem (10). The optimal solution to $\mathcal{P}_1^{(a)}$ can then be found by solving the following equation set, known as the Karush-Kuhn-Tucker (KKT) conditions [59]:

$$\begin{cases} \frac{\partial}{\partial y_i} \mathcal{L}(\boldsymbol{y}, z, \boldsymbol{\lambda}, \mu) = 0, \ \forall i \in \mathcal{N} \\ \frac{\partial}{\partial z} \mathcal{L}(\boldsymbol{y}, z, \boldsymbol{\lambda}, \mu) = 0 \\ \sum_{i=0}^{N} y_i = Y \\ \lambda_i (J_i(\boldsymbol{y}) - z) = 0, \ \forall i \in \mathcal{N} \\ J_i(\boldsymbol{y}) - z \le 0, \ \forall i \in \mathcal{N} \\ \lambda_i \ge 0, \ \forall i \in \mathcal{N} \end{cases} \tag{11}$$

To solve problem $\mathcal{P}_1^{(b)}$, we can use exhaustive search. This involves computing the cost $z^*(\boldsymbol{o}_k)$ for each possible offloading order $\boldsymbol{o}_k$ and selecting the one that yields the smallest cost. However, as $\boldsymbol{o}_k$ can take $N!$ possible values, evaluating each possible value is time-consuming. A significant reduction in the number of possible values to evaluate can be achieved by exploiting the parallelism in sending subtasks belonging to different subtrees of the master. Specifically, the offloading orders for nodes in any subtree $\mathcal{A}_t$ are independent of those in any other subtree $\mathcal{A}_{t'}$, where $t, t' \in \mathcal{I}_1$ and $t \ne t'$. Therefore, the number of possible values of $\boldsymbol{o}_k$ that need to be evaluated can be reduced to $\prod_{t \in \mathcal{I}_1} |\mathcal{A}_t|!$. Algorithm 1 summarizes the procedure of the proposed approach, named the centralized MILP-based optimization (CMO).

**Algorithm 1** CMO($\mathcal{T}$, $Y$)

**for** each $\boldsymbol{o}_k, k \in \{1, 2, \ldots, \prod_{i\in\mathcal{I}_1} |\mathcal{A}_i|!\}$ **do**
  Find $\{\boldsymbol{y}^*(\boldsymbol{o}_k), z^*(\boldsymbol{o}_k)\}$ by solving equation set (11);
**end for**
$\boldsymbol{o}^\star \leftarrow \arg\min_{\boldsymbol{o}_k} z^*(\boldsymbol{o}_k)$, $z^\star \leftarrow z^*(\boldsymbol{o}^\star)$, $\boldsymbol{y}^\star \leftarrow \boldsymbol{y}^*(\boldsymbol{o}^\star)$
**return** $z^\star, \boldsymbol{y}^\star, \boldsymbol{o}^\star$

*2) Computational Complexity Analysis:* As the equation set (11) involves $2N + 4$ unknown variables, solving it requires $O(N^3)$ amount of time in the worst case [60]. The computational complexity of CMO is hence $O(\prod_{t\in\mathcal{I}_1} |\mathcal{A}_t|!N^3)$, with a worst-case complexity of $O(N!N^3)$.

### B. Parallel MILP-based Optimization (PMO)

*1) Algorithm Description:* The parallelism involved in sending subtasks belonging to different subtrees of the master can be further harnessed to greatly enhance efficiency. Specifically, the key idea is to decompose problem $\mathcal{P}_1$ alternatively into two different subproblems. The first subproblem optimizes the task allocation and scheduling for nodes within each subtree, which can be solved in parallel. The second subproblem optimizes the total workload assigned to each subtree.

Mathematically, let $Y_t$ be the total workload assigned to nodes within the $t$-th subtree of the master, i.e., $Y_t = \sum_{i\in\mathcal{A}_t} y_i$, $t \in \mathcal{I}_1$. Additionally, let $\boldsymbol{y}_t = \{y_i | i \in \mathcal{A}_t\}$ and $\boldsymbol{o}_t = \{o_i | i \in \mathcal{A}_t\}$ represent the decision variables associated with nodes within the subtree $\mathcal{A}_t$. Then, the first subproblem can be formulated as follows:

$$
\begin{aligned}
\mathcal{P}_1^{(a')}: \min_{\boldsymbol{y}_t, \boldsymbol{o}_t, z_t} \quad & z_t \\
s.t.\ & z_t \geq J_i(\boldsymbol{y}_t, \boldsymbol{o}_t), \forall i \in \mathcal{A}_t \\
& \sum_{i\in\mathcal{A}_t} y_i = Y_t \\
& 0 \leq y_i \leq Y_t, \forall i \in \mathcal{A}_t \\
& o_i \in \mathcal{N} \setminus \{0\}, \forall i \in \mathcal{A}_t \\
& o_i \neq o_j, \forall i, j \in \mathcal{A}_t, i \neq j
\end{aligned}
$$

Since tasks assigned to different subtrees can be transmitted simultaneously, this problem can be solved independently and in parallel for different subtrees.

Suppose given $Y_t$, the optimal solution to problem $\mathcal{P}_1^{(a')}$ for subtree $\mathcal{A}_t$ is $\{\bar{z}_t(Y_t), \bar{\boldsymbol{y}}_t(Y_t), \bar{\boldsymbol{o}}_t(Y_t)\}$. The second subproblem aims to optimize the workload assigned to each subtree as well as to the master, denoted as $\boldsymbol{\mathcal{Y}} = \{Y_t | t \in \mathcal{I}_1\} \cup \{y_0\}$, which can be mathematically formulated as follows:

$$
\begin{aligned}
\mathcal{P}_1^{(b')}: \min_{\boldsymbol{\mathcal{Y}}} \quad & z \\
s.t.\ & z \geq \bar{z}_t(Y_t), \forall t \in \mathcal{I}_1 \\
& z \geq J_0(\boldsymbol{\mathcal{Y}}) \\
& y_0 + \sum_{t\in\mathcal{I}_1} Y_t = Y
\end{aligned}
$$

where $J_0(\boldsymbol{\mathcal{Y}}) = w_1 T_0^{total} + w_2 E_0^{total} = w_1 \frac{y_0 b}{f_0} + w_2 \left[\gamma_0 y_0 b (f_0)^2 + \sum_{t\in\mathcal{I}_1} \frac{e_0 Y_t}{R_{0,t}}\right]$. Of note, this subproblem can be conceptualized by abstracting each subtree as a single node.

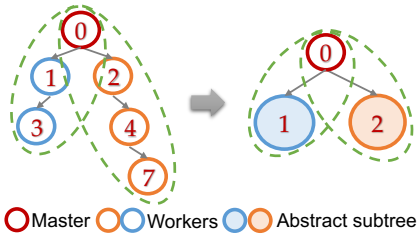



Figure 3. Illustration of how a network tree can be abstracted as a one-layer tree.

Therefore, $\mathcal{T}$ is abstracted as a one-layer tree (see Fig. 3). The optimization of the workload $Y_t$ assigned to each abstracted node (subtree) thus does not require consideration of the offloading order. Then, the optimal solution to $\mathcal{P}_1^{(b')}$, denoted as $Y_t^*$, $\forall t \in \mathcal{I}_1$, can be used to derive the optimal solution to the original problem $\mathcal{P}_1$. Particularly, $\boldsymbol{y}^\star = \{\bar{\boldsymbol{y}}_t(Y_t^*) | t \in \mathcal{I}_1\}$, and the optimal offloading order for nodes within each subtree $t$ is given by $\bar{\boldsymbol{o}}_t(Y_t^*)$.

Solving problem $\mathcal{P}_1^{(a')}$ given $Y_t$ is relatively straightforward. However, directly addressing subproblem $\mathcal{P}_1^{(b')}$ is challenging because $Y_t$ is continuous, and obtaining $\bar{z}_t(Y_t)$ in the constraints requires solving subproblem $\mathcal{P}_1^{(a')}$. Before we proceed with our approach to solving these subproblems, we present the following lemma and theorem, which allow for their simplification.

**Lemma 1.** *Given $\mathcal{T}$ and $Y$, for an arbitrary offloading order $\boldsymbol{o}_k$, suppose $\{\boldsymbol{y}^*(\boldsymbol{o}_k), z^*(\boldsymbol{o}_k)\}$ is an optimal solution to problem $\mathcal{P}_1^{(a)}$. Then, $\{\frac{Y'}{Y}\boldsymbol{y}^*(\boldsymbol{o}_k), \frac{Y'}{Y}z^*(\boldsymbol{o}_k)\}$ is an optimal solution to $\mathcal{P}_1^{(a)}$ when the task size is changed to $Y'$.*

*Proof.* As detailed in Sec. V, the optimal solution to $\mathcal{P}_1^{(a)}$ can be found by solving the equation set (11). Suppose $\{z^*(\boldsymbol{o}_k), \boldsymbol{y}^*(\boldsymbol{o}_k), \boldsymbol{\lambda}^*(\boldsymbol{o}_k), \mu^*(\boldsymbol{o}_k)\}$ is the obtained optimal solution for task size $Y$. Note that the cost of each node $J_i$ can be expressed as a linear combination of the task assignments $\{y_0, y_1, \ldots, y_N\}$, i.e., $J_i = a_{i,0}y_0 + a_{i,1}y_1 + ... + a_{i,|\mathcal{N}|}y_{|\mathcal{N}|}$, where $\{a_{i,0}, ..., a_{i,|\mathcal{N}|}\}$ are constants that depend on network characteristics. Hence, equation set (11) can be simplified as:

$$
\begin{cases}
\sum_{i=0}^{|\mathcal{N}|} \lambda_i \frac{\partial(a_{i,0}y_0 + a_{i,1}y_1 + ... + a_{i,|\mathcal{N}|}y_{|\mathcal{N}|})}{\partial y_i} + \mu = 0, \ \forall i \in \mathcal{N} \\
1 - \sum_{i=0}^{|\mathcal{N}|} \lambda_i = 0 \\
\sum_{i=0}^{N} y_i = Y \\
\lambda_i \left(a_{i,0}y_0 + a_{i,1}y_1 + ... + a_{i,|\mathcal{N}|}y_{|\mathcal{N}|} - z\right) = 0, \ \forall i \in \mathcal{N} \\
a_{i,0}y_0 + a_{i,1}y_1 + ... + a_{i,|\mathcal{N}|}y_{|\mathcal{N}|} - z \leq 0, \ \forall i \in \mathcal{N} \\
\lambda_i \geq 0, \ \forall i \in \mathcal{N}
\end{cases}
$$

When the task size is changed to $Y'$, the solution $\{\frac{Y'}{Y}z^*(\boldsymbol{o}_k), \frac{Y'}{Y}\boldsymbol{y}^*(\boldsymbol{o}_k), \boldsymbol{\lambda}^*(\boldsymbol{o}_k), \frac{Y'}{Y}\mu^*(\boldsymbol{o}_k)\}$ satisfies the above equation set. This indicates that it is an optimal solution to $\mathcal{P}_1^{(a)}$ for task size $Y'$. With this, the proof is now complete. □

Lemma 1 leads directly to the following theorem.

**Theorem 2.** *Given $\mathcal{T}$ and $Y$, suppose $\{z^\star, \boldsymbol{y}^\star, \boldsymbol{o}^\star\}$ is the optimal solution to problem $\mathcal{P}_1$. Then, $\{\frac{Y'}{Y}z^\star, \frac{Y'}{Y}\boldsymbol{y}^\star, \boldsymbol{o}^\star\}$ is the optimal solution to $\mathcal{P}_1$ when the task size is changed to $Y'$.*

*Proof.* Theorem 2 can be directly derived from Lemma 1 when $\boldsymbol{o}_k = \boldsymbol{o}^\star$. □

The proportionality property described in Theorem 2 infers that, given the optimal solution to $\mathcal{P}_1^{(a')}$ for any $Y_t'$, i.e., $\{\bar{z}_t(Y_t'), \bar{\boldsymbol{y}}_t(Y_t'), \bar{\boldsymbol{o}}_t(Y_t')\}$, the subproblem $\mathcal{P}_1^{(b')}$ can be simplified to:

$$\begin{aligned}\mathcal{P}_1^{(c')}: \min_{\boldsymbol{\mathcal{Y}}} \quad & z \\ s.t.\ & z \geq \frac{Y_t}{Y_t'}\bar{z}_t(Y_t'), \forall t \in \mathcal{I}_1 \\ & z \geq J_0(\boldsymbol{\mathcal{Y}}) \\ & y_0 + \sum_{t\in\mathcal{I}_1} Y_t = Y\end{aligned}$$

Since $Y_t'$ and $\bar{z}_t(Y_t')$ are known (by solving $\mathcal{P}_1^{(a')}$), $\mathcal{P}_1^{(c')}$ is now straightforward to solve.

Next, we describe our approach to solve subproblems $\mathcal{P}_1^{(a')}$ and $\mathcal{P}_1^{(c')}$. Particularly, for $\mathcal{P}_1^{(a')}$, we can solve it by leveraging the CMO algorithm (Algorithm 1). For each subtree $\mathcal{A}_t$, $t \in \mathcal{I}_1$, we run the CMO algorithm on the tree formed by $\mathcal{A}_t$ as well as the master (as highlighted by the green dashed circle in Fig. 3), denoted as $\mathcal{T}_t$, where the input $Y$ can take any value. Suppose the output generated by CMO for $\mathcal{T}_t$ is denoted as $\{\tilde{z}_t, \tilde{\boldsymbol{y}}_t, \tilde{\boldsymbol{o}}_t\}$, where $\tilde{\boldsymbol{y}}_t = \{\tilde{y}_i | i \in \mathcal{A}_t\} \cup \{\tilde{y}_0\}$ specifies the tasks allocated to the nodes within $\mathcal{T}_t$. Then we let $Y_t' = \sum_{i\in\mathcal{A}_t} \tilde{y}_i$, $\bar{z}_t(Y_t') = \max\{J_i(\tilde{\boldsymbol{y}}_t, \tilde{\boldsymbol{o}}_t) | i \in \mathcal{A}_t\}$. Given $Y_t'$ and $\bar{z}_t(Y_t')$ for each $t \in \mathcal{I}_1$, we can then solve the subproblem $\mathcal{P}_1^{(c')}$ using a commercial solver, such as Gurobi [61] and CVX [62].

Denote the optimal solution to subproblem $\mathcal{P}_1^{(c')}$ as $\boldsymbol{\mathcal{Y}}^* = \{Y_t^* | t \in \mathcal{I}_1\} \cup \{y_0^*\}$. The optimal solution to the original problem $\mathcal{P}_1$ can then be derived as:

$$y_i^\star = \frac{Y_t^*}{Y_t'}\tilde{y}_i, \forall i \in \mathcal{A}_t, t \in \mathcal{I}_1 \tag{12}$$

$$z^\star = \max\left\{\max_{t\in\mathcal{I}_1} \frac{Y_t^*}{Y_t'}\bar{z}_t(Y_t'), J_0(\boldsymbol{\mathcal{Y}}^*)\right\} \tag{13}$$

$\tilde{\boldsymbol{o}}_t$, $t \in \mathcal{I}_1$, specifies the optimal offloading order for subtasks assigned to each node within each subtree $\mathcal{A}_t$, where subtasks for different subtrees can be transmitted simultaneously.

Algorithm 2 summarizes the procedure of the parallel MILP-based optimization (PMO) method.

**Algorithm 2** PMO($\mathcal{T}$, $Y$)

1: **for** each $t \in \mathcal{I}_1$ **do**
2: $\{\tilde{z}_t, \tilde{\boldsymbol{y}}_t, \tilde{\boldsymbol{o}}_t\} \leftarrow$ CMO($\mathcal{T}_t$, $Y$) in parallel;
3: **end for**
4: Find $\boldsymbol{\mathcal{Y}}^*$ by solving $\mathcal{P}_1^{(c')}$;
5: Calculate $\boldsymbol{y}^\star, z^\star$ using (12) and (13), respectively;
6: **return** $z^\star, \boldsymbol{y}^\star$, and $\{\tilde{\boldsymbol{o}}_t | t \in \mathcal{I}_1\}$;

*2) Computational Complexity Analysis:* Since subtree $\mathcal{T}_t$, $t \in \mathcal{I}_1$, contains $|\mathcal{A}_t| + 1$ nodes, CMO($\mathcal{T}_t, Y$) requires $O((|\mathcal{A}_t| + 1)!(|\mathcal{A}_t| + 1)^3)$ time to execute. The complexity of PMO is $O(\max_{t\in\mathcal{I}_1}(|\mathcal{A}_t| + 1)!(|\mathcal{A}_t| + 1)^3)$ with a worst-case complexity of $\mathcal{O}(N!N^3)$.

### C. Offline-Online Computation

Despite the fact that the computational complexity of CMO and PMO grows rapidly as the network expands, both can be executed in real-time by transferring the majority of the computations offline. This can be achieved by leveraging the proportionality property presented in Theorem 2. Particularly, for any task size $Y$, we can execute Algorithm 1 *offline* to derive a baseline optimal solution $\{z^\star, \boldsymbol{y}^\star, \boldsymbol{o}^\star\}$. Then, during *online* computations, upon receiving a new task $Y'$, we can readily compute the associated optimal solution in real-time by scaling the baseline with a factor $\frac{Y'}{Y}$, i.e., $\{\frac{Y'}{Y}z^\star, \frac{Y'}{Y}\boldsymbol{y}^\star, \boldsymbol{o}^\star\}$.

This offline-online computation scheme also equips CMO and PMO with the ability to potentially handle dynamic networks with time-varying network characteristics. One approach to deploying them in dynamic networks is to periodically execute Algorithm 1 to update the baseline solution with the latest network information. Alternatively, the baseline solution can be updated when significant network changes occur, such as alterations in the network topology.

## VI. Heuristic Methods

In this section, we introduce three heuristic methods to further speed up the computation.

### A. Worker Selection

Through simulation studies, as presented in Sec. VII, we find that the solutions produced by CMO and PMO typically improve as more workers participate in computations. However, the rate of performance improvement diminishes as the network size reaches a certain threshold. This observation inspires us to consider selecting a subset of workers that contribute the most to performance improvement. Next, we introduce two worker selection methods: 1) a *node pruning* (NP) strategy, and 2) a *level pruning* (LP) strategy. These methods can be applied either individually or in combination. When PMO is utilized, they can be employed to prune each subtree $\mathcal{T}_t$ before executing Line 2 in Algorithm 2.

*1) Node Pruning (NP):* The key idea of NP is to "prune" nodes that are too costly to use. Specifically, this strategy evaluates each node one by one. For a given node $i$, it estimates the cost of using this node by performing partial offloading [63], which finds the optimal task partition between the master and node $i$ exclusively. The obtained cost, denoted as $z_i^p$, is then compared with the cost of local computing, i.e., the cost of processing the entire task $Y$ at the root node, denoted as $z^{(0)}$, which can be obtained by running CMO($\mathcal{I}_0, Y$). If the cost reduction, measured by $\frac{z^{(0)} - z_i^p}{z^{(0)}}$, exceeds a predefined threshold $\theta_p$, node $i$ is selected; otherwise, it is "pruned". Here, "prune" means that no workload is assigned to the node. If the node is a leaf, it is removed from the tree. However, if it is an intermediate node with unpruned children, it remains and only acts as a relay.

*2) Level Pruning (LP):* The key idea of LP is to trim nodes that are excessively distant from the master node, whose computing resources are too costly to use considering the significant communication costs. Specifically, this strategy

evaluates the top $\xi$ levels of the original network tree, removing levels from $\xi+1$ to $H$. The resulting tree, denoted as $\mathcal{T}^{\xi}$, satisfies $\mathcal{T}^{\xi} = \mathcal{T} \setminus \cup_{l=\xi}^{H} \mathcal{I}_l$.

### B. Genetic Algorithm

The worker selection methods allow us to reduce workers but may prune nodes that could significantly improve system performance. Here, we introduce a genetic algorithm (GA) [64] that allows us to evaluate large networks. It involves two phases: *initialization* and *training*. In the initialization phase, a population set $\boldsymbol{O} = \{\boldsymbol{o}_k\}$ is first randomly generated, which consists of $P$ offloading orders (chromosomes). The corresponding optimal task partition $\boldsymbol{y}^*(\boldsymbol{o}_k)$ and cost $z^*(\boldsymbol{o}_k)$ are then computed by solving (11), where $z^*(\boldsymbol{o}_k)$ is the fitness of the chromosome $\boldsymbol{o}_k$. Following the initialization, the training phase starts with Elitism, which picks the top $\alpha\%$ of the fittest members from the current population and propagates them to the next generation. After that, an iterative procedure is performed to create offspring. In each iteration, two offloading orders are randomly picked from the current population $\boldsymbol{O}$ according to the probability distribution $\left\{ \frac{1/z^*(\boldsymbol{o}_k)}{\sum_{j=1}^{P} 1/z^*(\boldsymbol{o}_j)} \middle| k \in \{1, 2, \ldots, P\} \right\}$. Ordered crossover [65] is then applied to create offspring. Subsequently, with a low probability $\beta$, mutation is performed to introduce diversity into the new population by shuffling individual offloading orders. The algorithm terminates upon meeting the stopping condition at which point it outputs the best solution found. In our simulations, we set the stopping condition for GA to be reaching a maximum number of generations, denoted as $G$.

## VII. Simulation Studies

In this section, we conduct simulation studies to evaluate the performance of the proposed approaches. We start by describing the experiment setup in Sec. VII-A. Next, we conduct two sets of studies to evaluate the optimality and efficiency of the proposed approaches in Sec. VII-B and Sec. VII-C, respectively. We then investigate the impact of key parameters in Sec. VII-D, followed by an analysis of the effects of network characteristics.

### A. Experiment Setup

We evaluated the proposed approaches on network graphs generated using model [66], with nodes representing edge servers and edges denoting the communication links. Each graph was transformed into a shortest-path tree via the Dijkstra's algorithm. Server computing capacities $f_i$ were uniformly sampled from $[1, 10]$ GHz, and transmission rates $R_{i,j}$ were randomly assigned from $[10, 100]$ Gbps to model heterogeneous edge computing systems. The effective switched capacity ($\gamma_i = 10^2$) and transmission power ($e_i = 30$dBm) were set based on measurements from prior work [3]. Tasks were configured with a size of $Y = 1$Gbits and a computational intensity of $b = 10^6$ cycles/Gbit. All simulations were executed on an Alienware Aurora 15 (Intel i9, 64GB RAM) using Python 3.8, with Gurobi 9.5 for MILP optimization.

### B. Optimality Analysis

We first evaluate the optimality of the two optimal approaches, CMO and PMO. For comparison, we implement the following four state-of-the-art distributed computing and computation offloading schemes as benchmarks:

- **Local computing (Local)**: In this approach, the master executes the entire task locally.
- **Partial offloading (Partial)**: In this approach, the master offloads part of the task to one of its one-hop neighbors. The offloading ratio and offloadee selection are optimized to minimize the task completion time.
- **Master-worker distributed computing (Master-worker)**: In this approach, the master distributes the task to its one-hop neighbors using the master-worker paradigm. The task allocation is optimized to minimize the task completion time.
- **Multi-hop offloading (Multi-hop)** [67]: In this approach, the master offloads the whole task to the most powerful and reliable server in the network, which may be multiple hops away.

Their performances are evaluated on four network graphs, which are transformed into trees with varying depths and breadths as illustrated in Fig. 4.

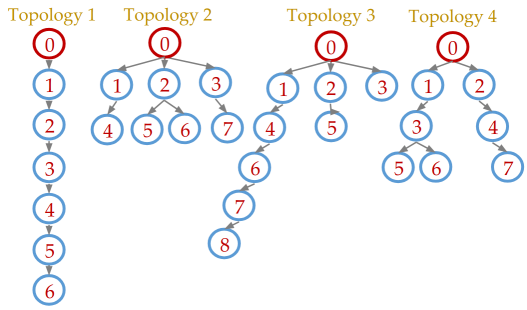



Figure 4. Network topologies evaluated in simulation studies.

In the first experiment, we set the weights in the objective function to $w_1 = 1$ and $w_2 = 0$, which transforms the objective of our approach to minimize the task completion time only, just like the benchmarks. As shown in Fig. 5(a), our approaches outperform all benchmarks across all scenarios. Among the benchmarks, **Local** and **Multi-hop** have the poorest performance since they only use the computing resources from a single server. **Partial** outperforms local computing and **Multi-hop** by utilizing the resources from two servers. The **Master-worker** achieves even better performance by utilizing computing resources from all servers within one hop. This experiment provides evidence that increasing the utilization of resources leads to better computing performance.

In the second experiment, we set the weights $w_1 = 0, w_2 = 1$, thereby transforming the objective of our approach to focus on minimizing energy consumption. As shown in Fig. 5(b), our approach continues to achieve the best performance across all scenarios. Among the benchmarks, **Multi-hop** exhibits the worst performance due to the energy consumption associated with multi-hop transmissions for large amount of data. On the

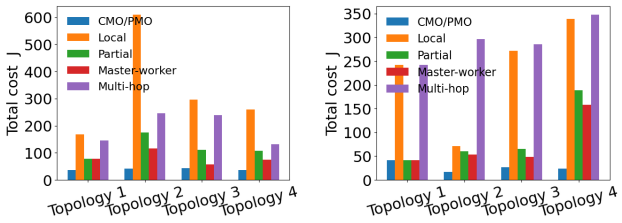



Figure 5. Total cost $J$ of different methods when considering (a) time consumption only and (b) only energy consumption only.

contrary, while **Local** computing also performs poorly because of the prolonged local computation time.

In the last experiment, we randomly set the weights to $w_1 = 0.5$ and $w_2 = 0.05$, so that both computation efficiency and energy consumption are considered in our approaches. Note that these weight values are also used in the following experiments. Fig. 6(a) shows the comparison results, demonstrating the promising performance of our approaches in balancing task completion time and energy consumption. In Fig. 6(b) and Fig. 6(c), we show the task completion time, i.e., $\max_{i\in\mathcal{N}} T_i^{total}$, and the maximum energy consumption by any server, i.e., $\max_{i\in\mathcal{N}} E_i^{total}$, respectively. The results indicate that our approach generally outperforms all the benchmarks in optimizing both objectives. However, for Topology 2, **Local** computing achieves the best task completion time while exhibiting the worst performance in energy consumption, highlighting the inherent trade-off between these two objectives.

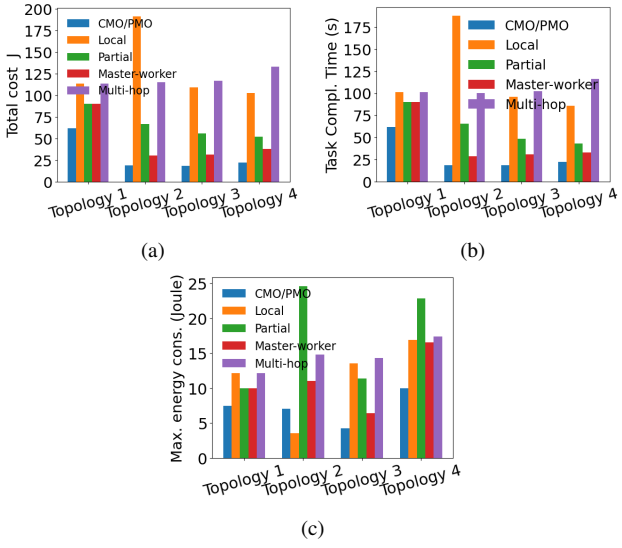



Figure 6. (a) Total cost, (b) task completion time, and (c) maximum energy consumption of different methods.

## C. Efficiency Analysis

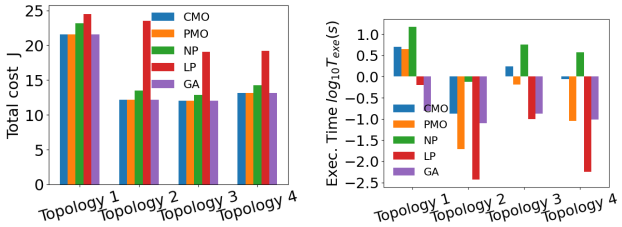
In this subsection, we evaluate the efficiency of the proposed optimal methods, CMO and PMO, as well as the proposed heuristic methods, NP, LP, and GA. For the implementation of the two worker selection methods, NP and LP, we first use them to prune the network tree, and then apply PMO to allocate tasks. GA is also implemented within the framework of PMO, and employed to determine the task allocation for each subtree, replacing CMO in Line 2 of Algorithm 2.

*1) Small-Scale Networks:* We first consider the four small-scale network topologies depicted in Fig. 4. In this experiment, the threshold parameter $\theta_p$ in NP is set to $0.312, 0.43, 0.3, 0.4$, respectively, such that one node in each topology is pruned. The threshold parameter $\xi$ in LP is set to $5, 1, 4, 2$ for the four topologies, respectively, resulting in the pruning of the last level of each topology. For GA, the parameters are configured as $G = 5, P = 4, \alpha = 0.2, \beta = 0.05$. To measure the efficiency of proposed approaches, we run each method 20 times and record the mean execution time, denoted as $T_{\text{exe}}$.

Fig. 7(a) shows the costs of the solutions found by the five methods. Comparing GA with the optimal methods, CMO and PMO, reveals that GA can find optimal solutions for small networks. This similarity in performance further demonstrates the optimality of GA for small networks. The worker selection methods, NP and LP, underperform compared to the other three methods, which is attributed to the reduced number of nodes involved in sharing the computational workload. Moreover, comparing the performance of LP across different topologies indicates that the extent of performance degradation is closely related to the proportion of nodes pruned from the network tree. Specifically, LP prunes 14.28%, 57.14%, 11.11%, and 37.5% of the nodes in the four topologies, respectively. The largest pruning proportion occur in Topology 2, resulting in the maximum level of performance degradation. For NP, as only one node is "pruned" in each topology, it performs better than LP in these scenarios.

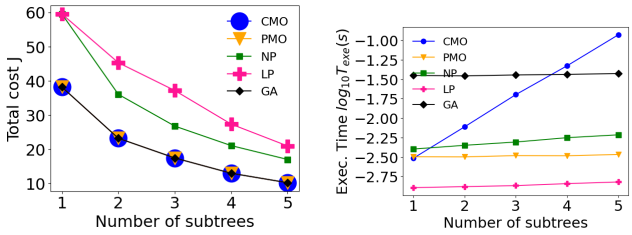
The base-10 logarithm of the execution time, i.e., $\log_{10} T_{\text{exe}}$, of each method is shown in Fig. 7(b). As expected, the optimal methods, CMO and PMO, are more time-consuming than the three heuristic methods. Moreover, PMO, being a parallelized version of CMO, significantly reduces execution time in Topologies 2-4 due to its parallelism. For Topology 1, since the root has a single subtree, PMO is equivalent to CMO. Among the heuristic methods, LP achieves the least execution time by pruning the most nodes and significantly reducing the search space. GA, on the other hand, is the least efficient and even underperforms PMO in Topologies 2 & 4. This suggests that for small networks, PMO can be directly applied. Furthermore, comparing NP and PMO, we can observe that NP does not improve efficiency in all scenarios, despite reducing the number of workers. This is because only one node is "pruned" in each topology, and the time saved by pruning is offset by the overhead generated by the pruning procedure.

As the performance of the proposed approaches largely depend on the network size, we further vary the network size by increasing the number of subtrees, $|\mathcal{I}_1|$, where each subtree consists of 2 levels and 1 node in each level. The scenario where the network size expands due to the growth of subtrees is explored in the subsequent subsection. In this experiment, we configure parameter $\theta_p$ in NP in a way such that one additional node is "pruned" when including an additional subtree. Parameter $\xi$ in LP is set to 1 in all cases, meaning that the node(s) in the last level are pruned. For GA, its parameters are configured as $P = 4, G = 100, \alpha = 0.2, \beta = 0.05$. Fig.




(a) (b)

Figure 7. (a) Total cost $J$ and (b) execution time of different methods for different network topologies.

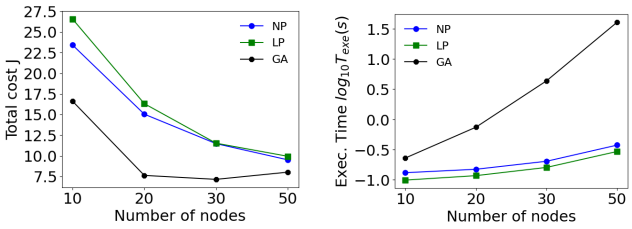



(a) (b)

Figure 8. (a) Total cost $J$ and (b) execution time of different methods as the number of subtrees increases.

8(a) shows the performance of different methods as the number of subtrees $|\mathcal{I}_1|$ increases. As we can see, increasing the number of subtrees results in a reduced total cost $J$, as more nodes are involved in sharing the computational workload. When comparing NP and LP, NP consistently outperforms LP. It's noteworthy that both methods prune the same number of nodes, each trimming one node from every subtree. This underscores the effectiveness of NP's worker selection process, which employs a more rigorous approach compared to LP that simply selects nodes at the top levels. However, the simplicity of LP makes it more efficient than NP, as shown in Fig. 8(b). Additionally, from Fig. 8(b), we can observe a significant increase in the execution time of CMO as more subtrees are considered, compared to the other four methods. This is due to the parallelism inherent in the other four methods. Moreover, comparing PMO with the other methods further demonstrates the good performance of PMO in both optimality and efficiency in cases of small networks.

*2) Large-Scale Networks:* In this experiment, we consider larger networks and evaluate the performance of the three heuristic methods, NP, LP, and GA. For the implementation of NP and LP, GA is applied after pruning to determine the task allocation. Given that all these methods evaluate subtrees in parallel and their efficiency is bounded by the largest subtree, we evaluate their performance on networks with a single subtree. This approach allows us to avoid considering the impact of the number of subtrees. These networks are randomly generated with node counts of 10, 20, 30, and 50. The parameters in NP and LP are configured to prune a similar number of nodes as follows: the threshold $\theta_p$ in NP is set to $0.36, 0.45, 0.5, 0.38$, and the threshold $\xi$ in LP is set to $4, 3, 3, 12$ for the corresponding networks, respectively. The parameters in GA, applied across all methods, are set to




(a) (b)

Figure 9. (a) Total cost $J$ and (b) execution time of different heuristic methods as the number of nodes in the network increases.

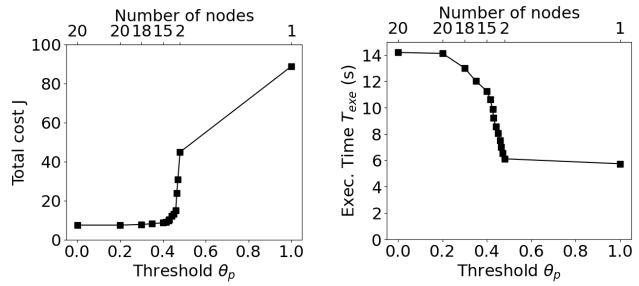
$P = 4, G = 100, \alpha = 0.2, \beta = 0.05$.

As shown in Fig. 9(a), the total cost $J$ generally decreases with the increase in network size for each method, as more nodes share the workload. Notably, the performance of GA degrades when the network size increases. This is becasue more servers are able to contribute to the task offloading. This degradation occurs because GA is configured to terminate after $G = 100$ iterations, and the larger search space introduced by the increased network size makes it more difficult to find high-quality solutions within the given iteration limit. Although GA's performance can be improved by increasing the value of $G$, doing so would further increase its execution time, as suggested by the results in Fig. 9(b). Comparing the three methods, we can observe that GA outperforms the other two methods by considering all nodes in the network. NP generates better solutions than LP, although they prune roughly the same number of nodes. Additionally, NP and LP achieve performance comparable to GA in large networks (greater than 30 nodes), but with significantly lower execution times, as shown in Fig. 9(b). This suggests that for large networks, NP and/or LP can be applied first to select a subset of workers, followed by GA for task allocation.

### *D. Parameter Impact Analysis*

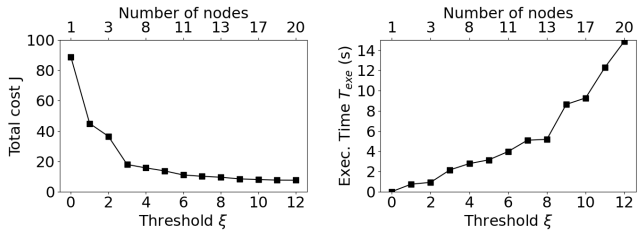
In this subsection, we investigate the impact of key parameters in the proposed heuristic methods, including (1) the threshold $\theta_p$ in NP, (2) threshold $\xi$ in LP. All experiments are conducted on the network with 20 nodes, as described in Sec. VII-C2. GA is employed for task allocation, using the same configuration detailed in Sec. VII-C2.

*1) Threshold $\theta_p$:* In NP, a worker is selected if the cost reduction from including this node exceeds the threshold $\theta_p$. Therefore, a higher threshold will result in fewer nodes being selected and more nodes being "pruned". This is demonstrated by the results shown in Fig. 10(a). As we can see, the best performance is achieved when $\theta_p = 0$, in which case no nodes are "pruned". The worst performance occurs at $\theta_p = 1$, where all workers are "pruned" and all computations are done locally at the master. Moreover, as $\theta_p$ decreases, more workers are selected, resulting in a decrease in cost $J$ (see Fig. 10(a)) but an increase in execution time $T_{\text{exe}}$ (see Fig. 10(b)). Notably, when $\theta$ is reduced to 0.4, $J$ tends to converge. This suggests that an appropriate value of $\theta_p$ that balances optimality and efficiency can be identified by selecting the value at which a sharp change in cost $J$ occurs.




(a) (b)

Figure 10. (a) Total cost $J$ and (b) execution time of NP as the threshold $\theta_p$ increases. The upper $x$-axes show the number of nodes selected as workers.




(a) (b)

Figure 11. (a) Total cost $J$ and (b) execution time of LP as the threshold $\xi$ increases.

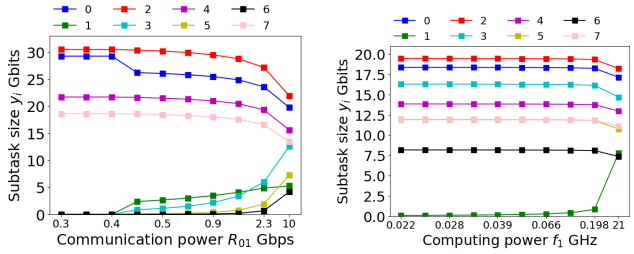
*2) Threshold $\xi$:* LP selects all nodes in the top $\xi$ levels as workers. In the special case when $\xi = 0$, no nodes are selected, resulting in all computations being conducted locally at the master. This leads to the highest cost $J$, as shown in Fig. 11(a). As $\xi$ increases and more nodes are selected, performance improves, as indicated by the decreasing cost $J$. However, the performance improvement slows down when $\xi$ exceeds 3. The best performance is achieved when $\xi$ reaches its maximum value, $H$ (height of the network tree), which is 12 in this experiment. Given the rapid increase in execution time with higher $\xi$, as shown in Fig. 11(b), a proper value for $\xi$ can be chosen at the point where the rate of decrease in $J$ slows down.

### *E. Impact of Network Characteristics*

The optimal task distribution decisions highly rely on the network characteristics. In this subsection, we explore how communication and computing parameters, specifically $R_{i,j}$ and $f_i$, affect these decisions. For this analysis, we focus on Topology 4, as shown in 4 in Fig. 4.

In the first experiment, we vary $R_{0,1}$, which represents the communication capacity between the master node (Node 0) and its left child (Node 1), from 0.3 Gbps to 10 Gbps. All other settings remain consistent with the previous studies. The optimal task allocation derived by PMO is shown in Fig.12(a). As the figure demonstrates, when $R_{0,1}$ is small, communication becomes a bottleneck, preventing the master from offloading tasks to Node 1 or to its descendants. However, as $R_{0,1}$ increases, more workload is offloaded to Node 1. Once $R_{0,1}$ exceeds certain thresholds, tasks are also offloaded to Node 1's children and even grandchildren. With more nodes contributing to workload distribution, nodes in the right subtree of the master begin to receive fewer tasks. This study suggests that if a communication link is too slow, both the connected downstream node and its descendants may be pruned from the topology before executing CMO/PMO. To understand the impact of the computing characteristic, we instead vary the computing power of Node 1, $f_1$, from 0.022 GHz to 21 GHz. As shown in Fig. VII-E, the master starts to offload tasks to Node 1 when its computing power exceeds a certain threshold. Moreover, when it shares more workload, the workloads assigned to all other nodes decrease simultaneously.

Notably, the network characteristics determine whether tasks are offloaded to a node, regardless of the total task size $Y$, as inferred from Theorem 2. To demonstrate this, we vary $Y$ while keeping the network characteristics constant. Table 1 summarizes the optimal task allocations and the corresponding total costs computed by PMO. As shown, when $Y$ increases, both the workload assigned to each node $y_i$ and the total cost $J$ rise proportionally.




(a) (b)

Figure 12. Optimal task allocations at different values of (a) $R_{0,1}$ and (b) $f_1$.

Table I
OPTIMAL TASK ALLOCATION AND TOTAL COST FOR DIFFERENT VALUES OF $Y$

| $y_i$ (Gbits) \ Server $i$ / $Y$ (Gbits) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | Cost $J$ |
|---|---|---|---|---|---|---|---|---|---|
| $Y = 10$ | 1.7 | 0.4 | 1.8 | 1.5 | 1.3 | 1.1 | 0.7 | 1.1 | 0.4 |
| $Y = 20$ | 3.4 | 0.8 | 3.6 | 3 | 2.6 | 2.2 | 1.4 | 2.2 | 0.8 |
| $Y = 40$ | 6.8 | 1.6 | 7.2 | 6 | 5.2 | 4.4 | 2.8 | 4.4 | 1.6 |

## VIII. CONCLUSION AND FUTURE WORKS

This paper introduces a novel multi-layered distributed computing framework that expands the computing capabilities of networked computing systems. Unlike conventional distributed computing paradigms that limit resource sharing to one-hop neighborhoods, our framework explores multi-hop offloading to enable resource sharing beyond one-hop neighborhoods, effectively utilizing the resources of the entire network. By transforming the network graph into a shortest-path tree, the resulting layered structure clearly captures node hierarchy and task distribution paths, simplifying the analysis of task allocation and scheduling. Building upon this layered structure, we formulated an MIP problem that jointly minimizes task completion time and energy consumption through optimal task allocation and scheduling. Two exact methods, CMO and PMO, were proposed to solve this problem optimally, with

PMO enhancing CMO's efficiency by exploiting the parallelism in task distribution across the master's subtrees. We also introduced an offline-online computation scheme to enable the real-time execution of CMO and PMO, allowing them to potentially handle dynamic networks with time-varying characteristics. To further enhance efficiency and scalability, three heuristic methods were introduced, including NP and LP for reducing the number of workers and GA for efficiently finding (sub-)optimal solutions. Simulation results demonstrate the superiority of our approaches over existing distributed computing and computation offloading schemes. Moreover, PMO shows promising performance in both optimality and efficiency for small networks. For larger networks, the results suggest applying NP or LP to reduce workers before using GA or PMO for task allocation. The results also show that NP outperforms LP in terms of optimality but is less efficient due to its more rigorous worker selection process. Additionally, studies on the impact of NP's and LP's parameters offer insights into their configurations. Lastly, the analysis of network characteristics highlights how the communication and computing capacities of individual servers influence task distribution decisions.

In the future, we will further enhance the practicality and reliability of the proposed methods by systematically exploring mobile edge servers, considering user-server association, and addressing scenarios where multiple users request computing services and may themselves be mobile. We will also investigate more general task structures (e.g., dependent tasks that can be represented as directed acyclic graphs), study more complicated edge system models, and explore the hierarchical master-work paradigm.

## ACKNOWLEDGMENT

We would like to thank National Science Foundation under Grant CAREER-2048266 and CCF-2402689 for the support of this work.

## REFERENCES

[1] S. Chen, H. Xu, D. Liu, B. Hu, and H. Wang, "A vision of iot: Applications, challenges, and opportunities with china perspective," *IEEE Internet of Things journal*, vol. 1, no. 4, pp. 349–359, 2014.

[2] K. Ma and J. Xie, "Joint task allocation and scheduling for multi-hop distributed computing," in *ICC 2024-IEEE International Conference on Communications*. IEEE, 2024, pp. 2664–2669.

[3] ——, "Joint task allocation and scheduling for multi - hop distributed computing," in *ICC 2024 - IEEE International Conference on Communications*, 2024, pp. 2664–2669.

[4] S. Hamdan, M. Ayyash, and S. Almajali, "Edge-computing architectures for internet of things applications: A survey," *Sensors*, vol. 20, no. 22, p. 6441, 2020.

[5] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[6] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[7] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, 2012, pp. 13–16.

[8] K. Lu, J. Xie, Y. Wan, and S. Fu, "Toward uav-based airborne computing," *IEEE Wireless Communications*, vol. 26, no. 6, pp. 172–179, 2019.

[9] B. Wang, J. Xie, K. Lu, Y. Wan, and S. Fu, "Learning and batch-processing based coded computation with mobility awareness for networked airborne computing," *IEEE Transactions on Vehicular Technology*, vol. 72, no. 5, pp. 6503–6517, 2023.

[10] H. Zhang, B. Wang, R. Wu, J. Xie, Y. Wan, S. Fu, and K. Lu, "Exploring networked airborne computing: A comprehensive approach with advanced simulator and hardware testbed," *Unmanned Systems*, 2023.

[11] W. Fan, Y. Su, J. Liu, S. Li, W. Huang, F. Wu, and Y. Liu, "Joint task offloading and resource allocation for vehicular edge computing based on v2i and v2v modes," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 4, pp. 4277–4292, 2023.

[12] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[13] M. Trigka and E. Dritsas, "Edge and cloud computing in smart cities," *Future Internet*, vol. 17, no. 3, p. 118, 2025.

[14] J. Linderoth, M. Yoder *et al.*, "Metacomputing and the master-worker paradigm," *Mathematics and Computer Science Division, Argonne National Laboratory, Tech. Rep. ANL/MCS-P792–0200*, 2000.

[15] Z. Kuang, L. Li, J. Gao, L. Zhao, and A. Liu, "Partial offloading scheduling and power allocation for mobile edge computing systems," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6774–6785, 2019.

[16] Q. Tang, H. Lyu, G. Han, J. Wang, and K. Wang, "Partial offloading strategy for mobile edge computing considering mixed overhead of time and energy," *Neural Computing and Applications*, vol. 32, pp. 15 383–15 397, 2020.

[17] Z. Liu, M. Liwang, S. Hosseinalipour, H. Dai, Z. Gao, and L. Huang, "Rfid: Towards low latency and reliable dag task scheduling over dynamic vehicular clouds," *IEEE Transactions on Vehicular Technology*, vol. 72, no. 9, pp. 12 139–12 153, 2023.

[18] J. Chen, H. Xing, Z. Xiao, L. Xu, and T. Tao, "A drl agent for jointly optimizing computation offloading and resource allocation in mec," *IEEE Internet of Things Journal*, vol. 8, no. 24, pp. 17 508–17 524, 2021.

[19] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Transactions on Wireless Communications*, vol. 16, no. 3, pp. 1397–1411, 2016.

[20] S. Barbarossa, S. Sardellitti, and P. Di Lorenzo, "Joint allocation of computation and communication resources in multiuser mobile cloud computing," in *Proceedings of IEEE 14th Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. IEEE, 2013, pp. 26–30.

[21] J. Li, H. Gao, T. Lv, and Y. Lu, "Deep reinforcement learning based computation offloading and resource allocation for mec," in *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2018, pp. 1–6.

[22] Y. Guo, D. Ma, H. She, G. Gui, C. Yuen, H. Sari, and F. Adachi, "Deep deterministic policy gradient-based intelligent task offloading for vehicular computing with priority experience playback," *IEEE Transactions on Vehicular Technology*, 2024.

[23] W. Zhao, Y. Cheng, Z. Liu, X. Wu, and N. Kato, "Asynchronous drl based multi-hop task offloading in rsu-assisted iov networks," *IEEE Transactions on Cognitive Communications and Networking*, 2024.

[24] C. Chen, Y. Zeng, H. Li, Y. Liu, and S. Wan, "A multihop task offloading decision model in mec-enabled internet of vehicles," *IEEE Internet of Things Journal*, vol. 10, no. 4, pp. 3215–3230, 2022.

[25] L. Liu, M. Zhao, M. Yu, M. A. Jan, D. Lan, and A. Taherkordi, "Mobility-aware multi-hop task offloading for autonomous driving in vehicular edge computing and networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 2, pp. 2169–2182, 2022.

[26] Z. Deng, Z. Cai, and M. Liang, "A multi-hop vanets-assisted offloading strategy in vehicular mobile edge computing," *IEEE Access*, vol. 8, pp. 53 062–53 071, 2020.

[27] W. Zhao, P. Gao, X. Hong, X. Zheng, and N. Kato, "Ppo based task offloading with ekf for position prediction in rsu-assisted iov," *IEEE Transactions on Cognitive Communications and Networking*, 2025.

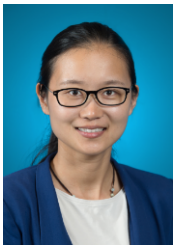
[28] J. Zhang, Y. Liu, and E. Yeh, "Result and congestion aware optimal routing and partial offloading in collaborative edge computing," *arXiv preprint arXiv:2205.00714*, 2022.

[29] J. Xie, Y. Jia, W. Wen, Z. Chen, and L. Liang, "Dynamic d2d multihop offloading in multi-access edge computing from the perspective of learning theory in games," *IEEE Transactions on Network and Service Management*, vol. 20, no. 1, pp. 305–318, 2022.

[30] H. Zhang, Y. Yang, B. Shang, and P. Zhang, "Joint resource allocation and multi-part collaborative task offloading in mec systems," *IEEE*

*Transactions on Vehicular Technology*, vol. 71, no. 8, pp. 8877–8890, 2022.
[31] C. Funai, C. Tapparello, and W. Heinzelman, "Computational offloading for energy constrained devices in multi-hop cooperative networks," *IEEE Transactions on Mobile Computing*, vol. 19, no. 1, pp. 60–73, 2019.
[32] B. Wang, J. Xie, K. Lu, Y. Wan, and S. Fu, "On batch-processing based coded computing for heterogeneous distributed computing systems," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 3, pp. 2438–2454, 2021.
[33] ——, "Multi-agent reinforcement learning based coded computation for mobile ad hoc computing," in *ICC 2021-IEEE International Conference on Communications*. IEEE, 2021, pp. 1–6.
[34] H. Zhang, J. Xie, and X. Zhang, "Communication-efficient $\delta$-stepping for distributed computing systems," in *Proceedings of International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE, 2023, pp. 369–374.
[35] K. Aida, W. Natsume, and Y. Futakata, "Distributed computing with hierarchical master-worker paradigm for parallel branch and bound algorithm," in *The 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE, 2003, pp. 156–163.
[36] Z. Hong, W. Chen, H. Huang, S. Guo, and Z. Zheng, "Multi-hop cooperative computation offloading for industrial iot–edge–cloud computing environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 12, pp. 2759–2774, 2019.
[37] Y. Wang, H. Wang, and X. Wei, "Energy-efficient uav deployment and task scheduling in multi-uav edge computing," in *Proceedings of International Conference on Wireless Communications and Signal Processing (WCSP)*. IEEE, 2020, pp. 1147–1152.
[38] H. Qi, M. Liwang, X. Wang, L. Li, W. Gong, J. Jin, and Z. Jiao, "Bridge the present and future: A cross-layer matching game in dynamic cloud-aided mobile edge networks," *IEEE Transactions on Mobile Computing*, 2024.
[39] F. Liu, J. Huang, and X. Wang, "Joint task offloading and resource allocation for device-edge-cloud collaboration with subtask dependencies," *IEEE Transactions on Cloud Computing*, vol. 11, no. 3, pp. 3027–3039, 2023.
[40] Y. Sun, Z. Wu, K. Meng, and Y. Zheng, "Vehicular task offloading and job scheduling method based on cloud-edge computing," *IEEE Transactions on Intelligent Transportation Systems*, 2023.
[41] W. Almuseelem, "Energy-efficient and security-aware task offloading for multi-tier edge-cloud computing systems," *IEEE Access*, 2023.
[42] T. Hwang, C. Yang, G. Wu, S. Li, and G. Y. Li, "Ofdm and its wireless applications: A survey," *IEEE transactions on Vehicular Technology*, vol. 58, no. 4, pp. 1673–1694, 2008.
[43] G. Ramkumar, "Enhancing cellular iot access with multi-hop uplink noma, and mobile edge computing for smart cities," in *2025 International Conference on Electronics and Renewable Systems (ICEARS)*. IEEE, 2025, pp. 514–519.
[44] Y. Yang, Y. Shi, X. Cui, J. Li, and X. Zhao, "A hybrid decision-making framework for uav-assisted mec systems: Integrating a dynamic adaptive genetic optimization algorithm and soft actor–critic algorithm with hierarchical action decomposition and uncertainty-quantified critic ensemble," *Drones*, vol. 9, no. 3, p. 206, 2025.
[45] S. Zhang, X. Tong, K. Chi, W. Gao, X. Chen, and Z. Shi, "Stackelberg game-based multi-agent algorithm for resource allocation and task offloading in mec-enabled c-its," *IEEE Transactions on Intelligent Transportation Systems*, 2025.
[46] A. Kaur, C. R. Krishna, and N. V. Patil, "A comprehensive review on software-defined networking (sdn) and ddos attacks: Ecosystem, taxonomy, traffic engineering, challenges and research directions," *Computer Science Review*, vol. 55, p. 100692, 2025.
[47] Z. Safavifar, E. Gyamfi, E. Mangina, and F. Golpayegani, "Multi-objective deep reinforcement learning for efficient workload orchestration in extreme edge computing," *IEEE Access*, 2024.
[48] F. Zhou, Y. Wu, R. Q. Hu, and Y. Qian, "Computation rate maximization in uav-enabled wireless-powered mobile-edge computing systems," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 9, pp. 1927–1941, 2018.
[49] H. Li, K. Ota, and M. Dong, "Learning iot in edge: Deep learning for the internet of things with edge computing," *IEEE Network*, vol. 32, no. 1, pp. 96–101, 2018.
[50] J. Chen and J. Xie, "Joint task scheduling, routing, and charging for multi-uav based mobile edge computing," in *Proceedings of IEEE International Conference on Communications*. IEEE, 2022, pp. 1–6.
[51] N. Fofana, A. B. Letaifa, and A. Rachedi, "Intelligent task offloading in vehicular networks: a deep reinforcement learning perspective," *IEEE Transactions on Vehicular Technology*, 2024.
[52] G. Wu, Z. Liu, M. Fan, and K. Wu, "Joint task offloading and resource allocation in multi-uav multi-server systems: An attention-based deep reinforcement learning approach," *IEEE Transactions on Vehicular Technology*, 2024.
[53] L. Liao, M. Tao, A. Dong, R. Xie, and Y. Zhang, "Graph-convolutional-network-enabled task offloading for industrial image recognition in digital twin edge networks," *IEEE Internet of Things Journal*, 2025.
[54] W. Fan, Y. Zhang, G. Zhou, and Y. Liu, "Deep reinforcement learning-based task offloading for vehicular edge computing with flexible rsu-rsu cooperation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 25, no. 7, pp. 7712–7725, 2024.
[55] J. Wu, Y. Zou, X. Zhang, J. Liu, W. Sun, and G. Du, "Dependency-aware task offloading strategy via heterogeneous graph neural network and deep reinforcement learning," *IEEE Internet of Things Journal*, 2025.
[56] E. W. Dijkstra, "A note on two problems in connexion with graphs," in *Edsger Wybe Dijkstra: His Life, Work, and Legacy*, 2022, pp. 287–290.
[57] S. Boyd, "Convex optimization–boyd and vandenberghe," 2004.
[58] A. Auslender and M. Teboulle, "Lagrangian duality and related multiplier methods for variational inequality problems," *SIAM Journal on Optimization*, vol. 10, no. 4, pp. 1097–1115, 2000.
[59] Z.-Q. Luo and W. Yu, "An introduction to convex optimization for communications and signal processing," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 8, pp. 1426–1438, 2006.
[60] W. H. Greub, *Linear algebra*. Springer Science & Business Media, 2012, vol. 23.
[61] J. P. Pedroso, "Optimization with gurobi and python," *INESC Porto and Universidade do Porto,, Porto, Portugal*, vol. 1, 2011.
[62] D. A. Guimaraes, G. H. F. Floriano, and L. S. Chaves, "A tutorial on the cvx system for modeling and solving convex optimization problems," *IEEE Latin America Transactions*, vol. 13, no. 5, pp. 1228–1257, 2015.
[63] Q. Hu, Y. Cai, G. Yu, Z. Qin, M. Zhao, and G. Y. Li, "Joint offloading and trajectory design for uav-enabled mobile edge computing systems," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1879–1892, 2019.
[64] S. Mirjalili and S. Mirjalili, "Genetic algorithm," *Evolutionary algorithms and neural networks: Theory and applications*, pp. 43–55, 2019.
[65] K. Deep and H. Mebrahtu, "New variations of order crossover for travelling salesman problem," *International Journal of Combinatorial Optimization Problems and Informatics*, vol. 2, no. 1, pp. 2–13, 2011.
[66] P. ERDdS and A. R&wi, "On random graphs i," *Publ. math. debrecen*, vol. 6, no. 290-297, p. 18, 1959.
[67] W. Gao, "Opportunistic peer-to-peer mobile cloud computing at the tactical edge," in *Proceedings of IEEE Military Communications Conference*. IEEE, 2014, pp. 1614–1620.

**Ke Ma** is currently a PhD candidate in the joint doctoral program of University of California, San Diego and San Diego State University. He received the B.Sc. degree in Communication Engineer from Donghua University, Shanghai, China, in 2019 and the M.S. degree in Computer Engineer from University of California at Riverside, California, in 2022.



**Junfei Xie** (S'13-M'16-SM'21) is currently an Associate Professor in the Department of Electrical and Computer Engineering at San Diego State University. She is the recipient of the NSF CAREER Award. Her current research interests include large-scale dynamic system design and control, airborne networks, airborne computing, and air traffic flow management, etc.