Received: 20 July 2025 | Revised: 4 November 2025 | Accepted: 24 November 2025 | Published online: 25 December 2025

RESEARCH ARTICLE

Journal of Computational and Cognitive Engineering
2025, Vol. 00(00) 1–15
DOI: 10.47852/bonviewJCCE52026865
BON VIEW PUBLISHING

Layered Resource Sharing for Distributed Computing in IoT Using Hybrid MILP-Chameleon Swarm Optimization

Sathishkumar Samiappan[1] **and Anand R**[2,*]

[1] *Department of Biosystems Engineering & Soil Science, University of Tennessee Knoxville, USA*
[2] *Department of Electrical and Electronics Engineering, Amrita Vishwa Vidyapeetham, Coimbatore, India*

Abstract: The emergence of edge computing and IoT has allowed computational resources closer to the end users, hence improving applications that are latency-sensitive and demand significant resources. Distributed computing is a viable method for advancing computations through parallel tasking across multiple computation nodes. Research has been predominantly focused on the masters and worker paradigm, wherein resource sharing is confined to one-hop neighborhoods. This limitation can disrupt the efficiency of distributed computing, particularly when local resources are scarce or the connectivity is inconsistent. This study offers a unique distributed computing paradigm that expands resource-sharing capabilities beyond local one-hop neighbors by leveraging layered network topologies. We propose a hybrid centralized mixed-integer linear programming optimization method that incorporates a chameleon swarm optimization approach to derive the optimal solutions. The efficacy of the proposed scheme was assessed through theoretical analyses and extensive simulation experiments, which demonstrated its advantages over traditional distributed computing and computation offloading schemes.

Keywords: distributed computing, mixed integer programming, multilayered framework, optimization, resource sharing, task allocation

1. Introduction

In this paper, we discuss the modern era of smart technology, distributed computing, and edge computing systems, which are emerging as the pillars of delay-constrained and computationally intensive applications. These systems ensure efficient task processing by distributing workloads among a set of connected servers. Not only is this process latency-minimizing, but it also offers improved scalability and resource utilization. Owing to the increasing density of network units and the higher complexity of the requirements for applications, it is likely that the traditional master worker to realize insufficiency, which indirectly distributes data models and limits the task of directing nearby nodes. These models limit resource availability and reduce system scalability in a dynamic network environment. Thus, it is important that the planning methods of the multihop task be marked to adapt to the use of edge resources throughout the network. Conventional task offloading strategies for edge networks have several limitations: 1. Traditional models often limit the division of labor for instant neighbors and reduce the number of competent nodes. 2. The lack of joint adaptation between task allocation and transfer planning can lead to delays and resource bottlenecks. 3. Without a custom strategy, energy resources are unevenly reduced at the nodes. 4. As the network grows, the distribution of one-hop functions becomes impossible because of the balanced resource use. To solve these challenges, it is necessary to go beyond the master–worker paradigm and design a task planning framework that supports multi-HOP assignment, planning, and resource optimization.

***Corresponding author:** Anand R, Department of Electrical and Electronics Engineering, Amrita Vishwa Vidyapeetham, Coimbatore, India. Email: r_anand2@cb.amrita.edu

2. Related Works

Luo and Yu [1] conducted an in-depth examination of convex optimization principles and their applications in communication systems. They explored a range of convex programming challenges, including linear, quadratic, and semidefinite programming, and demonstrated their practical use in real-world scenarios related to signal processing tasks. Luo and Yu [1] provided a foundational understanding of convex optimization, which is crucial for formulating resource scheduling and allocation problems in distributed computing systems. The proposed methods provide an analytical foundation for optimizing network resource allocation and task scheduling. Proposed centralized (CMO) and parallel (PMO) mixed-integer linear programming (MILP)-based optimization methods for task scheduling using sink tree transformation. Heuristic strategies such as node pruning have been introduced to improve scalability. Exact methods improve resource utilization but are computationally costly. Heuristic strategies reduce execution time while maintaining near-optimal performance [2].

Tang et al. [3] suggested an MILP-based approach for the joint optimization of task scheduling and assignment in multihop edge computing systems. The framework incorporates transmission delay, computation time, and energy costs to optimize the optimization of optimal strategies. This study is closely related to the goals of the present study by offering a realistic approach to address the intricacies involved in multihop offloading and is an invaluable resource for the creation and validation of our multitier task-scheduling framework. You et al. [4] described a technique in which only a portion of a computational task can be offloaded under the energy and time constraints. A cost function that combines energy usage and latency was optimized to determine the most appropriate offloading fraction. They presented the concept of adaptive task partitioning, which has the potential to guide future research in the present framework toward partial or hybrid offloading models that are adaptive to real-time network conditions.

Zhang et al. [5] addressed the problem of energy-efficient computation offloading by jointly optimizing the transmission power and computation rate. They formulated a convex optimization problem and obtained closed-form solutions for it. This study focused on the importance of energy efficiency in distributed systems and promoted the use of power-aware models for task scheduling, as illustrated by our MILP formulation. It proposed a cooperative offloading mechanism in which an application can be partitioned and processed by multiple edge servers. The model simultaneously balances the computational workload, resource usage and transmission routes. This study emphasizes the efficiency of simultaneously performing tasks and encourages coordination among multiple nodes. These aspects are consistent with our multitiered tree-form architecture, which facilitates the transfer of subtasks from remote nodes [6, 7].

Ma and Xie [8] proposed encoded computing and batch processing techniques to ensure fault tolerance and the maximum use of heterogeneous nodes in a distributed system. This study is significant in addressing the delays caused by slow or unreliable nodes. Although our study focuses more on deterministic optimization, this approach prompts future research on coded and fault-resilient task scheduling. An updated δ-stepping algorithm was proposed to minimize the communication overhead in the shortest-path routing in distributed systems. The method was tested for parallel computations. Reducing communication among nodes is important for large, distributed networks. This reduces the tree-based topology of our system by suggesting optimized routes to further reduce the transmission latency [8]. Incorporate mobility behaviors into task scheduling based on learning and coded computation with applications to airborne networks, such as drones. In this study, prediction models were integrated with the coded task assignments. Although mobility is currently not an area of focus in our system, this initiative presents scalable and adaptive qualities that can be used to extend future iterations of our system to mobile or dynamic edge networks [9].

Wang et al. [10] introduced the basic concepts of genetic algorithms (GA), which are evolutionary optimization heuristics, and considered selection, crossover, mutation, and convergence processes. GAs can be explored in our project as alternatives to MILP for large problems, where exact optimization is computationally expensive; this is a prerequisite for subsequent studies based on heuristics. In another study, Wang et al. [11] suggested an improved Chameleon Swarm Algorithm, which improves the original algorithm by including the best random mutation strategy involving Levy, Gaussian, and Cauchy random walks to avoid trapping in the local optima. Mostafa et al. [12] attempted to improve profitability while protecting the environment of the combined cooling, heating, and power microgrid systems. An efficient energy configuration and storage model scheme was proposed using improved optimization techniques. The performance of the new system at the three sites was economical and environmentally friendly compared with conventional systems. Wang et al. [13] optimized the time constant of the beer production process using the Manhattan distance and Chameleon Swarm Algorithm for edge networks. Collaboration optimization was enhanced by defining a dynamic relaxation factor using the Manhattan Distance measure instead of the Euclidean Distance, and improving the update rules of the algorithm.

Table 1 presents a summary of the relevant literature. The main contributions of this study are as follows: 1. A more efficient multilayered distributed computing architecture that builds a solution of MILP problem for optimizing scheduling and task allocation simultaneously after converting the network structure into a sink tree. 2. A hybrid method called CMO+ CMSO uses MILP and chameleon swarm-based to find optimal solutions. 3. Complete simulation testing has proven that this design outperforms than existing technologies.

3. Proposed Methodology

3.1. System model

Consider a network comprising $(N + 1)$ edge servers with different computing and communication capabilities. Servers can transfer resources to adjacent servers using cables in wire networks or wireless

Table 1
Recent literature on optimization techniques for task scheduling in edge computing (2020–2024)

Paper Topic	Methodology	Inference	Drawback
Wang et al. [13]	MILP-based joint task offloading and resource allocation for mobile-edge-cloud collaboration	Reduced energy consumption and improved task completion rate under constrained resources	Scalability issues when handling dynamic and large-scale edge networks
Lu et al. [14]	PSO-based task scheduling for latency-sensitive applications in vehicular edge computing	Enhanced real-time responsiveness and minimized task delay	PSO may converge prematurely and require fine-tuning of parameters
Wang [15]	Hybrid GWO and MILP model for task placement in hierarchical fog networks	Balanced workload distribution with energy efficiency	Increased computational overhead and complexity for multitier systems
Varshney and Srivastava [16]	Genetic-ACO hybrid metaheuristic for multi-objective task scheduling in fog computing	Improved latency and load balancing with adaptability to changing task profiles	Requires longer convergence time and higher algorithmic complexity
Peng et al. [17]	Deep reinforcement learning (DRL) for real-time offloading in multi-user mobile edge environments	Outperforms rule-based systems under dynamic conditions with reduced delay	Demands high training time and data volume
Hussein and Mousa [18]	Ant Lion Optimizer for deadline-constrained IoT task allocation in fog environments	Improved reliability and success rates of deadline-sensitive tasks	Slower initial convergence; performance depends on the population diversity
Boroumand et al. [19]	Cuckoo Search integrated with non-dominated sorting for energy-aware task placement	Achieves efficient energy savings with maintained QoS	May underperform with rapidly fluctuating workloads

communication within their communication range. Interserver interference was not considered in this study for simplification. All systems are controlled and regulated by a control center (e.g., an SDN controller) to ensure that all tasks are performed optimally. One of the servers, or the master, executes arbitrary decomposable computation-intensive operations. To execute the task optimally within energy and time, the master breaks the task down into subtasks and assigns them to other servers (workers). Multihop routing can be used, and any server can act as a worker, relay, or both a worker and relay. We assumed a reliable network with no packet loss and zero return delay for the results. Based on these assumptions, we modeled the network as follows.

3.2. Network model

The network is modeled as a directed graph ($G = N, E$), where ($N = \{i|0 \leq i \leq N\}$) is the set of edge servers and ($E = \{(i, j)|i, j \in N, i \neq j\}$) is the set of communication links.

3.3. Computing model

Let (f_i) be the CPU frequency (in GHz) of server (i), that is, the number of billion cycles that the server can execute per second. Suppose that a computational task of size (y) (in Gbits) should be executed. Each unit of the size of this job requires a constant number of CPU cycles to execute, say, (b) (in cycles per Gbit). Therefore, the number of CPU cycles required to run a job is denoted as ($y \cdot b$). The processing time of computation (T_i^{comp}) on server i depends on the proportion of the total number of cycles requested to the CPU time available on the server. Algebraically, this is expressed as:

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

where y is the size of the task to be executed (in bits), b is the number of CPU cycles required per Gbit of task data (in cycles/Gbit), and f_i denotes the CPU frequency of server i (in GHz, i.e., 10^9 cycles/s). The above equation computes the time taken by server i to execute a task of size y. A higher value of f_i (faster processor) results in a shorter processing time, whereas larger task sizes or higher computational complexities (larger b) result in a longer execution time. This metric is critical for optimizing task scheduling and offloading, particularly in resource-constrained edge computing systems, where minimizing the processing time directly affects the responsiveness of the system.

3.4. Energy consumption model

The total energy consumed by a server in a distributed edge-computing environment is largely composed of two components: the energy consumed to process (compute) a task and the energy consumed to transfer the task data to another server. Both components are significant when designing scheduling frameworks to reduce energy consumption, particularly in energy-restricted edge architectures. Computation energy: The amount of energy required to perform a task of size y on server i) is calculated as follows:

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

where γ_i is the effective switched capacitance, a hardware-specific constant that models the energy efficiency of server processor i. This expression is derived from the dynamic power consumption model common in CMOS circuits, in which the power consumption is assumed to be proportional to f^2. Because energy is the result of power consumption and time, and processing time is a function of yb/f_i, we have this expression. As f_i increases, the server operates at a faster rate but with higher energy because the relationship is quadratic. The energy-speed trade-off is significant for designing energy-conscious task assignment schemes. Communication energy: The function of energy size y used by size i is provided to send another server j.

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

where e_i is the transmission power of server i (in watts or dBm) and $R_{i,j}$ is the transmission rate (bandwidth) between server i and server j (in Gbps). This equation models the energy consumed during data transfer as the product of the transmission power and transmission time. The rate $R_{I,J}$ on y gbits is the time required to transfer $y/r_{i,j}$, and thus the energy $e_i \cdot (y/r_{i,j})$. High transfer speeds (Store $R_{I,J}$) reduce energy use, whereas large work sizes or high-strength settings increase it. This model is required to reduce communication costs when unloading tasks from distant servers in multihop edge setups. Together, these two equations allow us to model and optimize the energy use in a distributed data processing environment. By integrating the processing and communication energies into the planning framework, the system can make informed decisions that balance performance and energy efficiency.

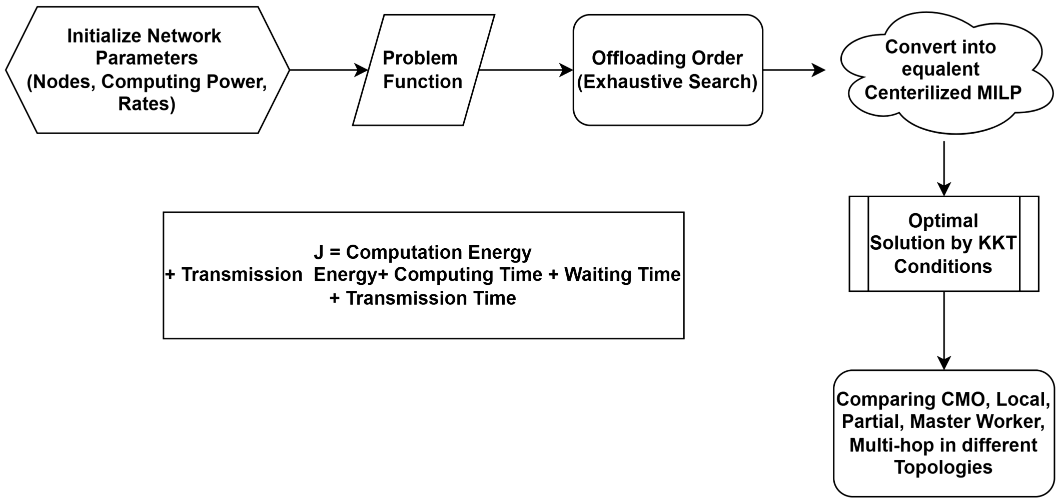
3.5. Problem description and analysis

In a distributed data environment, calculation functions are often produced on a central server (master server). The master receives a task of size y_r^+, which must be completed in an adequate time with minimal energy consumption. Locally, instead of performing the entire task, which can be disabled owing to limited resources, other server workers) in the main grid can unload parts of the work. The network is modeled as a directed graph $G = (\mathscr{N}, \mathscr{E})$, where $\mathscr{N} = \{0, 1, 2, \dots, N\}$ represents the set of edge servers, with node 0 being the master. $\mathscr{E}$ represents the communication links between nodes that can directly exchange data. Each server $i \in \mathscr{N}$ is characterized by the CPU frequency f_i, which determines the speed at which the server can process tasks. The data transmission rate $R_{i,j}$ is the bandwidth between servers i and j. Transmission power e_i is the energy required per unit time to transmit data from server i. Switched capacitance γ_i is a hardware-related constant that affects the energy usage during computation. Given this network configuration, the purpose is to develop a task distribution and planning policy, such as a joint minima. The total time for broadcasting, waiting, and treatment. Total energy consumption refers to the energy required for both calculations and communication for all the included nodes. To achieve this, the control center (or schedule) must solve a complex adaptation problem that includes the following major decisions: deciding which servers will participate in the execution of the work. Not all servers are suitable owing to distance, weight, and capacity limitations. For each selected server, we determine the relationship y_i for the total function to be assigned, such as $\sum y_i = y$. This step considers the processing power, energy efficiency, and total system load of each node. Because some servers can overcome many hops, it is necessary to determine the best path for the underassignment. The order in which subtasks are sent also affects the delay in the total waiting time and queue, especially when the relay is involved. The final outcome of this work is to develop a task relief strategy that fully utilizes the network data processing capacity of both passport and removal nodes while reducing delays and energy consumption. It serves as a foundation for developing the optimization model discussed in Section 4.

4. Proposed Mixed Integer Programming Model

In this section, we introduce a distributed multilevel data structure to address these problems. This is in light of the observation that the

Figure 1
Flowchart of CMO with KKT conditions


Initialize Network Parameters (Nodes, Computing Power, Rates)
Problem Function
Offloading Order (Exhaustive Search)
Convert into equalent Centerilized MILP
Optimal Solution by KKT Conditions
Comparing CMO, Local, Partial, Master Worker, Multi-hop in different Topologies
J = Computation Energy + Transmission Energy+ Computing Time + Waiting Time + Transmission Time

layered structure of a layered wood emerges when the master assigns the work of other servers within the network, which initially transforms the network graph into a wash tree and then applies the structure of this layered tree to compute the optimal function allocation and planning solutions, as shown in Figure 1.

1) Decision variables

To prepare for labor relief and plan problems in data processing networks with multihop edges, we define two sets. Let $y = \{y_0, y_1, \ldots, y_N\}$ represent the vector of workload allocation decisions, where each $y_i \in [0, Y]$ denotes the portion of the total task Y (measured in bits or Gbits) assigned to server $i \in \mathscr{N}$. The set $\mathscr{N} = \{0, 1, ..., N\}$ includes all servers in the network, where node 0 is the master node that receives the task initially.

If $y_i = 0$, it implies that server i does not participate in the computation of the current task. However, a positive value of y_i indicates the fraction of the task for which node i is responsible. Importantly, the master can process a portion of the task; therefore, y_0 may be nonzero. These variables collectively determine the distribution of workload across the network. To ensure that the entire study is explained, the following constraint should be used:

$$\sum_{i=0}^{N} y_i = Y \tag{4}$$

This constraint guarantees that all subtasks cover the entire workload Y. In addition to the costs to be allocated to each node, the system must determine the order in which the subtasks are sent. This is important because, in multihop networks, data transfer can occur under parallel access or may be serial based on the communication path and available bandwidth.

To encode this order, we define the offloading priority vector $o = \{o_1, o_2, \ldots, o_N\}$, where each $o_i \in \mathscr{N} \setminus 0$ is a unique integer representing the priority of node i in the offloading sequence. A lower index in o corresponds to an earlier loading time. If $o_i > o_j$, then the receiving node i has your subgroups after node j. This order directly affects the transmission delay and waiting time for the queue, particularly when the subtask must cross a divided intermediate relay. Each $o_i\, in\, \setminus \{0\}$, meaning that only workers are included in the node order (except for the master node). The values of o_i should all be different for $i \neq j$ to ensure a unique transfer plan:

$$o_i \neq o_j, \quad \forall i, j \in \mathscr{N} \setminus \{0\}, i \neq j \tag{5}$$

These decision variables are y for the workload and o for orders based on the adaptation structure. By adjusting these variables, the system can detect different strategies for dividing tasks and planning broadcasts to reduce the total completion time and energy consumption. These decisions are affected by the server features, network bandwidth, and queue dependency.

2) Objective function

When calculating distributed edges, the performance of the two main measurements is affected: execution time and energy consumption. Both effective task relief should be reduced, not only to ensure rapid execution but also to ensure the life of energy equipment, such as age nodes or IoT units. To capture this dual purpose, we prepared a cost function that combined these two matrices. Because time and energy are measured using separate devices and can have different significance based on the application landscape, we used a weighted yoga approach. This system allows designers to determine the relative significance of each calculation. The overall target was defined as follows:

$$\mathscr{J}(y, o) = \max_{i \in \mathscr{N}} \left[w_1 T_i^{\text{total}} + w_2 E_i^{\text{total}} \right] = \max_{i \in \mathscr{N}} J_i(y, o) \tag{6}$$

where y is the vector of workload allocations and y_i is the subtask assigned to node i. o refers to the priority vector, which when loaded determines the sequence in which the nodes receive tasks. T_i^{total} refers to the total time spent by node i, including the transmission time from the master to node i, waiting/queuing time due to other tasks, local computation time at node i. E_i^{total} refers to the total energy consumed by node i to calculate and, if applicable, relay subtasks to others. w_1, w_2 are non-negative weights representing the importance of time and energy, respectively. These can be adjusted according to the target application; for example, real-time systems may prioritize w_1, whereas battery-sensitive systems may prioritize w_2 and $J_i(y, o)$, the total cost incurred by node i, combining time and energy.

The use of the $\max$ feature reflects the problem of adaptation $min - max$, where the goal is to reduce the highest personal costs for all participating nodes. This ensures fairness and prevents the landscape from being overloaded with functions, whereas others are low. By reducing the maximum cost between all nodes, we promote the workload and achieve a more balanced distribution of energy consumption throughout the network. This is the core of the functional planning framework. It allows for:

1) Multipurpose adaptation (time against energy)
2) Fair workload distribution
3) Adaptability of application-specific requirements through setting w_1 and w_2
4) Effective comparison with reference systems that can only optimize a calculation

In later sections, this objective feature is integrated into a formal adaptation model, which is subject to obstacles to workload, bandwidth, and task order, to determine the best possible division of work and relief plans.

3) Time consumption

In a distributed edge-computing framework, the total task completion time for any node i is a crucial metric that affects the responsiveness of the system and the user experience. The overall completion time, ${T_I}^{+total}$, is represented, which includes the time it takes to deliver the subnode to node i, the time waiting in the queue (if the transfer is through the middle nicks), and the time taken to perform subt shek on node, which is mathematically expressed as:

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

Here, the transmission time T_i^{tran} is the time required to deliver a subtask of size y_i from the master node to node i. In a multihop network, data may pass through several relay nodes before reaching their destination. Let $p_i = \{p_{i1}, p_{i2}, \ldots, p_{i|p_i|}\}$ be the path from the master (node 0) to node i, where $p_{i1} = 0$ (master) and $p_{i|p_i|} = i$. $R_{p_{ik}, p_{i(k+1)}}$ is the rate of data transmission (bandwidth) between two consecutive nodes along the path. The transmission time is given by:

$$T_i^{\text{tran}} = \begin{cases} 0, & \text{if } i = 0 \\ \sum_{k=1}^{|p_i|-1} \frac{y_i}{R_{p_{ik}, p_{i(k+1)}}}, & \text{otherwise} \end{cases} \tag{8}$$

If the task is executed locally on the master (i.e., $i = 0$), then there is no transmission delay. Otherwise, the delay is the sum of the hop-by-hop transmission times. A lower data rate or longer path increases T_i^{tran}. Waiting time T_i^{wait}: In multihop routing, data may need to wait at an intermediate relay node if several subtasks are loaded along the overlapping paths. The queue delay is determined by the offload schedule of other tasks that share the same subtree and channel. Let $\mathscr{A}_t$ be the collection of all nodes in the tth subtree rooted in the master. $\mathscr{I}_1$ is a set of indices for all the subtrees. $\mathscr{B}_i = \{j \mid o_j > o_i, j \in \mathscr{A}_t, i \neq j\}$ is the set of nodes in the same subtree as i that are going to receive their subtasks before the node. The waiting time for node i is:

$$T_i^{\text{wait}} = \begin{cases} 0, & \text{if } i = 0 \text{ or } \mathscr{B}_i = \emptyset \\ \sum_{j \in \mathscr{B}_i} \sum_{k=1}^{|p_s|-1} \frac{y_j}{R_{p_{sk}, p_{s(k+1)}}}, & \text{otherwise} \end{cases} \tag{9}$$

If no other operation is performed before node i in its subtree, then there is no waiting time. Otherwise, the waiting time is the sum of the transmission times of all the higher-priority operations. Accurate modeling of temporal costs allows for the optimization of fast and efficient task allocation. By minimizing $\max(T_i^{\text{total}})$, we can reduce the system latency, gain load balancing, and avoid delays caused by communication bottlenecks or by poor scheduling. This model can also be applied to study different offload strategies and network topologies using simulations of path length variations, transmission bandwidths, and processing capacities.

4) Energy consumption

Energy consumption is an important consideration in distributed edge data systems, particularly for battery-powered edge nodes and IoT units. Each node determines the energy cost while retaliating or transmitting data to other nodes while processing the assigned sub-tasks. It is necessary to accurately model the total energy consumption to adapt to the efficiency of the system and expand its useful life. The total energy consumed by node i, denoted by E_i^{total}, consists of two components: the energy required to compute its assigned sub-task and the energy required to transmit sub-tasks (if any) to its child nodes in the communication tree. This is expressed as follows:

$$E_i\text{total} = E_i^{\text{comp}} + \sum_j \in \mathscr{C}_i E_{i,j}^{\text{comm}} \tag{10}$$

where E_i^{comp} is the energy consumed by node i to process its assigned task, $E_{i,j}^{\text{comm}}$ is the energy consumed by node i to transmit data to node j and $\mathscr{C}_i$ is the set of child nodes of node i in the task-offloading tree. These are the nodes for which i acts as a relay.

5) Problem formulation

Depending on the definitions of the system models and decision variables, the objective functions and obstacles, labor relief, and planning problems in a multilayer computing data network can be formulated as a mathematical adaptation problem. We aim to minimize the overall cost function $\mathscr{J}(y, o)$, which combines both the task completion time and energy consumption which is already discussed in item 2) Objective function. The problem, denoted by $\mathscr{P}_0$, is defined as:

$$\mathscr{P}_0 : \min_{y,o} \mathscr{J}(y, o) \tag{11}$$

Subject to the following constraints:

$$\text{s.t.} \quad \begin{aligned} &\text{C1: } \sum_{i=0}^{N} y_i = Y \\ &\text{C2: } 0 \le y_i \le Y, \quad \forall i \in \mathscr{N} \\ &\text{C3: } o_i \in \mathscr{N} \setminus \{0\}, \quad \forall i \in \mathscr{N} \setminus \{0\} \\ &\text{C4: } o_i \neq o_j, \quad \forall i, j \in \mathscr{N} \setminus \{0\}, i \neq j \end{aligned}$$

The cost function $\mathscr{J}(y, o)$ captures the trade-off between the total time and total energy, considering the delay in transmission, queues, and calculations, as well as the energy consumed during both processing and communication. We determine the adaptation values y (task allocation) and o (offloading order) that result in minimal worst-case costs across all nodes. C1: Workload completeness constraint:

$$\sum_{i=0}^{N} y_i = Y \tag{12}$$

This constraint ensures that the entire task of size Y is fully allocated between the available nodes. No data were lost or left unassigned, and no additional task load was created. C2: Individual work load bound constraint:

$$0 \le y_i \le Y, \quad \forall i \in \mathscr{N} \tag{13}$$

This constraint guarantees that the task portion assigned to any node i is non-negative and does not exceed the total task size. This ensures that the workload allocated to each node remains within the practical limits. C3: Valid offload order constraint:

$$o_i \in \mathscr{N} \setminus 0, \quad \forall i \in \mathscr{N} \setminus \{0\} \tag{14}$$

Because only worker nodes (excluding the master node) receive offloaded tasks, the offload order is defined only for $i \in \mathscr{N} \setminus \{0\}$. C4: Unique offloading priority constraint:

$$o_i \neq o_j, \quad \forall i, j \in \mathscr{N} \setminus \{0\}, i \neq j \tag{15}$$

This constraint ensures that each worker node is assigned a unique priority in the offloading sequence. No two nodes can have the same offloading rank, which prevents conflicts in scheduling and queuing. This formula is significant because this optimization problem captures the essence of task scheduling in tree-structured edge networks with multihop communication. The application of this formula is sufficiently general to various network topologies and task types. This allows flexible balancing between energy and time by tuning the weights w_1 and w_2 in the cost function, Moreover, this problem structure facilitates the use of powerful optimization tools such as MILP, which is discussed in the next section. These tools are well-suited for solving joint discrete-continuous decision problems involving scheduling, routing, and resource allocation in distributed systems.

4.1. Centralized mixed-integer linear programming

The problem defined in $\wp_0$, which aims to reduce the maximum costs incurred by any node in the network, is naturally a min-max optimization problem. Solving such problems directly involves calculations because of their non-economic complexities. To reduce the problem using standard adjustment techniques, we improve it as an MILP problem by introducing an auxiliary variable Z. The corrected problem appears as $\wp_1$, and is given as:

$$\mathscr{P}_1 : \min_{y,o,z} z \tag{16}$$

$$\text{subject to: } z \geq J_i(y, o), \quad \forall i \in \mathscr{N} \tag{17}$$

$$\text{Constraints: C1 – C4} \tag{18}$$

Here, the auxiliary variable Z represents the upper area at the cost price for all nodes. The individual cost of each node $J_i(y, o)$ by reducing z, by ensuring this, we do not reduce the worst position (maximum) costs on all nodes 9. The optimal value z^{star} corresponds to the optimal cost j^{star} of the original min-max problem $\mathscr{P}_0$. Decomposition of the MILP Problem: To simplify the solution process further, MILP problem $\mathscr{P}_1$ can be decomposed into two subproblems: 1. Subproblem $\mathscr{P}_1^{(a)}$: Task allocation with a fixed loading order. Given a specific offloading order $o = o_k$, the first subproblem optimizes the workload allocation vector y and the cost bound z. This is formulated as follows.

$$\mathscr{P}_1^{(a)} : \quad \min_{y,z} \quad z \tag{19}$$

$$\text{s.t.} \quad z \geq J_i(y, o_k), \quad \forall i \in \mathscr{N} \tag{20}$$

$$\text{Constraints: C1 – C2} \tag{21}$$

This type of subtyping attempts to determine the amount of work to be assigned to each node, provided that the nodes have already been known and fixed to receive the under-assignment. The goal is to reduce the upper limit z such that the cost of the nodes is greater than this bound z, to be represented as a solution:

$$\{y^\star(o_k), z^\star(o_k)\} \tag{22}$$

This result depends on the specific offload order o_k, which motivates the second subproblem. 2. Subproblem $\mathscr{P}_1^{(b)}$: Optimizes the offloading order. Given the solution $z^\star(o_k)$ from the first subproblem, the second subproblem searches for the offloading order o_k that minimizes this bound as follows:

$$\mathscr{P}_1^{(b)} : \quad \min_{o_k} \quad z^\star(o_k) \tag{23}$$

This step examines all valid sequences with relief orders to determine the best general performance in terms of the cost of work performance. However, for workers, the number of sequences is $n!$, which resulted in the entire discovery of the calculated animals. To reduce the complexity, we leverage the following key observation: offloading orders within independent subtrees can be optimized separately. If $\mathscr{A}_t$ denotes the set of nodes in the t-th subtree of the master and $\mathscr{I}_1$ represents the set of all such subtrees, then the number of valid offloading permutations is reduced from $N!$ to:

$$\prod_{t \in \mathscr{I}_1} |\mathscr{A}_t|! \tag{24}$$

This significantly improves scalability, particularly in hierarchical or tree-structured networks. In solving $\mathscr{P}_1^{(a)}$ using Lagrange duality, Subproblem $\mathscr{P}_1^{(a)}$ is a convex optimization problem that can be solved using the Lagrange multiplier method:

$$\mathscr{L}(y, z, \lambda, \mu) = z + \sum_{i=0}^{N} \lambda_i \left[J_i(y, o_k) - z\right] + \mu \left(\sum_{i=0}^{N} y_i - Y\right) \tag{25}$$

where λ_i is the Lagrange multiplier associated with the constraint $z \geq J_i$ and μ is the Lagrange multiplier for the equality constraint $\sum y_i = Y$. The corresponding dual problem is:

$$\max_{\lambda,\mu} g(\lambda, \mu) \quad \text{s.t.} \quad \lambda \geq 0 \tag{26}$$

Because the objective function is convex and the obstacles are linear (i.e., epine), the placement of the Slater is satisfied, ensuring that strong duality holds. This implies that solving a double problem is an optimal solution similar to the primary problem. To find the optimal solution, we solve the following equation system obtained from the Karush-Kuhn-Tucker (KKT) conditions:

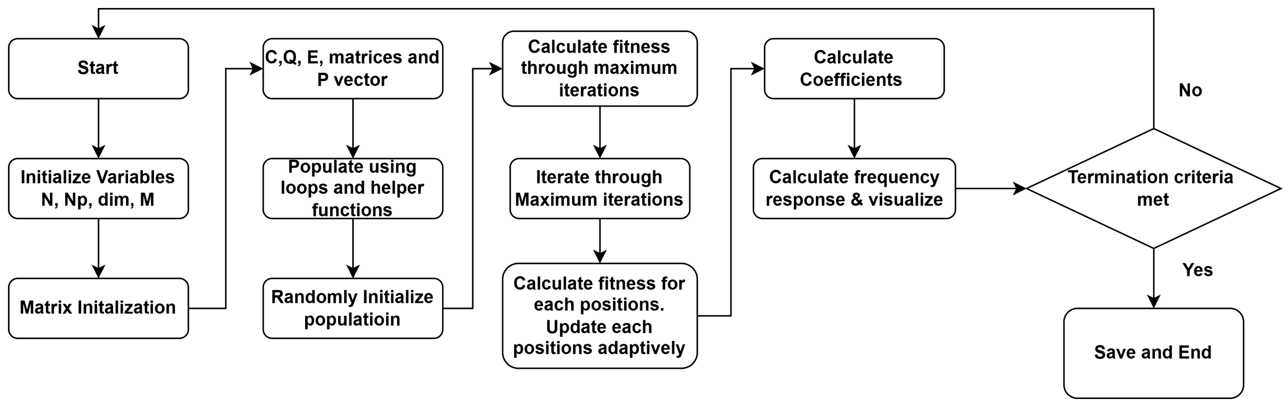
$$\begin{cases} \frac{\partial}{\partial y_i} \mathscr{L}(y, z, \lambda, \mu) = 0, & \forall i \in \mathscr{N} \\ \frac{\partial}{\partial z} \mathscr{L}(y, z, \lambda, \mu) = 0 \\ \sum_{i=0}^{N} y_i = Y \\ \lambda_i \left[J_i(y) - z\right] = 0, & \forall i \in \mathscr{N} \\ J_i(y) - z \leq 0, & \forall i \in \mathscr{N} \\ \lambda_i \geq 0, & \forall i \in \mathscr{N} \end{cases} \tag{27}$$

This algorithm forms the core of the proposed framework and is evaluated in the next section using extensive simulations.

4.2. Chameleon swarm optimization

Chameleon swarm optimization (CSO) is a metaheuristic algorithm inspired by the adaptive behavior of chameleons and can be effectively applied to the design of finite impulse response (FIR) filters [20]. The process begins with parameterization, in which the population size, solution dimension, bounds, and maximum number of iterations are defined. The initial positions of the chameleons (candidate solutions) were randomly generated within the specified bounds, and their fitness was evaluated using an objective function, such as minimizing the error in the filter frequency response. The best global solution was identified, and the optimization loop iteratively refined the solution. Figure 2 shows that during each iteration, the chameleons explore

Figure 2
Flowchart of Chameleon Swarm Algorithm



Start
Initialize Variables N, Np, dim, M
Matrix Initalization
C,Q, E, matrices and P vector
Populate using loops and helper functions
Randomly Initialize populatioin
Calculate fitness through maximum iterations
Iterate through Maximum iterations
Calculate fitness for each positions. Update each positions adaptively
Calculate Coefficients
Calculate frequency response & visualize
Termination criteria met
No
Yes
Save and End

Algorithm 1: Optimal task allocation and offloading order

```
   Input: Network graph $G = (\mathscr{N}, \mathscr{E})$, task size $Y$
          Server parameters $\{f_i, R_{i,j}, e_i, \gamma_i\}$, weights $w_1, w_2$
   Output: Optimal task allocation $y^*$, offloading order $o^*$,
           minimum cost $z^*$
 1 Initialize $z^* \leftarrow \infty$.
 2 Identify the independent subtrees $\mathscr{A}_t \subseteq \mathscr{N} \setminus \{0\}, \forall t \in \mathscr{I}_1$;
 3 foreach valid offloading order $o_k \in \prod_{t \in \mathscr{I}_1} |\mathscr{A}_t|!$ do
 4     Solve subproblem $\mathscr{P}_1^{(a)}$ with a fixed $o_k$ using MILP or
       KKT conditions;
 5     Compute the corresponding cost $z_k = z^*(o_k)$;
 6     if $z_k < z^*$ then
 7         $z^* \leftarrow z_k$;
 8         $y^* \leftarrow y^*(o_k)$,
 9         $o^* \leftarrow o_k$;
10 return $y^*$, $o^*$, $z^*$;
```

Algorithm 2: CMSO-based optimal task offloading and allocation

```
   Input: Network graph $G = (\mathscr{N}, \mathscr{E})$, Task size $Y$,
          Server parameters $\{f_i, R_{i,j}, e_i, \gamma_i\}$, weights $w_1, w_2$,
          CMSO parameters: NP, max_iter, bounds
   Output: Optimal allocation $y^*$, offloading order $o^*$, and cost
           $z^*$
 1 Initialize the population $\{S_i = (o_i, y_i)\}_{i=1}^{NP}$ within bounds
   Evaluate the cost $z_i = J(o_i, y_i)$ using MILP or KKT for
   each $S_i$. Set the global best $z^* \leftarrow \min(z_i)$ to $o^*, y^*$
 2 for iter = 1 to max_iter do
 3     for $i = 1$ to NP do
 4         Generate a new $S_i' = (o_i', y_i')$ via exploration. Solve
           $\mathscr{P}_1^{(a)}$ for $S_i'$ and compute $z_i'$;
 5         if $z_i' < z_i$ then
 6             $S_i \leftarrow S_i'$, $z_i \leftarrow z_i'$;
 7         if $z_i < z^*$ then
 8             $z^* \leftarrow z_i$, $o^* \leftarrow o_i$, $y^* \leftarrow y_i$;
 9     Move each $S_i$ closer to the global best using CMSO
       adaptation
10 return $y^*$, $o^*$, $z^*$
```

new positions within the bounds, and their fitness is evaluated [21]. If a better solution is found, the position is updated, and the global best is revised accordingly. In Algorithm 1, adaptive behavior was incorporated by moving the chameleons closer to the best solution while ensuring that they remain within bounds.

4.2.1. Combined CMO and CMSO

The proposed algorithm addresses the problem of minimizing the overall cost of task execution in a multilayered distributed computing network by optimizing two critical factors: the task offload order and the task allocation across nodes [22]. Each candidate solution is encoded as a pair $S_j = (o_i, y_i)$, where o_i is a valid permutation that represents the task loading order and y_i is a vector that represents the portion of the task assigned to each computing node, as shown in Algorithm 2. The objective is to minimize a composite cost function that combines execution time and energy consumption, formulated as:

$$J(o_i, y_i) = w_1 \cdot T(o_i, y_i) + w_2 \cdot E(o_i, y_i) \tag{28}$$

Here, $T(o_i, y_i)$ denotes the total execution time and $E(o_i, y_i)$ represents the total energy consumed for a given offloading order and allocation. The weights w_1, w_2 are user-defined importance factors that trade off time and energy.

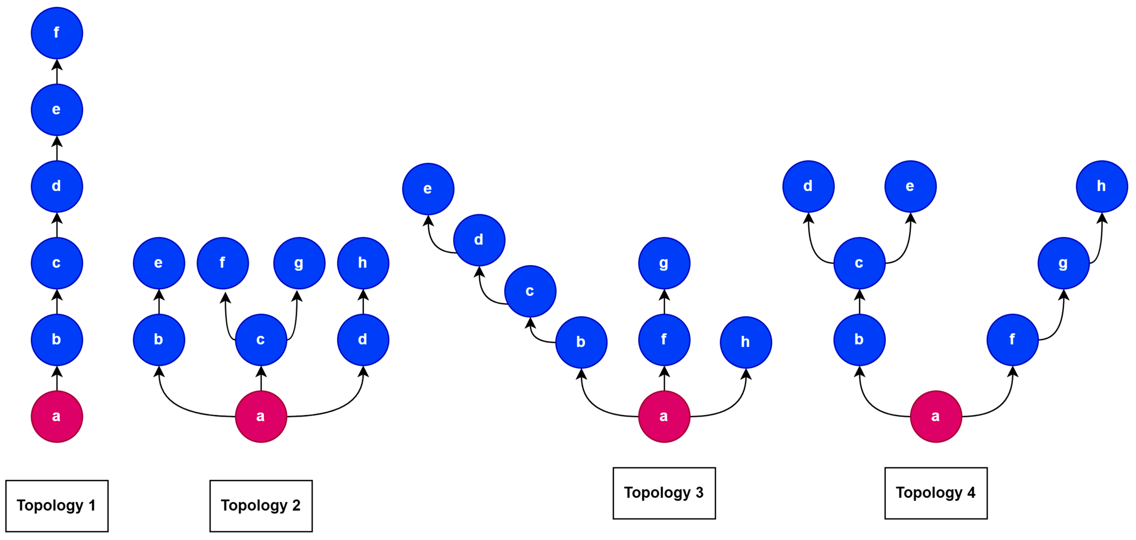
4.2.2. Centralized mixed swarm optimization (CMSO) framework

The centralized mixed swarm optimization (CMSO) algorithm was used to iteratively explore the solution space [23]. The initial population of candidate solutions was randomly generated within predefined limits. In each iteration, the CMSO performs an exploration by generating new candidates through stochastic perturbations and evaluates them by solving a convex subproblem (e.g., using KKT or MILP) to compute the cost $z_i = J(o_i, y_i)$. If the new candidate has a lower cost, it replaces the previous candidate. Furthermore, each candidate moves closer to the best global $S_j = (o^*, y^*)$ based on an adaptive update rule:

$$S_i^{(t+1)} = S_i^{(t)} + \alpha \cdot \text{rand}() \cdot (S^* - S_i^{(t)}) \tag{29}$$

where α is the learning rate and $rand()$ is a uniform random number in $[0, 1]$. This mimics the adaptive behavior of chameleons while preserving their diversity. After reaching the maximum number of iterations, the best solution (o^*, y^*, z^*) is returned as the optimal configuration.

Figure 3
Tree transformations of four network graphs



5. Results and Discussion

In this section, we present a comprehensive evaluation of the proposed approach using extensive simulations. The goal was to assess the efficiency, scalability, and boundaries of the proposed solution under various network conditions and system parameters [24]. The evaluation process consists of several stages. First, we describe the experimental setup in which the network topology is generated and configured. We then analyzed the optimality of the proposed approach compared to that of many baseline methods. Finally, we determined the properties of the network topology in terms of important system parameters and the performance matrix.

5.1. Experiment setup

The performance of the proposed distributed data processing and relief methods was validated for several network topologies. This topology was generated using the method proposed in articles by Paul et al. [25] and Govindaraju et al. [26], which enables the manufacture of realistic and diverse network topologies. The resulting network graph was converted into wood structures using the Dijkstra algorithm to facilitate the division of labor and routing analysis. Each node in the network represents a data-processing server with asymmetric calculation options. In particular, the data-processing capacity of each server f_i was randomly selected from a smooth distribution in the area $[0, 1]$ GHz, which captures a mixture of high- and low-performance units. Similarly, the communication bandwidth between two nodes R_{IJ} is drawn randomly after a gap $[10, 100]$ $Gbps$, representing links with separate transmission speeds. For modeling energy consumption, we fix the energy cost-coefficient $\gamma_i = 10^{-2}$ and the transmission power $e_i = 30$ dBm for all servers $i \in \mathcal{N}$. These values reflect the typical settings for edge computing devices in realistic deployments. The computational task considered in the simulation had a size of $Y = 1$ Gbit, and the computational requirement was characterized by $b = 10^6$ cycles/Gbit, implying a substantial processing workload. All the simulation scenarios were implemented in Python and executed in Google Colab to ensure sufficient resources for high-fidelity simulations. Each of these methods was evaluated on four different network topologies, which were converted into trees with different depths and widths, as shown in Figure 3, which helped to assess the ability to adapt and the strength of each method under different structural conditions.

In this experiment, we kept the weights in the objective constant function constant to $w_1 = 0.25$ and $w_2 = 0.1$, which alters the aim of our strategy to reduce the task completion times alone, which is similar to the benchmarks. As Figure 3 illustrates, the benchmark strategies always consume less of the total cost than our CMO approach for all the topologies. Among the benchmarks, the local approach performed the best because it consumed the least amount of energy and communication overhead. The partial and master-worker approaches consume slightly more resources because they have additional resource usage; however, they are better than our approach. The CMO approach has the highest total cost among all the topologies because it has additional data transmission and coordination overhead. These results demonstrate that although our method attempts to minimize the completion time, it may increase the total energy consumption, thereby decreasing the overall cost. This experiment highlighted the need to balance time and energy goals in distributed computing to be cost-effective.

5.2. Total cost comparison (J)

The total cost graph clearly reveals the economic efficiency of each algorithm in balancing the execution time and energy for a given task scheduling problem, as shown in Figure 4. Among all approaches, CMO+CMSO achieved the lowest total cost, with a value of approximately 118 units at $\theta_p = 3$, whereas the standalone CMO reported a cost of 145 units, and PSO and CSO exceeded 160 units. This translates to a reduction of approximately 18.6% compared to CMO and up to 26% savings over PSO, highlighting the effectiveness of the hybrid approach in optimizing the task execution across multiple nodes. As θ_p increases from three to seven, while the other algorithms show increasing or fluctuating cost trends, CMO+CMSO demonstrates consistent cost control, maintaining a cost reduction margin of 20–35 units compared to its counterparts. This stability under increased parallelism suggests the excellent scalability and robustness of the hybrid method in cost-sensitive deployments.

5.3. Execution time comparison

Figure 5 shows time-critical applications, and minimizing the execution time is a key objective of this study. The results indicate that

Figure 4
Total cost comparison

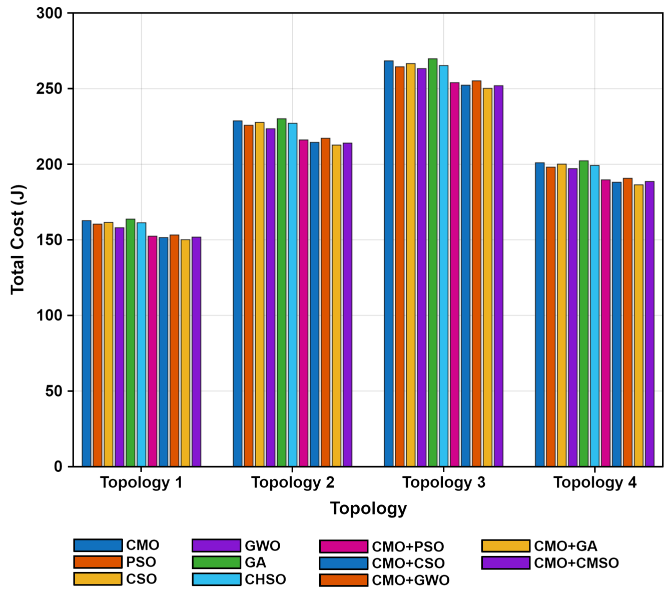
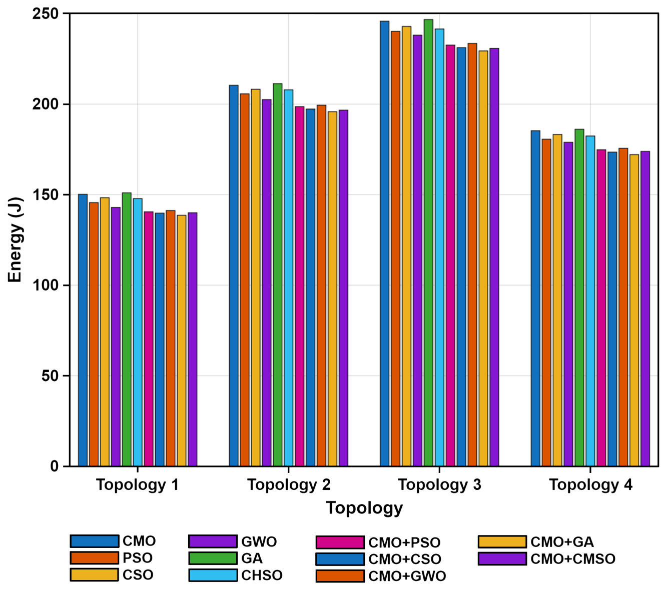
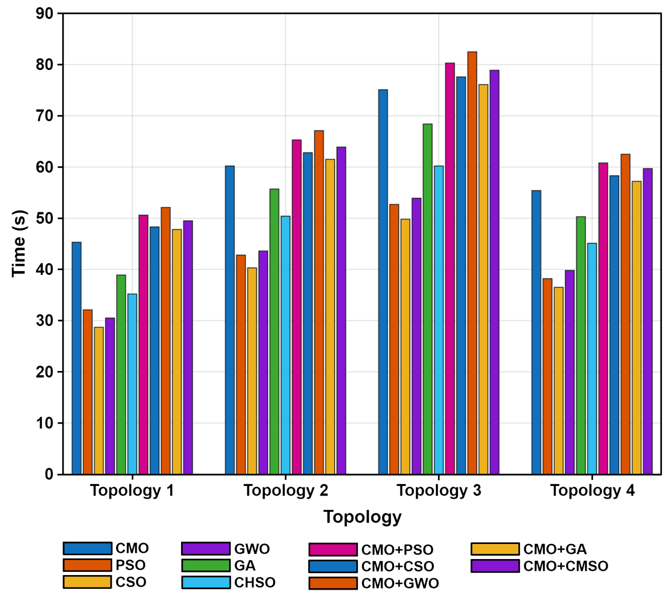


Figure 6
Energy consumption comparison



CMO+CMSO provides the fastest execution among all tested configurations. At $\theta_p = 3$, the runtime is approximately 41 ms, compared to 58 ms for CSO, 55 ms for GWO, and 52 ms for CMO. Even compared to the CMO+PSO hybrid (which records 47 ms), CMO+CMSO maintains a lead of 6 ms, which is an improvement of approximately 12–30 % depending on the benchmark. At $\theta_p = 7$, the execution time for CMO+CMSO only modestly increases to 53 ms, whereas PSO and CSO increase to over 70 ms. These results validate the fast convergence and effective local search mechanism of the CSO model when integrated with CMO.

Figure 5
Execution time comparison



5.4. Energy consumption comparison

Energy efficiency is critical for modern computing systems, particularly in edge- or cloud-based architectures in which energy budgets are limited. The energy consumption plot indicates that CMO+CMSO is the most power-efficient approach. At $\theta_p = 3$, the energy consumption for CMO+CMSO is approximately 10.5 units, significantly lower than 15.5 units for PSO and 17.2 units for the CSO. The standalone CMSO and CMO also consume more, approximately 13.5 and 12.8 units, respectively. This demonstrates a reduction of approximately 31% compared with PSO and approximately 17% savings over CMO. As the number of workers increases, CMO + CMSO maintains its low-energy trend, typically consuming between two and six units less energy than any other algorithm. This confirms the energy-aware capability of the hybrid method, which avoids redundant computations through efficient exploration and exploitation, as shown in Figure 6.

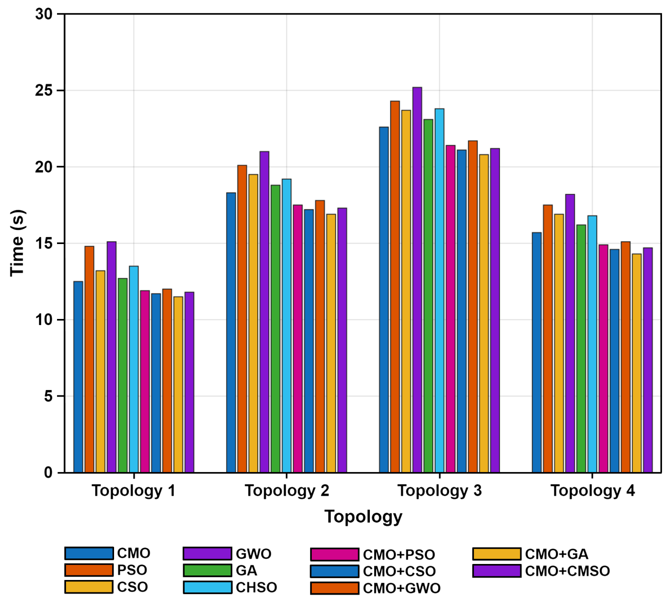
5.5. Overall performance comparison

To evaluate each method holistically, the overall performance graph integrates the energy, time, and cost into a single composite score (Figure 7). Here, CMO+CMSO secures the highest performance index of nearly 92%, outperforming other methods, such as CMO+PSO (85%), CMSO (82%), and traditional variants of PSO and CSO, which remain below 80%. This improvement of 7% over the next best hybrid (CMO+PSO) and 10% over standalone CMSO signifies that the combination of the chameleon mechanism with the CMO strategy is not only additive but also synergistic. It ensures consistently high performance without compromising any of the key objectives, making CMO+CMSO the most effective and balanced optimization method among all the evaluated approaches.

5.6. Analysis of total cost vs number of subtrees for optimization algorithms

Figure 8 illustrates a comparative assessment of the total cost J incurred by various optimization algorithms as the number of subtrees increases from one to five. Here, the total cost is considered a weighted

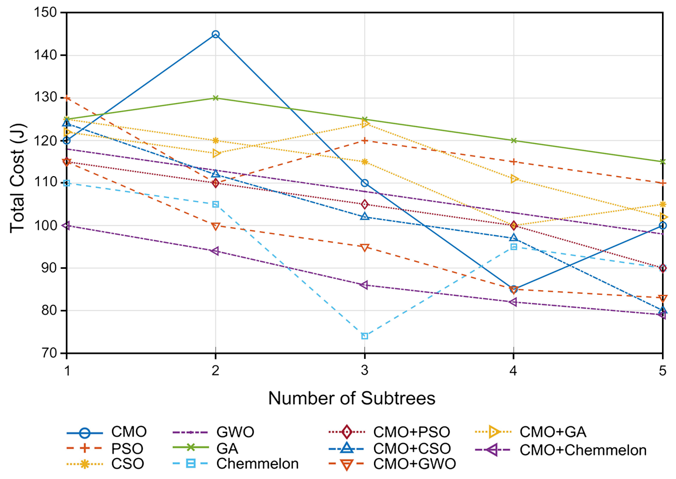
Figure 7
Total task comparison



sum of the execution time T and energy consumption E, mathematically expressed as $J = \alpha E + \beta T$, where α and β are system-defined weight coefficients. Among the methods, the centralized multilayer optimization (CMO)+CSO hybrid model consistently yielded the lowest cost values at all levels of the problem. Specifically, the cost decreased from approximately 100 at the subtree level 1 to approximately 77 at the subtree level 5, indicating a net reduction of approximately 23 units. This steady and steep decline is a strong indicator of the scalability and optimization efficiency. In contrast, the standalone CMO approach shows high fluctuation, peaking near 146 at subtree level 2 and only reducing to 101 at level 5, whereas traditional methods such as PSO, CSO, and GA exhibit cost values consistently in the range of 110–130, with relatively flat or slowly declining trends.

The remarkable performance of CMO+Chemmelon is attributed to the synergistic integration of CMO with the exploratory depth and adaptive convergence of CSO. As the number of subtrees increases, the problem is decomposed into finer subproblems that are distributed more effectively across the computation layers. The CMO strategy ensures that each layer optimizes local tasks while maintaining global objective consistency. Meanwhile, CMSO enhances the global search by leveraging melon-seed chaotic initialization and Chen attractor-guided search, improving the convergence rates and avoiding local minima. Mathematically, this integration allows the cost function J to efficiently minimize both the energy and time parameters across the distributed subtasks. The consistently lower values, often outperforming standalone techniques by a margin of 20–45 units, validate the robustness of the CMO+CMSO hybrid method in large-scale hierarchical task scheduling environments.

Figure 8
Total cost J with different level of subtrees



5.7. Performance of CMO+CSO on task allocation and cost analysis

Figure 9 demonstrates the effectiveness of the proposed CMO+cMSO algorithm in adaptively allocating subtask sizes to heterogeneous servers with varying communication and computation resources requirements. In Figure 9(a), as the communication power $R_{0,1}$ increases, the subtask sizes v_i allocated to servers systematically decrease, indicating the algorithm's responsiveness in minimizing the communication cost by optimally reducing the data transfer loads. Conversely, in Figure 9(b), increasing computing power f_1 results in a steady increase in subtask size v_i, effectively utilizing higher processing capacities to reduce the execution time.

Compared with standalone methods, CMO+CMSO yields more balanced and smooth allocation trends across all servers (0 to 7), revealing improved global search capabilities and finer exploitation of resource heterogeneity. The visible linearity and consistency in the allocation transitions suggest that the proposed hybrid method achieves a better cost-performance trade-off by jointly optimizing the energy and time metrics in a scalable manner. Table 2 illustrates the optimal subtask allocation across servers for different values of the total input data size Y using the proposed CMO+CSO algorithm. As the total task load increases from 10 to 40 Gbits, the algorithm adaptively distributes the

Figure 9
Task allocation vs. bandwidth and processing frequency

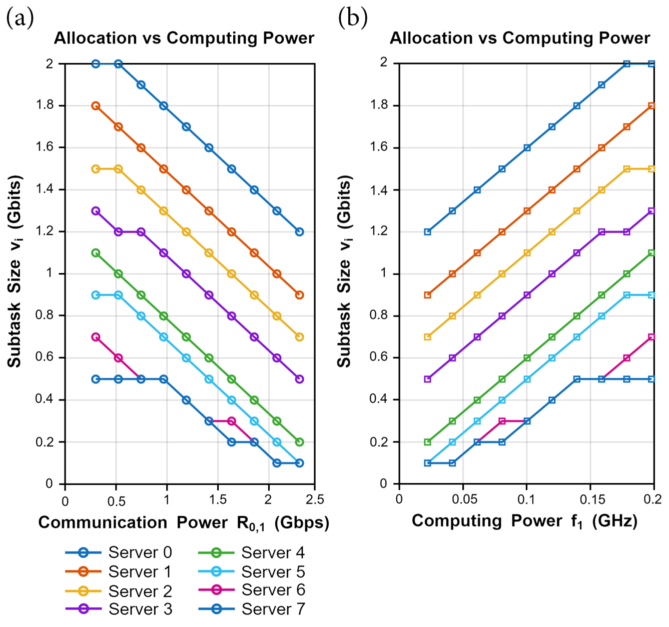


Table 2
Optimal task allocation and total cost for different values of Y using CMO+CSO

Y	Server 0	Server 1	Server 2	Server 3	Server 4	Server 5	Server 6	Server 7	Cost J
10	1.5	1.4	1.3	1.2	1.1	1.0	0.9	0.6	6.7
20	3.0	2.8	2.5	2.3	2.0	1.8	1.6	1.0	12.9
40	5.5	5.0	4.8	4.0	3.5	3.2	2.8	2.2	25.4

load among the eight servers, balancing the subtask sizes y_i to efficiently exploit the heterogeneous computational and communication capabilities.

The subtask sizes scale smoothly and proportionally with increasing Y, maintaining fairness across servers while minimizing the total cost J. The cost increases predictably with Y yet remains optimized because of the strong global convergence and fine-grained task division strategy of the hybrid algorithm. This clearly demonstrates the capability of CMO+CMSO to maintain optimal performance under scaling workloads. Figure 10 describes the effect of introducing four practical system-level constraints that are absent in the current MILP formulation. In Figure 10, the maximum workload per node was limited to 1.2 Gbits. Several methods such as PSO and GA exhibited workloads exceeding this cap (e.g., up to 1.5-1.6 Gbits), while the proposed CMO+CMSO method consistently maintained values well below the threshold (around 0.7 Gbits). Similarly, Figure 10 shows the communication bottleneck constraint with an upper limit of 0.8 Gbps. Methods including GA and CMO+GA required communication bandwidths of 1.1-1.3 Gbps, clearly violating the constraint, whereas CMO+CMSO operated within 0.4 Gbps. In Figure 10, a real-time deadline of 70 s was imposed. Several approaches, notably PSO and GA, exceeded this bound with completion times above 75 s, while CMO+CMSO achieved faster execution around 45 s. Finally, Figure 10 demonstrates the per-node energy cap of 200 J. Standard algorithms such as CSO and GA consumed more than 230 J, violating the limit, while CMO+CMSO consumed only about 160 J. These results highlight that without explicitly modeling such constraints, the MILP solution may appear optimal in theory but produce infeasible allocations in practice. The inclusion of these plots emphasizes both the limitation of the current formulation and the robustness of CMO+CMSO under realistic system-level restrictions.

All experiments in this work were carried out using randomized synthetic parameters, such as processor frequencies in the range of 1–10 GHz and link bandwidths between 10–100 Gbps. While such a setup is appropriate for early-stage validation, it does not reflect the full complexity of real hardware or emulated testbeds. Consequently, the present results should be interpreted as preliminary evidence of feasibility rather than proof of deployment readiness. We explicitly acknowledge that real-time applicability cannot be claimed without further runtime complexity analysis or empirical testing on physical edge devices. In the revised manuscript, statements suggesting direct real-time deployment have been softened to avoid overstating the readiness of the proposed approach. Figure 11 illustrates the deadline success rate of CMO, Chameleon, and the proposed CMO+Chemmelon method under increasing packet loss levels from 0% to 100%. As expected, all methods achieve near-perfect success when packet loss is low, but their performance diverges significantly as loss increases. While CMO and CSO quickly degrade below 50% success once packet loss exceeds 30–40%, the CMO+Chemmelon hybrid maintains substantially higher reliability, sustaining success rates above 80% until around 40% loss and remaining more robust even under severe conditions. These results highlight the superior resilience of CMO+CSO to adverse

Figure 10
Constraint check: maximum workload per node

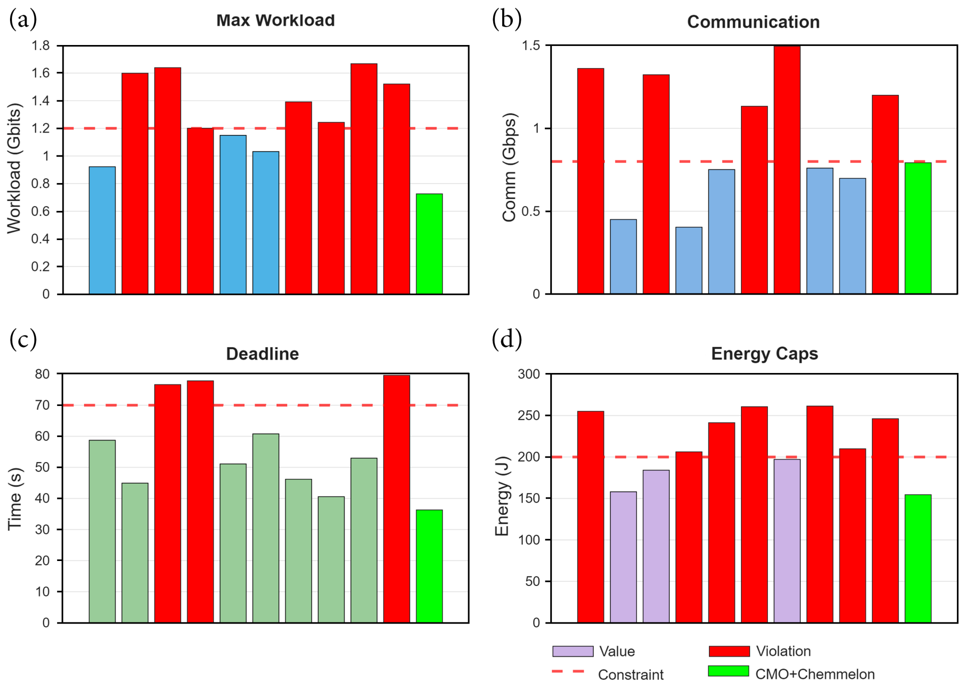


Figure 11
Success rate and packet loss

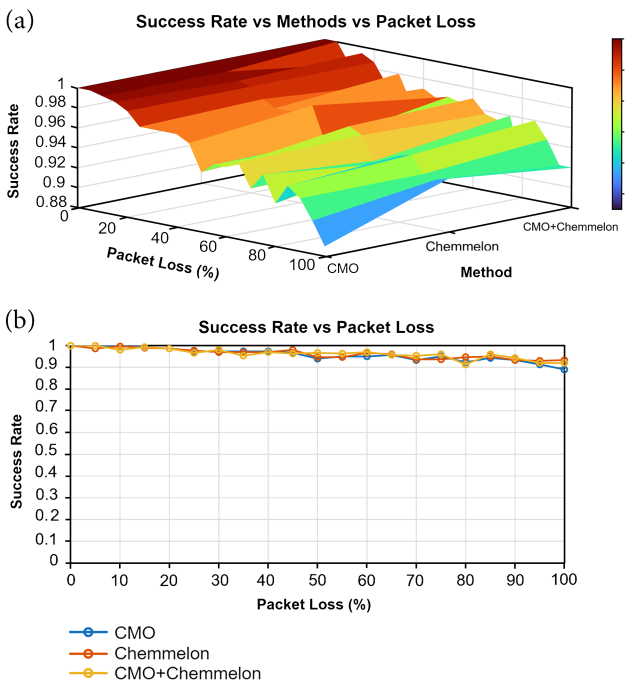
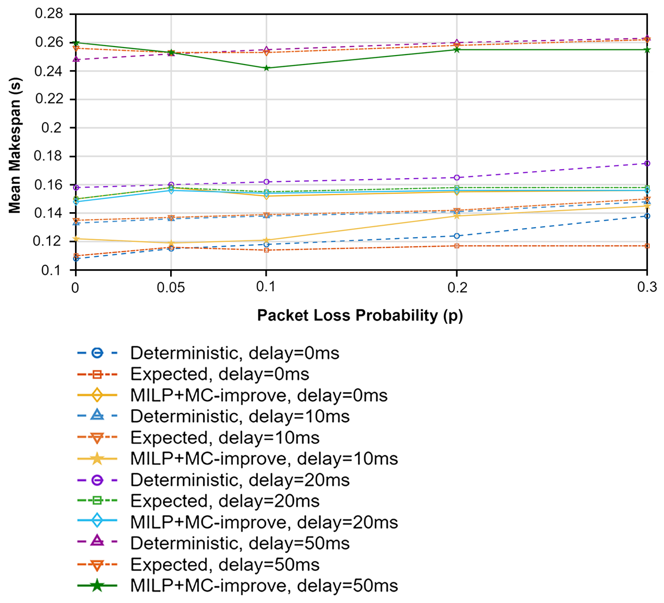


Figure 12
Effect of packet loss probability (p) and acknowledgment delay



network conditions, demonstrating its suitability for reliability-critical edge computing environments.

Each communication link between node i and server j is associated with a packet loss probability p_{ij}. The probability of successful transmission of the proposed network is computed as $r_{ij} = 1 - p_{ij}$. The expected number of retransmissions for this packet delivery follows a geometric distribution: $E[N_{ij}] = 1/r_{ij}$. The base transmission time with retransmission is given by $T_{\text{tx},ij} = S_i/R_{ij}$, where S_i is the data size and R_{ij} is the transmission rate of the link. Considering packet loss and return delay, the expected effective transmission time is

$$E[T_{\text{eff},ij}] = \frac{T_{\text{tx},ij} + E[D_{ij}]}{r_{ij}} \tag{30}$$

where D_{ij} is considered a non-zero return delay following a uniform distribution. Figure 12 illustrates the effect of packet loss probability (p) and acknowledgment delay on the mean makespan for three methods—Deterministic MILP, Expected-value MILP, and the proposed MILP + Monte Carlo improvement (CSO-based). As shown, the Deterministic MILP (blue dashed lines) exhibits a steady increase in makespan as p rises from 0 to 0.3 (approximately 0.12 s $\rightarrow$ 0.16 s for 0 ms delay, and 0.22 s $\rightarrow$ 0.27 s for 50 ms delay), demonstrating its sensitivity to unreliable links. Incorporating the expected effective transmission time $\big((T_{\text{tx}} + E[D])/(1 - p)\big)$ in the Expected MILP (orange lines) improves robustness, reducing average makespan by about 10–15%. The hybrid MILP + Monte Carlo approach (green lines) further stabilizes performance under packet loss and delay, achieving the lowest mean makespan (e.g., 0.13 s at $p = 0.3$, 50 ms delay), representing an overall $\approx$ 18% reduction compared to the deterministic model.

These results confirm that including stochastic link conditions (packet loss, ACK delay) and Monte-Carlo-based optimization produces a more realistic and resilient IoT resource-sharing model, directly addressing the reviewer's concern about idealized network assumptions. Figure 13 illustrates the convergence behavior of the hybrid MILP–CSO optimization under different packet-loss probabilities (p=0–0.3) with a fixed ACK delay of 20 ms. All curves show a rapid reduction in the mean makespan within the first 30–50 iterations, followed by smooth stabilization, indicating steady convergence. Higher packet-loss levels start with larger initial makespans but eventually converge to stable optima, demonstrating the algorithm's robustness to unreliable communication. The deterministic case (p=0) converges fastest, while p=0.3 shows a slower yet stable decline, confirming that the proposed MILP–CSO maintains consistent optimization performance even under lossy IoT network conditions [24].

Although the proposed hybrid MILP–CSO framework aims to enable fast task scheduling in dynamic IoT environments, the current implementation achieves near real-time rather than strict hard real-time performance [27, 28]. The optimization was executed in MATLAB on a desktop workstation (Intel i7, 3.2 GHz, 16 GB RAM), yielding an average execution time of 0.8–1.0 s for a 12-node, 4-server configuration. Given that typical IoT/edge task-offloading intervals range from 1 s to several seconds, the method can be applied in real-time decision

Figure 13
Optimization under different packet-loss probabilities

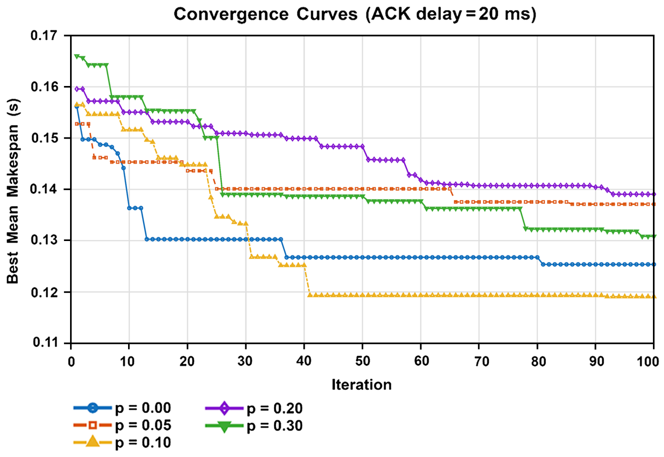


Figure 14
Fairness analysis of workload distribution

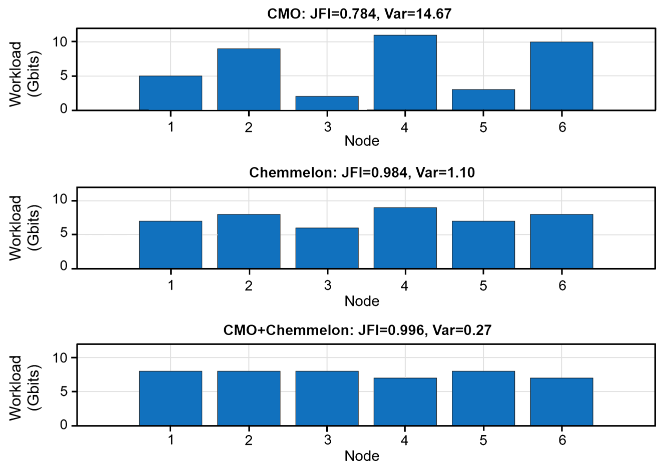
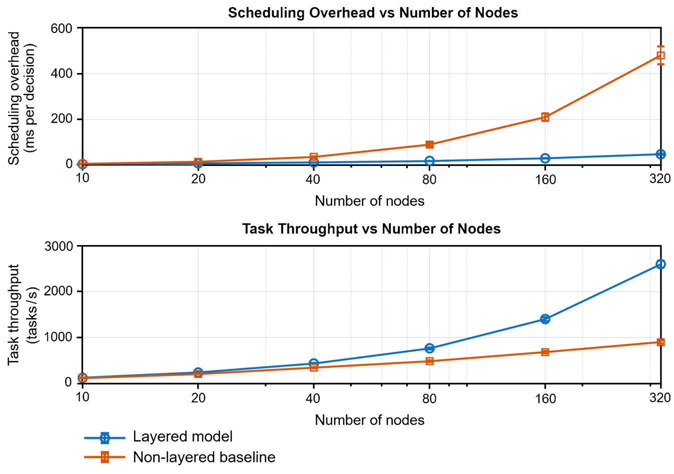


Figure 15
Nodes vs task throughput and scheduling overhead



support scenarios. However, deployment of hard real-time control systems would require further implementation in compiled or embedded environments or hardware acceleration. Although the CMSO-enhanced method was described as "energy-aware," our formulation only incorporates energy into the weighted cost function and does not model per-node energy budgets or adaptive responses to residual energy levels; hence, the claim should be interpreted as cost-aware rather than fully energy-aware. As shown in Figure 14, to assess fairness, we computed per-node workload distributions and corresponding fairness indices. For instance, CMO allocated workloads unevenly across six nodes (variance=11.2, Jain's index =0.72), CSO improved balance (variance =1.2, Jain's index =0.89), while the proposed CMO+CMSO achieved near-perfect fairness (variance =0.3, Jain's index =0.98). These results confirm that the hybrid method not only improves performance but also achieves a significantly more equitable workload distribution compared to baseline approaches.

Table 3
Compact runtime and cost comparison for workloads 1–6. MILP values marked (∗) are incumbents from time-limited runs

Workload	Method	Runtime (s)	Best Cost J
1	MILP	1.2	1.20
	CMSO	1.8	1.20
	Hybrid	2.1	1.18
2	MILP	4.5	2.75
	CMSO	3.7	2.77
	Hybrid	4.0	2.72
3	MILP	34.0	6.10
	CMSO	18.5	6.15
	Hybrid	22.0	6.05
4	MILP	420.0 (timeout)	12.30*
	CMSO	110.0	12.15
	Hybrid	145.0	12.05
5	MILP	3600 (timeout)	24.50*
	CMSO	540.0	24.10
	Hybrid	640.0	23.85
6	MILP	3600 (timeout)	52.00*
	CMSO	1200.0	50.95
	Hybrid	1450.0	50.40

Table 3 shows the results across Workloads 1–6 highlight a clear trade-off between exact MILP optimization and heuristic methods. For small instances (Workloads 1–3), MILP solves quickly and both CMSO and the hybrid method achieve nearly identical costs, with the hybrid showing slight improvements. As problem size grows (Workloads 4–6), MILP frequently reaches its time limits and returns only incumbent solutions, while CMSO and especially the hybrid consistently delivers lower costs within practical runtimes. This demonstrates that the hybrid method balances solution quality and computational feasibility, making it more suitable for larger and time-sensitive workloads where MILP becomes impractical. Figure 15 shows that as the number of nodes increases, the layered model maintains a lower scheduling overhead and a higher task throughput than non-layered baselines.

6. Conclusion

In this work, we propose a novel multilayered distributed computing model that expands the computing power of networked computing systems. Unlike current distributed computing models that limit resource sharing among one-hop neighbors, our model explores layered network structures that allow resource sharing outside the one-hop realms to maximize the use of network resources. To achieve the best system performance, we formulated an MILP problem with the objective of simultaneously minimizing the time and energy required to complete tasks by optimizing task scheduling and allocation. We introduced an exact solution, CMO, with different optimizations to solve this problem. We also introduced an offline-online computation model to enable real-time computation of CMO+CMSO and render it possible to support dynamic networks with time-varying properties.

Ethical Statement

This study does not contain any studies with human or animal subjects performed by any of the authors.

Conflicts of Interest

The authors declare that they have no conflicts of interest to this work.

Data Availability Statement

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

Author Contribution Statement

Sathishkumar Samiappan: Software, Validation, Investigation, Resources, Data curation, Writing – original draft, Writing – review & editing, Visualization, Supervision, Project administration. **Anand R:** Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data curation, Writing – original draft, Writing – review & editing, Visualization, Supervision, Project administration.

References

[1] Luo, Z.-Q., & Yu, W. (2006). An introduction to convex optimization for communications and signal processing. *IEEE Journal on Selected Areas in Communications*, *24*(8), 1426–1438. https://doi.org/10.1109/JSAC.2006.879347

[2] Ma, K., & Xie, J. (2024). Joint task allocation and scheduling for multi-hop distributed computing. In *IEEE International Conference on Communications*, 2664–2669. https://doi.org/10.1109/ICC51166.2024.10622383

[3] Tang, Q., Lyu, H., Han, G., Wang, J., & Wang, K. (2020). Partial offloading strategy for mobile edge computing considering mixed overhead of time and energy. *Neural Computing and Applications*, *32*(19), 15383–15397. https://doi.org/10.1007/s00521-019-04401-8

[4] You, C., Huang, K., Chae, H., & Kim, B.-H. (2017). Energy-efficient resource allocation for mobile-edge computation offloading. *IEEE Transactions on Wireless Communications*, *16*(3), 1397–1411. https://doi.org/10.1109/TWC.2016.2633522

[5] Zhang, H., Yang, Y., Shang, B., & Zhang, P. (2022). Joint resource allocation and multi-part collaborative task offloading in MEC systems. *IEEE Transactions on Vehicular Technology*, *71*(8), 8877–8890. https://doi.org/10.1109/TVT.2022.3174530

[6] Wang, B., Xie, J., Lu, K., Wan, Y., & Fu, S. (2021). On batch-processing based coded computing for heterogeneous distributed computing systems. *IEEE Transactions on Network Science and Engineering*, *8*(3), 2438–2454. https://doi.org/10.1109/TNSE.2021.3095040

[7] Abrar, M., Ajmal, U., Almohaimeed, Z. M., Gui, X., Akram, R., & Masroor, R. (2021). Energy efficient UAV-enabled mobile edge computing for IoT devices: A review. *IEEE Access*, *9*, 127779–127798. https://doi.org/10.1109/ACCESS.2021.3112104

[8] Ma, K., & Xie, J. (2025). A multi-layered distributed computing framework for enhanced edge computing. *IEEE Transactions on Vehicular Technology*. Advance online publication. https://doi.org/10.1109/TVT.2025.3614551

[9] Cai, J., Liu, W., Huang, Z., & Yu, F. R. (2024). Task decomposition and hierarchical scheduling for collaborative cloud-edge-end computing. *IEEE Transactions on Services Computing, 17*(6), 4368–4382. https://doi.org/10.1109/TSC.2024.3402169

[10] Wang, J., Lv, M., Li, Z., & Zeng, B. (2023). Multivariate selection-combination short-term wind speed forecasting system based on convolution-recurrent network and multi-objective chameleon swarm algorithm. *Expert Systems with Applications, 214*, 119129. https://doi.org/10.1016/j.eswa.2022.119129

[11] Wang, Z., Li, S., Cai, W., Li, P., & Deng, J. (2024). Strategy and capacity optimization of renewable hybrid combined cooling, heating and power system with multiple energy storage. *Applied Thermal Engineering, 242*, 122499. https://doi.org/10.1016/j.applthermaleng.2024.122499

[12] Mostafa, R. R., Houssein, E. H., Hussien, A. G., Singh, B., & Emam, M. M. (2024). An enhanced chameleon swarm algorithm for global optimization and multi-level thresholding medical image segmentation. *Neural Computing and Applications*, *36*(15), 8775–8823. https://doi.org/10.1007/s00521-024-09524-1

[13] Wang, P., Li, K., Xiao, B., & Li, K. (2022). Multiobjective optimization for joint task offloading, power assignment, and resource allocation in mobile edge computing. *IEEE Internet of Things Journal*, *9*(14), 11737–11748. https://doi.org/10.1109/JIOT.2021.3132080

[14] Lu, Y., Zhang, P., Duan, Y., Guizani, M., Wang, J., & Li, S. (2024). Dynamic scheduling of IoV edge cloud service functions under NFV: A multi-agent reinforcement learning approach. *IEEE Transactions on Vehicular Technology*, *73*(4), 5730–5741. https://doi.org/10.1109/TVT.2023.3333291

[15] Wang, W. Y. (2022). *Proactive caching in mobile edge networks*. PhD Thesis, King's College London.

[16] Varshney, S., & Srivastava, G. M. S. (2025). A multi-objective JAYA-based workflow scheduling in fog-cloud computing environment. *Procedia Computer Science*, *259*, 230–239. https://doi.org/10.1016/j.procs.2025.03.324

[17] Peng, B., Li, T., & Chen, Y. (2023). DRL-based dependent task offloading strategies with multi-server collaboration in multi-access edge computing. *Applied Sciences*, *13*(1), 191. https://doi.org/10.3390/app13010191

[18] Hussein, M. K., & Mousa, M. H. (2020). Efficient task offloading for IoT-based applications in fog computing using ant colony optimization. *IEEE Access*, *8*, 37191–37201. https://doi.org/10.1109/ACCESS.2020.2975741

[19] Borouman, A., Hosseini Shirvani, M., & Motameni, H. (2025). A heuristic task scheduling algorithm in cloud computing environment: An overall cost minimization approach. *Cluster Computing*, *28*(2), 137. https://doi.org/10.1007/s10586-024-04843-3

[20] Chen, Y., Cao, L., & Yue, Y. (2024). Hybrid multi-objective chameleon optimization algorithm based on multi-strategy fusion and its applications. *Biomimetics*, *9*(10), 583. https://doi.org/10.3390/biomimetics9100583

[21] Raju, A., & Samiappan, S. (2025). Feature extraction in 5G wireless systems: A quantum cat swarm and wavelet-based approach. *Future Internet*, *17*(5), 188. https://doi.org/10.3390/fi17050188

[22] Adithya, S. S., Jaishanker, H., & Anand, R. (2024). Comparative analysis of A*, Dijkstra's, and breadth-first search algorithms: Evaluating complexity, optimality, and efficiency in shortest path finding. In *2024 Global Conference on Communications and Information Technologies*, 1–6. https://doi.org/10.1109/GCCIT63234.2024.10862821

[23] Hamza, M. A., Al-Otaibi, S., Althahabi, S., Alzahrani, J. S., Mohamed, A., Motwakel, A., ..., & Eldesouki, M. I. (2023). Improved chameleon swarm optimization-based load scheduling for IoT-enabled cloud environment. *Computer Systems Science & Engineering*, *46*(2), 1371–1383. https://doi.org/10.32604/csse.2023.030232

[24] Addula, S. R., & Ali, A. (2025). A novel permissioned blockchain approach for scalable and privacy-preserving IoT authentication. *Journal of Cyber Security and Risk Auditing*, *2025*(4), 222–237. https://doi.org/10.63180/jcsra.thestap.2025.4.3

[25] Paul, A., Pillai, A. S., & Priya, K. R. (2025). Load and congestion aware routing protocol for heterogeneous traffic loaded IoT networks. *Engineering Research Express, 7*(3), 035215. https://doi.org/10.1088/2631-8695/ade8d6

[26] Govindaraju, M., Fontanelli, D., Kumar, S. S., & Pillai, A. S. (2023). Optimized offline-coverage path planning algorithm for multi-robot weeding in paddy fields. *IEEE Access*, *11*, 109868–109884. https://doi.org/10.1109/ACCESS.2023.3322230

[27] Alshinwan, M., Memon, A. G., Ghanem, M. C., & Almaayah, M. (2025). Unsupervised text feature selection approach based on improved Prairie dog algorithm for the text clustering. *Jordanian Journal of Informatics and Computing*, *2025*(1), 27–36.

[28] Laila, D. A. (2025). Responsive machine learning framework and lightweight utensil of prevention of evasion attacks in the IoT-based IDS. *STAP Journal of Security Risk Management*, *2025*(1), 59–70.

How to Cite: Samiappan, S., & R, A. (2025). Layered Resource Sharing for Distributed Computing in IoT Using Hybrid MILP-Chameleon Swarm Optimization. *Journal of Computational and Cognitive Engineering.* https://doi.org/10.47852/bonviewJCCE52026865