

Article

LITO: Lemur-Inspired Task Offloading for Edge–Fog–Cloud Continuum Systems

Asma Almulifi [†] and Heba Kurdi ^{*,†} 

Computer Science Department, College of Computer and Information Sciences, King Saud University, Riyadh 11451, Saudi Arabia

* Correspondence: hkurdi@ksu.edu.sa

[†] These authors contributed equally to this work.

Abstract

Edge, fog, and cloud continuum architectures that interconnect resource-constrained devices, intermediate edge servers, and remote cloud data centers face persistent challenges in handling heterogeneous and latency-sensitive workloads while reducing energy consumption and improving resource utilization. Classical task offloading approaches either rely on static heuristics, which lack adaptability to dynamic conditions, or on metaheuristic optimizers, which often incur high computational overhead and centralized coordination. This paper proposes LITO, a lemur-inspired task offloading algorithm for edge, fog, and cloud continuum systems that models the infrastructure as a social system in which computing nodes assume distinct roles that mirror lemur social hierarchies. Building on an abstracted model of lemur group behavior, LITO incorporates two key lemur-inspired mechanisms: an energy-aware task assignment mechanism based on sun basking, a thermoregulation behavior in which lemurs seek favorable warm spots, mapped here to selecting energetically efficient execution nodes, and a cooperative scheduling policy based on huddling, group clustering under stress, mapped here to sharing load among overloaded nodes. These mechanisms are combined with a continual supervised policy-learning layer with contextual bandit feedback that refines offloading decisions from online feedback. The resulting multi-objective formulation jointly minimizes energy consumption and deadline violations while maximizing resource utilization and throughput under high-load conditions in the edge and fog segment of the continuum. Simulations under diverse workload regimes and task complexities show that LITO outperforms representative multi-objective offloading baselines in terms of energy consumption, resource utilization, latency, Service Level Agreement (SLA) violations, and throughput in congested scenarios.

Keywords: edge–fog–cloud continuum; fog computing; edge computing; task offloading; resource management; Internet of Things; supervised policy learning



Academic Editor: Charith Perera

Received: 20 January 2026

Revised: 22 February 2026

Accepted: 24 February 2026

Published: 27 February 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

Fog computing has emerged as a pivotal paradigm that extends the cloud model by decentralizing storage, computation, and networking resources toward the periphery of Internet of Things (IoT) environments. Within such ecosystems, smart sensors deployed across domains such as smart healthcare, smart transportation, and smart cities continuously generate massive volumes of heterogeneous data. Even though cloud computing offers scalable resources, its dependence on centralized infrastructures inevitably incurs bandwidth congestion, high latency, and heightened data privacy concerns [1,2].

To mitigate these challenges, computation has progressively migrated closer to the edge. While edge devices (EDs) provide preliminary processing and rapid responsiveness, their limited resources impose fundamental restrictions on sustained performance and scalability. Fog computing therefore emerges as an intermediate layer that bridges edge and cloud environments.

Fog computing extends cloud computation, storage, and networking capabilities to the network edge, thereby addressing latency and Quality of Service (QoS) challenges [3,4]. It strategically deploys geo-distributed fog nodes such as routers, switches, and access points between IoT devices and centralized cloud infrastructure, facilitating local data storage, computation, and resource orchestration while simultaneously alleviating congestion. Architecturally, fog/edge–cloud collaborative computing is commonly described as a multi-layer model spanning end devices, nearby fog/edge nodes, and the back-end cloud [5]. This layered design enables efficient local processing while maintaining seamless cloud connectivity, offering advantages such as low latency, interoperability, scalability, real-time interaction, and support for heterogeneous devices [6]. By relocating computation and decision-making closer to the data source, fog computing enables faster responses, optimized bandwidth utilization, and improved QoS. Reflecting this importance, the global fog computing market, valued at USD 448 million in 2024, is expected to reach USD 5.41 billion by 2035, propelled by industrial automation and IoT adoption [7]. Furthermore, fog computing supports application domains unsuitable for cloud-only solutions, particularly those requiring geographically distributed systems, ultra-low latency, and large-scale adaptive control [8].

To enable such diverse applications, the fog network encompasses heterogeneous devices endowed with computation, storage, and communication capabilities. However, the inherently dynamic nature of fog environments—compounded by device mobility and fluctuating workloads—makes resource management a significant challenge. Effective resource management is essential to fully leverage the processing power of fog and improve overall performance [9]. Within this framework, task offloading plays a vital role, allowing IoT end devices to delegate workloads to nearby fog nodes for remote computation, thereby reducing latency while conserving energy. As the proliferation of IoT devices increases, maintaining stringent QoS parameters such as response time, throughput, and resource utilization becomes critical [4]. Offloading strategies seek to optimize computing performance by judiciously selecting appropriate layers, fog or edge, depending on energy efficiency, processing demand, and cost-effectiveness [10]. However, the offloading process remains profoundly complex, requiring effective fog node selection to balance resources and ensure seamless operations [11]. Consequently, contemporary research in fog computing focuses on optimizing objectives such as energy minimization, service delay reduction, and bandwidth efficiency, while ensuring compliance with constraints imposed by QoS guarantees, power limitations, and application-specific requirements [12].

While conventional heuristic-based offloading strategies often fall short in dynamic and heterogeneous environments [13], metaheuristic algorithms have shown greater promise due to their ability to explore large solution spaces and balance exploration and exploitation [14]. Metaheuristics, many of which are inspired by biological or physical processes, offer adaptive and scalable solutions to optimization problems, supporting improved responsiveness and better resource utilization [15–17]. However, scaling these methods in large fog networks introduces new limitations. Traditional algorithms such as Genetic Algorithms (GA), Particle Swarm Optimization (PSO), Grey Wolf Optimization (GWO), and Simulated Annealing (SA) may suffer from low convergence speed, susceptibility to local optima, and computational inefficiency in real-time settings [18]. Moreover, many approaches assume relatively stable conditions or require extensive tuning/training

to remain effective as network states and resources vary, which can hinder adaptability and scalability in real-world deployments [19,20].

Motivated by these gaps, we introduce the Lemur-Inspired Task Offloading (LITO) framework, which is a decentralized and adaptive approach. The term decentralized in LITO indicates the distribution of decision-making authority across Primary Nodes (PNs), Support Nodes (SNs), and EDs, in which each tier takes autonomous yet coordinated decisions without depending on a single centralized optimizer. EDs independently find local execution or offloading through Very fast decision trees (VFDT) classifiers, which are trained on streaming task features. By utilizing aggregated telemetry and the continual supervised policy-learning formulation with contextual bandit feedback (CSPL-CB) learner, PNs make segment-level global routing decisions. In contrast, SNs implement hybrid Earliest Deadline First–Dynamic Resource Scheduling (EDF–DRS) scheduling locally. This layered autonomy minimizes coordination overhead and avoids the dependence on global recomputation. Adaptiveness in LITO is attained through feedback-driven policy refinement and multi-level continual learning. VFDT updates local offloading decisions incrementally under varying workload conditions, whereas CSPL-CB at PNs refines offloading policies continuously using oracle supervision and contextual bandit feedback under non-stationary conditions. Depending on real-time resource availability and urgency factors, SNs adjust scheduling priorities dynamically. The feedback and adaptation module facilitates policy evolution under fluctuating network bandwidth, energy levels, and node utilization. These mechanisms enable LITO to preserve deadline adherence, energy efficiency, and workload balance across evolving fog infrastructures.

We first formulated the task offloading problem as a constrained multi-objective optimization problem that jointly accounts for energy consumption, end-to-end latency, and resource–utilization balance under deadline and capacity constraints. We then evaluated LITO against representative multi-objective baselines within a fog-computing simulation environment by designing scenarios that vary workload intensity, task complexity, and system configuration. Finally, we complemented these external baselines with an ablation study that isolates the effect of LITO design choices (local-only execution, full offloading, hybrid edge policy, support-node-only execution, and the proposed edge support collaboration), providing a focused yet comprehensive evaluation of the proposed framework. Across these experiments, the proposed LITO approach demonstrates up to 60% reduction in energy consumption and over 30% improvement in resource utilization compared to state-of-the-art multi-objective offloading baselines, alongside lower latency and higher SLA compliance under stress-tested fog scenarios.

The main contributions of this work are outlined as follows:

- Lemur-inspired task offloading framework: We proposed a Lemur-Inspired Task Offloading (LITO) framework that models fog environments as social systems. By assigning role-based hierarchies to Edge Devices (EDs), Support Nodes (SNs), and Primary Nodes (PNs), the system enables decentralized, adaptive task distribution inspired by lemur group behaviors.
- Hybrid local learning and scheduling: We developed an integrated hybrid learning architecture that combines Very Fast Decision Trees (VFDT) at the ED level for incremental, local decision-making with a hybrid Earliest Deadline First–Dynamic Resource Scheduling (EDF–DRS) policy at the SN level, improving responsiveness under varying workload conditions.
- Continual policy learning with contextual bandits: We employed a continual supervised policy-learning formulation with contextual bandit feedback (CSPL-CB) at the PN level to optimize global offloading policies over time. This approach supports

learning continuity and adaptability to evolving network dynamics, enhancing long-term policy efficiency.

- Comprehensive empirical evaluation: We conducted a comprehensive empirical evaluation across varying workloads, task complexities, and system configurations.

This paper is organized as follows. Section 2 reviews the related literature. Section 3 details the proposed system design. Section 4 outlines the evaluation methodology, and Section 5 presents the experimental results. Finally, Section 6 concludes the paper and discusses directions for future research.

2. Literature Review

Task offloading has attracted significant research attention because of the increasing demand for latency-sensitive and resource-constrained applications. Several studies address these challenges using metaheuristic-based optimization techniques that search for near-optimal offloading strategies under multiple performance objectives.

2.1. Metaheuristics-Based Task Offloading Approaches

Existing studies on task offloading in edge and fog environments can be classified into (i) trajectory-aware energy optimization, (ii) multi-objective metaheuristic optimization, and (iii) workflow and digital twin-assisted optimization frameworks.

Unmanned Aerial Vehicle (UAV)-assisted MEC systems present unique limitations because of obstacle avoidance, limited battery storage, and dynamic maneuverability demands. Hence, a joint optimization framework [21] is proposed to improve energy efficiency by simultaneously enhancing user task offloading rates and UAV trajectories. This framework combines hybrid alternating metaheuristics and transformation techniques to effectively enable obstacle-free UAV navigation and reduce total energy consumption. However, the offloading tasks are assumed with fixed bandwidth and computational requirements, overlooking real-world heterogeneity in which tasks may differ in deadlines, priority, and dependencies across devices.

User equipment-driven offloading decisions become a critical challenge because of high energy consumption and congestion at the edge. To address these, offloading is considered as a constrained multi-objective optimization problem in [22], and metaheuristic techniques, like Tug of War Optimization, Hybrid Improved Whale Optimization, and Adaptive Equilibrium Optimization, are employed that minimize execution delays and energy utilization with improved task satisfaction. However, it does not explicitly consider multi-user task offloading circumstances. To address this problem, a dynamic multi-user, multi-server MEC system is developed in [23] along with stochastic task arrivals. A PSO-based algorithm aids in minimizing average task delay via efficient load balancing and adaptive server selection. Even though makespan and energy are optimized, the trade-off between them is not investigated under diverse application requirements.

In order to optimize multiple objectives like energy consumption, makespan, and resource utilization, a quantum-inspired PSO approach [24] is proposed for workflow scheduling in edge environments. However, the PSO parameters, such as learning factors, swarm size, and iterations, are fixed across diverse experiments. To tackle the task of offloading challenges on the Internet of Drones (IoD), PSO is integrated with fog base stations in [25] to minimize transmission delays and enhance processing and storage capacity. However, it leverages static acceleration coefficients and a fixed number of iterations that may not efficiently adapt to real-time fluctuations.

In IIoT, a multi-objective algorithm task offloading, named ODTRA [26], is proposed that utilizes probabilistic recursive local search (PRLS) and the water cycle metaphor (WCM) to minimize the execution time. Digital twin is leveraged for real-time monitoring and

predictive modelling, while allowing intelligent and proactive resource allocation and task offloading. It improves performance by minimizing latency, enhancing energy efficiency, and ensuring QoS. However, fixed parameters such as noise, bandwidth, distance, and energy weights do not consider real-world variations.

2.2. Metaheuristics-Based Task Offloading Approaches in Fog Computing

Existing studies on metaheuristic-based task offloading in fog environments can be divided into six groups on the basis of their optimization objectives and decision strategies. They are (i) latency–load balancing optimization, (ii) multi-objective system-wide optimization, (iii) joint offloading–scheduling integration, (iv) context-aware optimization approaches, (v) hybrid and multi-strategy metaheuristic frameworks, and (vi) distributed and online optimization approaches.

Several studies [27–29] tackle the problem of balancing task offloading time, latency, and workload distribution in three-tier IoT–fog–cloud architectures. By identifying the inefficiency of cloud-only offloading in latency-sensitive IoT applications, a hybrid framework is proposed in [27] using the Flamingo Search Algorithm (FSA) for initial exploration and the Honey Badger Algorithm (HBA) for refinement. This framework integrates the benefits of both algorithms to construct an optimized offloading framework with reduced latency and improved workload balance but fails to scale efficiently. Similarly, in [28], offloading is formulated as a multi-objective optimization problem, concentrating on resource utilization and delay reduction in fog computing. The Discrete Jaya Algorithm (DJA) is utilized with a task migration scheme, while ensuring resilience and improving adaptability and efficiency. However, the NP-hard nature makes convergence slow. Processing and energy constraints in end devices necessitate effective task offloading across IoT–fog–cloud. Existing approaches focus on localized optimization rather than system-wide optimization. A task offloading model [30] has been developed using an improved non-dominated sorting crow search algorithm (INSCSA) with multiple objectives such as response time, energy, cost, and availability. It attains system-wide optimization with improved overall efficiency. Both studies enable efficient exploration and exploitation, thus enhancing resource utilization and system responsiveness within dynamic fog environments.

Studies [31–33] underscore the interdependence between scheduling and offloading tasks within heterogeneous fog–cloud systems. In [31], the trade-off between delay and energy is highlighted in cloud-fog systems in which task execution order directly influences stability. A Multi-objective Arithmetic Optimization Algorithm (MoAOA) is developed with a personalized fitness function by considering energy, delay, and task priority. It achieves balanced offloading and efficient scheduling, dynamically adapting to varied workloads. However, it is computationally expensive when the network size increases, resulting in extended convergence time for large-scale deployments. To address the issues related to high latency, inefficient resource allocation, and energy consumption in fog computing, offloading is applied hierarchically in [32], in which tasks are transferred from devices to fog nodes, higher-capacity fog nodes, and then to the cloud when needed. A hybrid GA-SA-GWO is leveraged to optimize latency, task scheduling, energy utilization, and overall system performance. Even though hybrid metaheuristics are leveraged, the main contribution is in combining hierarchical offloading decisions and scheduling. However, this hybrid algorithm will increase computational complexity and delay the offloading decision-making process in highly dynamic environments. To tackle these challenges, GA Hybrid-Fog is proposed in [33], which combines GA with fog base stations and fog UAVs to optimize both transmission and fog computing latency. It considers both static and mobile fog resources but does not address heterogeneous task requirements or task priority management. This requirement influences system reliability when high-priority tasks are

delayed due to low-priority jobs. The model assumes ideal conditions for noise power, transmission bandwidth, and processing cycles.

Context-aware optimization methods [34–36] adapt task offloading strategies for specific workload characteristics or application scenarios. In [34], smart ant colony optimization (SACO) is developed to address the latency issues in IoT-sensor-based applications. Notable reductions in offloading delay are achieved by comparing SACO with throttled scheduling, Round Robin, and other bio-inspired algorithms. A study [35] has extended this task offloading process to workflow applications within heterogeneous fog–cloud infrastructures. Fuzzy dominance-based clustering is leveraged along with the hybrid Harmony Search Algorithm (HSA) and GA for task scheduling. It effectively balances makespan, cost, and resource utilization; however, it assumes static task arrival patterns in workflows. For critical task execution, a two-stage framework [36] is proposed with enhanced task offloading (ETO). It chooses the computing layer, which is followed by the multi-objective firefly algorithm (MFA) for device selection in the selected layer. This layered approach aids in minimizing energy consumption and delay while improving reliability. However, the study does not consider the resource heterogeneity across fog, edge, and cloud layers, affecting real-world performance.

Existing studies [37–39] have explored hybrid and multi-strategy optimizers to resolve the shortcomings of conventional metaheuristic algorithms. A logarithm Mayfly Algorithm (IMA) [37] is introduced for bag-of-tasks applications within fog systems. IMA effectively balances local exploitation and global exploration by refining global coefficients and individual learning. It proves efficiency gains in cost and execution time, but it fails to integrate strict task deadlines, budget limits, varying network latencies, or fluctuating energy availability into its optimization process. In [38], an Integrated Snake Optimizer (ISO) is utilized along with circle map initialization, elite reverse learning, and synergistic global search to improve solution diversity and convergence speed and reduce costs and delays. Even though it addresses independent bag-of-tasks applications, it may not generalize well to interdependent or workflow-based tasks common in real IoT–fog–cloud applications. A multi-objective optimization problem is solved using Non-dominated Sorting GA II (NSGA-II) and the Bees Algorithm [39] within mobile fog systems. A balanced trade-off between energy consumption and task execution delay is ensured, while achieving robustness in dynamic mobile environments. However, it depends on hierarchical agglomerative clustering, which incurs high time complexity. Moreover, it is computationally expensive on large networks, even if clustering is assumed to be infrequently executed.

Studies [40,41] have explored distributed and online optimization models. A fully distributed computation offloading algorithm [40] is proposed to reach the Nash equilibrium without centralized coordination. Even though fully distributed, it assumes normal user behavior and improves execution cost, without solving hierarchical fog coordination or adaptation in case of non-stationary workloads. Similarly, a two-time-scale accuracy-aware online optimization framework [41] is proposed that depends on Lyapunov decomposition. Although it dynamically handles delay and energy constraints, its performance is affected by the dependence on centralized control and mathematically intensive optimization. To solve dynamic task offloading in fog environments, a distributed twin-delayed deep deterministic policy gradient (TD3) framework [42] is leveraged with multiple actors, enhancing learning and scalability. However, training instability occurs in the case of highly non-stationary workloads. Distributed deep reinforcement learning (μ -DDRL) [43] addresses directed acyclic graph-based service offloading through asynchronous actor–critic learning with proximal policy optimization (PPO) clipping and V-trace. Although it enables faster convergence and better adaptability, the decision-time overhead is higher. Deep deterministic policy gradient (DDPG)-based offloading [44] is proposed to improve delay

in heterogeneous edge networks. However, its single-agent design restricts scalability and robustness in large-scale distributed settings. In contrast, LITO combines hierarchical decentralization with lightweight continual learning to support adaptive, scalable, and resource-aware task delegation across fog layers.

Existing metaheuristic-based methods depend on global optimization paradigms that lack integrated online policy refinement and isolate offloading from hierarchical coordination. Although they enhance energy and latency trade-offs, their architectural designs are inadequate for scalable, multi-tier fog systems functioning under heterogeneous and emerging workload conditions. Consequently, coordinated decision-making across edge and fog nodes remains underexplored. These structural shortcomings drive the need for a decentralized, learning-driven framework that combines hierarchical coordination with adaptive policy evolution, as realized in the proposed LITO architecture. Table 1 presents a comparative analysis of well-known metaheuristic-based task offloading approaches, highlighting their primary objectives, decision-making metrics, optimization techniques, application domains, and key limitations.

From the surveyed literature, metaheuristic-based task offloading across fog and IoT environments has achieved notable improvements in latency, energy efficiency, and workload balancing. However, several research gaps persist. First, task and environmental heterogeneity are only partially addressed. In many studies [21,25,26,32,33,36], tasks are modeled with fixed parameters and simplified timing assumptions. Strict deadline guarantees, task priorities, and inter-task dependencies are typically not modeled explicitly, which limits the adaptability of these approaches to diverse real-world workloads. Hybrid and multi-objective approaches [27,28,30–32,39,45] enhance optimization quality but often incur high computational costs or rely on complex clustering and global coordination, which constrains their applicability in large-scale deployments. Likewise, static parameterization of metaheuristic algorithms in works such as [21,24,27,29–31,33–35,39,45] (for example, fixed PSO and GA coefficients) hinders adaptability to dynamic workloads. Offloading approaches in [28,30,37,45] frequently assume stable resource conditions within each time slot or provide limited treatment of node and link failures, which can lead to inefficiencies and reduced robustness in unstable environments, especially across multiple tiers of the edge–fog–cloud continuum.

Existing studies have often optimized task offloading in isolation and overlook the interdependence between task scheduling and offloading. Studies [31,45] have considered joint optimization, but scalability and dynamic adaptation challenges remain unsolved. Advanced optimizers such as IMA [37], ISO [38], and NSGA-II with Bees [39] improve convergence and robustness, yet they are computationally expensive and still do not integrate dynamic constraints such as heterogeneous deadlines, fluctuating energy availability, and varying network conditions. Most existing metaheuristic-based task offloading approaches depend on system-wide information exchange and centralized optimization with global coordination [21,24–34,36,37,39,45], whereas only limited works have considered decentralized [29] or partially decentralized architectures [35]. Therefore, there is still a need for a decentralized and adaptive framework that can handle heterogeneous, large-scale, and real-time environments and explicitly coordinate decisions across multiple tiers of the edge–fog–cloud continuum.

Table 1. Summary of well-known metaheuristics-based task offloading approaches.

Ref.	Optimization Objectives	Decisions Metrics	Algorithmic Strategy	Centralized/Decentralized	Dynamic Adaptation Capability	Application Domain	Limitations
[21]	Joint energy-efficient task offloading and UAV trajectory optimization	Energy consumption, amount of offloaded data, maneuverability, and obstacle avoidance	Hybrid alternating metaheuristics	Centralized	Static	UAV-assisted MEC	Limited scalability: task and device heterogeneity are not addressed
[24]	Multi-objective workflow scheduling optimization	Average task delay, energy cost, workload distribution	Quantum-Inspired PSO	Centralized	Static	Real-time MEC with variable workloads	Static PSO parameterization
[25]	Task offloading optimization in fog/IoD	Transmission delay, fog computing delay, storage capacity, processing capacity	PSO	Centralized	Partial (iterative but not environment-adaptive)	IoD	PSO is susceptible to premature convergence and local optima
[26]	Multi-objective task offloading with Digital Twin integration	Task execution time, bandwidth, server capacity, and device energy consumption	PRLS, WCM	Centralized	Partially dynamic (state-aware iterative optimization)	Industrial IoT	Fixed parameters are considered
[27]	Latency-aware hybrid offloading optimization	Latency, load balancing, and task offloading time	FSA, HBA	Centralized	Static	Latency-sensitive IoT	More complex due to two-stage optimization
[28]	Joint resource utilization and delay minimization	Delay, resource utilization, task failure handling	DJA	Centralized	Partial (limited dynamic capability)	IoT-fog computing	NP-hard nature makes convergence slow
[30]	Multi-objective system-wide offloading optimization	Response time, energy consumption, cost, and availability criteria	INSCSA	Centralized	Static	Fog computing	Lack of consideration of dynamic workloads
[31]	Joint offloading and scheduling optimization	Energy consumption, transmission delay, task priority	MoAOA	Centralized	Static	Real-time IoT with cloud-fog	High computational complexity
[29]	Dynamic offloading under OFDM	Delay, energy consumption	Parallel Multi-threaded PSO	Decentralized	Dynamic	IoT-fog	Lack of dynamic condition handling
[45]	Pareto-based makespan-cost optimization	Makespan, cost	LD-NPGA	Centralized	Static	IoT-fog	Generate diverse Pareto sets while increasing decision complexity

Table 1. Cont.

Ref.	Optimization Objectives	Decisions Metrics	Algorithmic Strategy	Centralized/Decentralized	Dynamic Adaptation Capability	Application Domain	Limitations
[32]	Hierarchical latency and energy optimization	Delay, resource, and energy utilization	GA-SA-GWO	Centralized	Dynamic	Edge-fog	Computational complex; slow offloading decisions
[33]	Hybrid fog-based latency optimization	Transmission delay, delay, storage capacity	Genetic Algorithm	Centralized	Static	IoD-fog	Task prioritization and heterogeneity are ignored
[34]	Latency reduction in IoT applications	Latency, offloading time	SACO	Centralized	Static	IoT-fog computing	Performance relies on parameter tuning
[35]	Multi-objective scheduling in workflows	Makespan, cost, resource utilization	HAS, Genetic algorithm	Partially Decentralized	Static	Scientific workflows; fog-Cloud	Lack of adaptivity to dynamic workloads
[36]	Multi-objective critical task offloading	Energy consumption, delay, resource utilization, server availability	MFA	Centralized	Partial dynamic	Edge-Fog-Cloud	Limited consideration of heterogeneity
[37]	Multi-strategy offloading optimization	Execution time, operating cost, and convergence	IMA	Centralized	Partial dynamic	IoT; cloud-fog	Focus only on a single objective
[39]	Delay and energy-aware optimization	Power consumption, delay, and offloading probability	NSGA-II; Bees Algorithm	Centralized	Static	IoT-fog	Computationally expensive clustering dependency
[42]	Delay and energy-aware distributed load balancing optimization	System cost, latency, energy consumption, resource utilization	Distributed Twin-Delayed DDPG	Decentralized	Dynamic	MEC-enabled vehicular /fog computing	Training instability under extreme non-stationarity
[43]	Multi-objective critical task offloading	Execution time, convergence speed, scalability, decision time overhead	Asynchronous PPO with V-trace and PPO clipping	Decentralized	Dynamic	Fog computing	Higher decision time overhead
[44]	Multi-strategy delay-minimization and mobility-aware offloading optimization	Task delay, completion rate, energy consumption	Actor-Critic DRL (DDPG)	Decentralized	Partially dynamic	Heterogeneous MEC	Less stable convergence

3. System Design

3.1. Problem Formulation

The LITO algorithm is designed to improve efficiency and adaptability in fog computing by dynamically offloading tasks from IoT devices to edge and fog nodes. The set of tasks is defined as $T = \{t_1, t_2, \dots, t_n\}$, in which each task t_i has the following attributes: workload W_{t_i} (CPU cycles), deadline D_{t_i} , Priority P_{t_i} , dependency set d_{t_i} ($d_{t_i} \subseteq T$), and energy requirement E_{t_i} . The feature vector of task t_i is given by,

$$X_i = (W_{t_i}, D_{t_i}, P_{t_i}, E(d_{t_i}), E_{t_i}) \quad (1)$$

As a set of task dependencies d_{t_i} is high-dimensional categorical field, it is encoded using adjacency binary vector as $E(d_{t_i})$.

Assume the set of nodes is $M = \{m_1, m_2, \dots, m_k\}$ in which each node m_j has capacity C_j (cycles/s), energy level E_j (residual energy of nodes), and communication delay L_{ij} . Let $A_j \subseteq T$ denote the set of tasks assigned to node m_j within a scheduling slot of length T_s , and let U_j denote the utilization of node m_j . The main objective is to minimize the Resource Utilization Gap (RUG), which measures the difference between available computational resources and the amount actually utilized during task execution, while ensuring that deadlines, workload balance, and energy constraints are met. The objective function is given by:

$$\min J = \sum_{t_i \in T} (\alpha L_{t_i} + \beta E_{t_i} + \gamma I_{t_i}) \quad (2)$$

where L_{t_i} is the end-to-end delay (including both communication and execution); E_{t_i} is the energy consumed at the node that runs the task; I_{t_i} quantifies the contribution of task t_i to the imbalance in resource utilization across (SNs), which is directly related to the RUG metric. Coefficients α , β , and γ are weighting factors that reflect system priorities.

The optimization problem is formulated under practical system constraints. It ensures energy awareness, deadline compliance, feasibility, and balanced utilization across fog nodes.

- **Deadline Constraint (C1):** Each task needs to finish before its deadline under EDF scheduling: C1 enforces that each task must respect its deadline while being scheduled under an EDF policy:

$$C1 : \text{Completion_Time}(t_i) \leq D_{t_i}, \forall t_i \in T \quad (3)$$

ensuring strict adherence to timing requirements.

- **Energy-Aware Offloading Constraint (C2):** A task is offloaded from an ED only when local energy $E_{t_i}^{local}$ is more than a predefined threshold E_{Th} :

$$C2 : E_{t_i}^{local} > E_{Th} \text{ (during offloading)} \quad (4)$$

enforcing energy conservation at constrained EDs.

- **Capacity Constraint (C3):** The total workload allocated to an SN should not exceed its available capacity:

$$C3 : \sum_{t_i \in A_j} W_{t_i} \leq C_j T_s; \forall m_j \in M \quad (5)$$

preventing overload within a scheduling slot T_s .

- **Load Balancing Constraint (C4):** It constrains the difference between the most and least utilized SNs to be within a threshold δ , thus promoting balanced resource usage:

$$C4 : \max_{m_j \in M} U_j - \min_{m_j \in M} U_j \leq \delta \quad (6)$$

promoting balanced resource allocation.

- **Feasibility Constraint for Offloaded Tasks (C5):** For each offloaded task, execution and communication delay must satisfy:

$$C5: L_{ij} + \frac{W_{t_i}}{C_j} \leq D_{t_i}, \forall t_i \text{ offloaded to } m_j \quad (7)$$

guaranteeing deadline feasibility at the chosen fog node.

3.2. Lemur Behavior as Inspiration for PLBA

Lemurs, shown in Figure 1, are primates native to Madagascar that live in social groups and typically inhabit forested, tree-rich environments [40]. Their societies are matriarchal, with dominant females guiding group decisions and managing access to resources. Males usually play supportive roles, such as territorial defense and conflict mediation, while offspring depend on the group for protection and learn by observing adult behavior [42].

In this work, we idealize these social dynamics into a role-based model that we call the Pure Lemur Behavior Abstraction (PLBA). PLBA is a conceptual description of how roles are distributed in a lemur group, how information flows between them, and how the group reacts to changes in the environment. PLBA acts as a conceptual behavioral framework that motivates the algorithmic mechanisms in LITO, but it does not define any learning rules or computational steps.

The LITO algorithm then instantiates PLBA in a fog-computing context. Specifically, fog components are assigned roles that mirror those in PLBA: PNs emulate the leadership role of dominant females by taking global offloading decisions; SNs correspond to supportive males that stabilize the system and help handle overflow tasks; and EDs play the role of offspring, with limited resources but the ability to make local decisions and adapt based on feedback. PLBA provides the behavioral template, while LITO defines the concrete decision rules and learning mechanisms that these nodes follow.

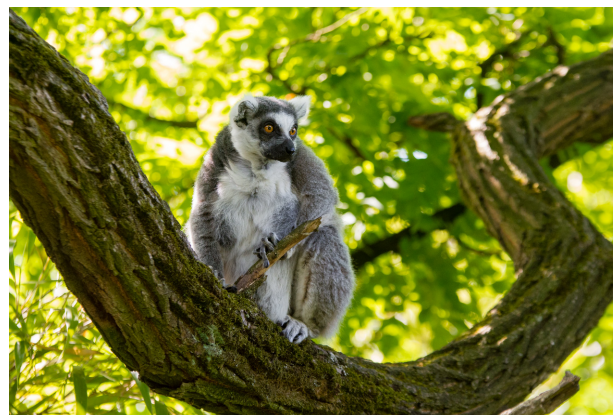


Figure 1. A Lemur (CC0 1.0), source: Wikimedia Commons [41].

Individual lemur behaviors also motivate specific mechanisms in LITO. Sun basking, where lemurs conserve energy by seeking warmth, inspires an energy-aware offloading policy that prefers nodes with better energy conditions. Huddling, a group behavior used for warmth and protection during threats or cold conditions [42], motivates cooperative task redistribution and load balancing when the system is under stress or when some nodes fail. The correspondence between these lemur behaviors in PLBA and their algorithmic counterparts in LITO is summarized in Table 2.

Table 2. Mapping lemur behaviors to the LITO framework.

Algorithm	Lemur Behavior (PLBA)	LITO Mechanism
System Objective	Adaptive group survival and efficiency	Adaptive and efficient task offloading in fog systems
Hierarchy/Roles	Females (leaders), Males (supporters), Offspring (learners)	PNs (global coordinators), SNs (overflow handlers), EDs (local learners and executors)
Inputs	Environmental stimuli (threats, temperature, food), internal energy/health state	Resource availability, task urgency, energy levels
Monitoring Mechanism	Continuous observation of surroundings and internal status	Real-time sensing of network and device conditions
Decision Flow	Female-led decisions, male support, offspring follow and learn	PN-led routing, SN task support, ED local decision-making and learning
Energy Restoration (Sun Basking)	Seeking sunlight to replenish energy	Offloading tasks to resource-rich nodes to optimize energy usage
System Stability (Huddling)	Clustering for warmth and protection under stress	Cooperative task redistribution during heavy load or node failures
Learning Mechanism	Offspring observe adult behavior to develop survival strategies	EDs incrementally learn using VFDT and performance feedback
Execution Outcome	Coordinated foraging, relocation, and defense	Distributed task allocation ensuring low latency, energy efficiency, and SLA adherence

The behavioral abstractions are mapped to explicit quantitative decision rules to reinforce the correspondence between algorithmic realization and biological inspiration.

Sun Basking to Energy-Based Offloading Rule: When an individual's thermal energy is less than a threshold within biological systems, sun basking is triggered. In LITO, this behavior is modeled via an energy-awareness condition at EDs. Specifically, when an ED's residual energy E_j is less than a predefined fraction of its initial energy ηE_j^{init} , the task is considered offload-eligible and assessed for fog-layer assignment:

$$E_j < \eta E_j^{init} \quad (8)$$

where $0 < \eta < 1$ indicates the energy preference coefficient. Amongst candidate fog nodes, selection is based on an energy-efficiency score:

$$Score_j = \frac{C_j}{P_j^{busy}} \quad (9)$$

where C_j indicates computational capacity; P_j^{busy} indicates busy-state power consumption. Nodes that have higher scores are prioritized for task assignment.

Huddling to Utilization-Imbalance Trigger: The huddling-inspired cooperative load redistribution is performed through the utilization-imbalance constraint defined in C4 (Equation (6)), in which redistribution is activated when the utilization gap across SNs is more than threshold δ .

These threshold-based mappings create a direct and quantifiable correspondence between algorithmic decision triggers and lemur behavioral stimuli in LITO.

3.3. System Model

The proposed framework adopts a multi-layered infrastructure embedded in an edge–fog–cloud continuum. The system is organized into four operational layers and two auxiliary modules for coordination and performance feedback:

IoT Layer. This foundational tier is responsible for sensing, data generation, and initial communication. It includes physical devices such as sensors and actuators that produce tasks with heterogeneous workloads and latency requirements. Tasks are encapsulated into digital descriptors and sent to the edge layer for processing.

Edge Layer (Local Offloading Decision). Edge devices (EDs), representing the “ordinary lemurs” in the social model, are the first programmable tier above IoT devices. They perform lightweight preprocessing, local queuing, and initial offloading decisions based on local context (for example, queue length, residual energy, and link quality). When local execution is inefficient or infeasible, the task is packaged and forwarded to the next layer.

Fog Layer (Hierarchical Global Offloading). This layer is logically split into two sublayers that collectively implement hierarchical global offloading and scheduling:

- **Upper Fog Sublayer (Primary Nodes).** Primary Nodes (PNs) are high-capacity fog servers or industrial-grade controllers that maintain a global, though approximate, view of the edge–fog segment. They receive summarized state information from Support Nodes (SNs), including load conditions, energy levels, and task outcomes, and run the continual supervised policy-learning with contextual bandit feedback (CSPL-CB) mechanism to select suitable execution targets for incoming tasks.
- **Lower Fog Sublayer (Support Nodes).** Support Nodes (SNs), which play the role of “supporting lemurs,” are intermediate servers that receive offloaded tasks from PNs and EDs. They manage execution queues using a hybrid Earliest Deadline First–Dynamic Resource Scheduling (EDF–DRS) policy, enforce feasibility constraints, and execute tasks using their local computational resources. After execution, SNs report status and results upward to the PNs and, when needed, back to EDs.

Cloud Layer (Back-End Analytics and Non-Critical Workloads). Above the fog layer, a cloud tier provides virtually elastic compute and storage resources. While LITO’s real-time offloading decisions primarily operate between EDs, SNs, and PNs, the cloud layer serves as a back-end for training and updating policy models using aggregated telemetry, executing delay-tolerant or batch workloads, and storing long-term logs. Conceptually, this completes the edge–fog–cloud continuum: latency- and energy-critical tasks are retained in the edge and fog strata, whereas non-urgent processing can be migrated to the cloud when beneficial.

System Coordination and Control Module. The system manager maintains a registry of all active devices and nodes, tracks their roles and capabilities, and enforces global configuration policies. It assists in node discovery, admission and departure handling, and the dissemination of updated offloading and scheduling policies across all SNs and EDs.

Feedback and Adaptation Module. This module gathers detailed performance metrics from EDs, SNs, and PNs, including throughput, energy consumption, task latency, and fairness indicators. It periodically aggregates these metrics and exposes them to the CSPL-CB learner and to off-line analytics in the cloud layer. This feedback loop enables continual adaptation of LITO’s decision policies to changes in workload patterns, node capacities, and network conditions. The overall multi-layered infrastructure is depicted in Figure 2.

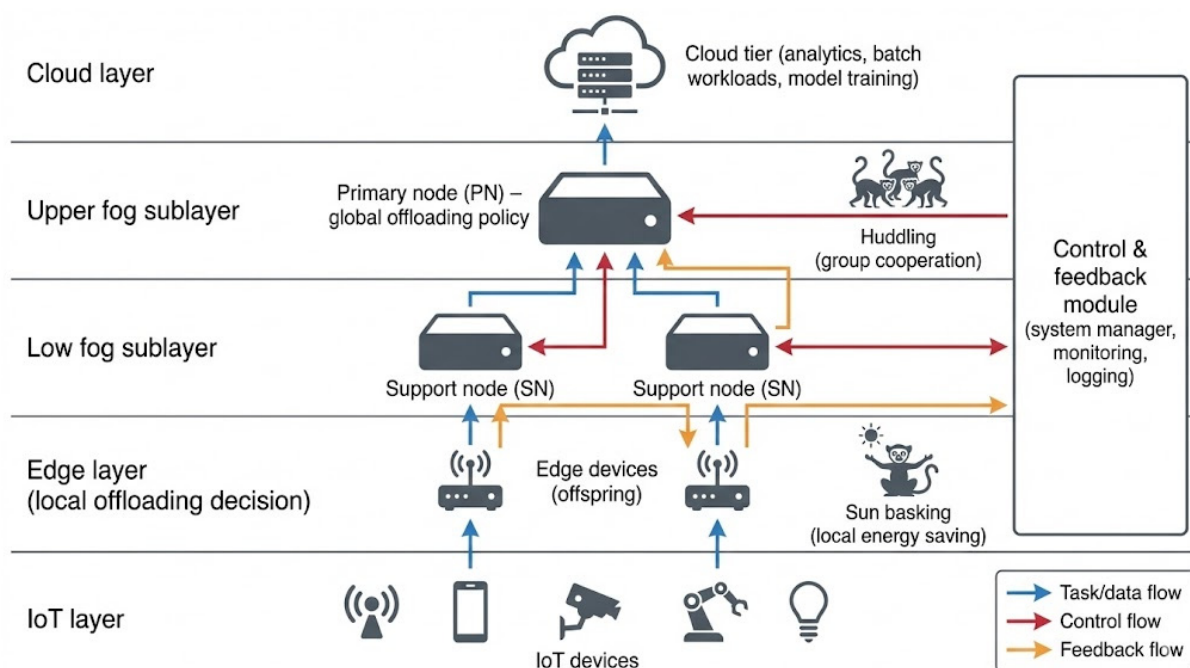


Figure 2. Lemur-inspired edge-fog-cloud infrastructure and node roles.

3.4. Proposed Lemur-Inspired Task Offloading Algorithm

LITO is the algorithmic framework that instantiates the lemur-inspired behaviors described in Section 3.2 on top of the edge-fog-cloud infrastructure defined in Section 3.3. It specifies how IoT devices, edge devices, support nodes, and primary nodes cooperate to decide where tasks are executed and how resources are scheduled. The framework is organized into six main stages: system initialization, system management, local task offloading at edge devices, global offloading decisions at primary nodes, scheduling and execution at support nodes, and feedback and adaptation. Figure 3 presents the high-level, role-based workflow of these stages across the continuum tiers, while Figure 4 depicts the internal component architecture of the LITO framework.

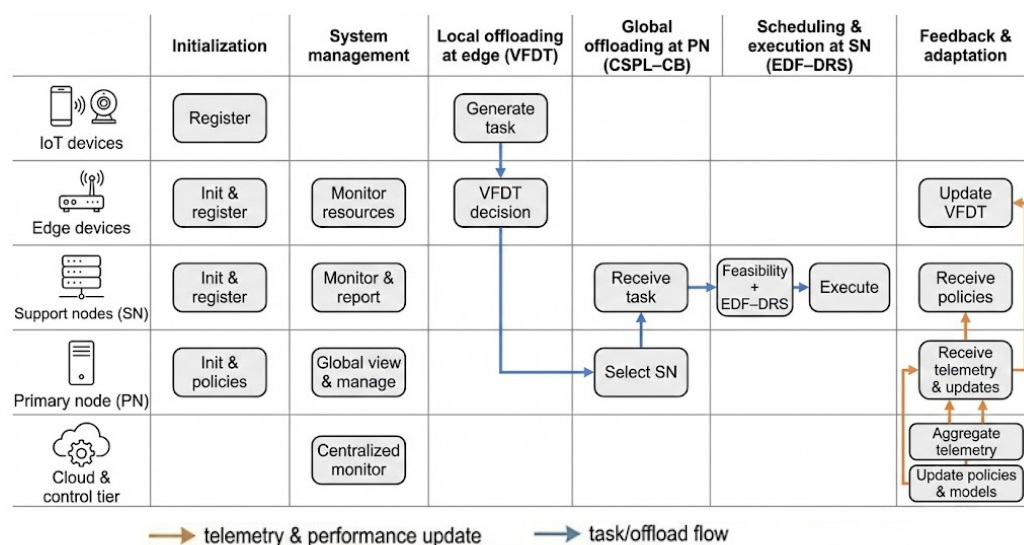


Figure 3. Role-based workflow of the LITO algorithm for decentralized task offloading.

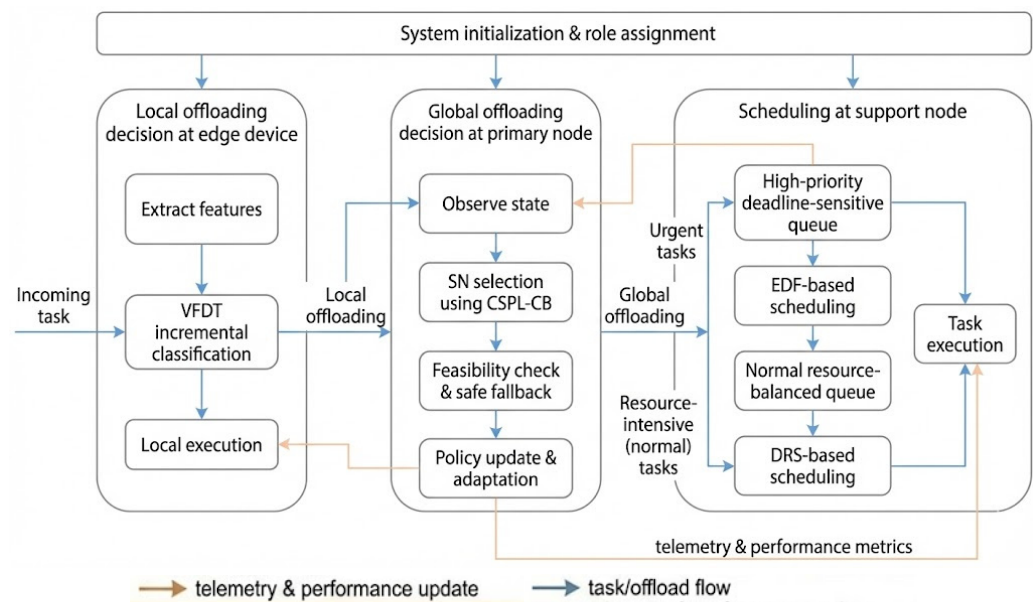


Figure 4. Component architecture of the LITO framework across the edge–fog–cloud continuum.

Following initialization, each node carries out role-specific responsibilities within this workflow. IoT devices generate tasks and forward them to edge devices. Edge devices either execute tasks locally or, guided by Very Fast Decision Tree (VFDT) classifiers, offload them to primary nodes. Primary nodes transform global observations into offloading decisions using the CSPL-CB learner and distribute tasks to suitable support nodes. Support nodes then apply the hybrid EDF–DRS scheduler to execute tasks while balancing deadlines and load. Throughout this process, telemetry is collected and fed back to the coordination and learning components, enabling continual adaptation. Detailed algorithms for each stage are provided in Sections 3.4.1–3.4.5.

Following initialization, each node carries out role-specific resource monitoring, similar to behavioral sensing in lemur groups. EDs extract task-level features and use VFDT classifiers for real-time offloading decisions. PNs collect global telemetry from SN and apply CSPL-CB to refine offloading strategies. SNs implement a hybrid task scheduling mechanism that combines EDF for latency-sensitive tasks with DRS for load-balanced execution.

The LITO framework supports hierarchical task delegation. Tasks generated by IoT devices are first processed by EDs, which either execute them locally or offload them to PN based on VFDT outcomes. PNs aggregate telemetry, select suitable SNs based on CSPL-CB, and check whether execution on the chosen node is feasible. SNs then queue and execute tasks using EDF or DRS, depending on task priority and resource demand. The architecture incorporates two biologically inspired strategies: sun basking, which reflects proactive energy conservation at the edge layer by reducing local execution and preferring resource-rich nodes, and huddling, which represents coordinated task allocation from PNs to SNs on the basis of global offloading decisions for adaptive and balanced workload distribution within the fog layer. After task execution, all nodes submit performance feedback to the PN. This feedback loop allows continual policy refinement and helps maintain responsiveness and resilience under dynamic conditions. Figure 4 illustrates the system architecture of the LITO algorithm.

3.4.1. Initialization

This phase coordinates all components in the network and ensures that each node operates according to its assigned role and responsibilities. It also supports the establishment

of the fog computing environment, role assignment, network performance monitoring, and communication management across all entities in the system.

Initially, the system starts and loads with its configuration parameters to establish a connection with the centralized manager, retrieve essential setup data, and initialize system parameters based on predefined settings. This phase ensures that the network begins in a controlled and consistent state. After initialization, the processing power, storage capacity, and connectivity of each fog node are assessed to assign roles as PNs or SNs. EDs are designated as offspring nodes by default because of their resource constraints and reliance on fog nodes for task offloading. This structured role allocation mirrors the hierarchical organization observed in lemur groups and ensures that each node plays a role aligned with its capabilities. Algorithm 1 illustrates the initialization process.

Algorithm 1: Initialization

Input: System configuration files, list of nodes, resource profiles of each node

Output: Fully initialized fog system with assigned roles (PN, SN, ED)

```

1  Initialize the fog system by System Manager
2  Retrieve configuration data
3  for each fog node  $m_j$  do
4      Evaluate available resources
5      Assign role: PN (female lemur), SN (male lemur), or ED (offsprings)
6  end
7  Establish secure communication channels
8  Configure periodic monitoring and reporting:
   each SN and ED send its state vector (CPU, memory, queue length, energy, link quality)
   to the PN every  $\Delta t$  time units
  
```

3.4.2. System Management

Once roles are established, a system-wide monitoring solution is activated to continuously collect data on network traffic, node health, task execution, and resource utilization. This monitoring helps detect anomalies or performance bottlenecks early and supports stable and efficient operation. Secure communication channels are also established and maintained between the centralized system manager and all nodes so that the manager can distribute updated roles, configurations, and tasks, and nodes can report their status or negative feedback. This bidirectional communication keeps the network synchronized.

The system manager is responsible for role adjustments, global monitoring, and reporting. Based on information about node status, it can reassign roles when a node degrades or new resources are added. It also produces system-wide reports that summarize task completion rates, node health, and overall resource utilization, which can guide future optimization.

3.4.3. Local Task Offloading in EDs Using VFDT

Local offloading enables EDs to handle their resources intelligently by deciding when to offload tasks to fog layer and when to execute them locally, while balancing local execution and remote processing. By offloading tasks appropriately, EDs can conserve energy, prolong operational lifetime, and improve application performance.

Existing ML classifiers, such as traditional Decision Trees, Random Forests, or Support Vector Machines, often require high memory and offline training and cannot efficiently manage streaming task data. This drawback leads to increased delay in dynamic environments. Moreover, they do not cope well with continuous adaptation to varying workloads or network conditions. Hence, VFDT (also called as Hoeffding Tree) is leveraged since it

processes streaming data in real time, learns incrementally, and offers statistically confident decisions for dynamic task offloading with reduced computational cost. It also helps local task offloading to adapt continuously to changing workloads, network conditions, and device constraints within the fog environment. This adaptive mechanism is conceptually inspired by lemur behaviors such as sun basking and huddling, where lemurs regulate their energy and thermal balance (analogous to local offloading for energy optimization).

Initially, tasks are inspected to determine their suitability for offloading. Each task t_i is analyzed in terms of CPU cycles, energy demand, memory requirement, and deadline. Tasks are then combined with metadata such as execution size, context, and priority. VFDT incrementally constructs a decision tree from the streaming task data, which makes it suitable for EDs where decisions must be made quickly and repeatedly.

When a t_i instance arrives, its feature vector X_t (as in Equation (1)) is fed to the VFDT to determine the class label: Offload (the task is offloaded to a PN) or Local Execution (the task is processed locally). The algorithm starts at the root node and moves down the tree by checking the attribute tests (splits) at internal nodes. At each ED m_j , VFDT keeps statistical summaries of the observed attributes at the leaf node. For numerical features, these statistics may include histograms, counts, or summary values essential for calculating split criteria during incremental learning. The decision-making process depends on Information Gain (G), which is given by:

$$G(T, A) = H(T) - \sum_{v \in \text{values}(A)} \frac{|T_v|}{|T|} H(T_v) \quad (10)$$

where $H(T)$ indicates the task dataset T 's entropy at m_j ; T_v indicates the subset split by attribute A ; "values" (A) indicates the set of possible values or ranges that A can take. The Hoeffding Bound (ϵ) [43] is utilized to ensure statistically reliable splits. ϵ defines the confidence that the best attribute chosen after observing n instances is similar to the one that would be chosen with infinite data. ϵ is based on the range of the splitting criterion, the number of observed instances n , and the confidence parameter δ . The split criterion is applied:

If the best attribute G_{best} is better than the second best G_{2nd} by at least ϵ (i.e., $G_{best} - G_{2nd} > \epsilon$), split on the best attribute.

If G_{best} and G_{2nd} are within ϵ (i.e., $G_{best} - G_{2nd} \leq \epsilon$), the standard VFDT would wait for more samples, but, here, a tie-break split is done to reduce decision latency.

This split rule aids in avoiding premature decisions by ensuring that attributes with nearly equal gain are not selected incorrectly from limited samples. Using ϵ guarantees reliability with probability $1 - \delta$, whereas the selection of ϵ balances speed and accuracy. It ensures that VFDT makes confident splits only when sufficient samples have been gathered, reducing redundant complexity and enhancing computational efficiency. Until the task instance reaches a leaf node that does not have any splits, this process continues. Figure 5 illustrates the process of local decision making using VFDT.

When a decision is made to offload a task, the ED sends the packaged task data, metadata, and execution instructions to the PN over a secure, low-latency communication channel. Tasks that are classified as Local Execution typically require fewer CPU cycles, lower energy, and have short deadlines. These are executed locally to minimize communication overhead and response delay while conserving energy. Scheduling based on Earliest Deadline First (EDF) ensures that delay-sensitive tasks are executed quickly.

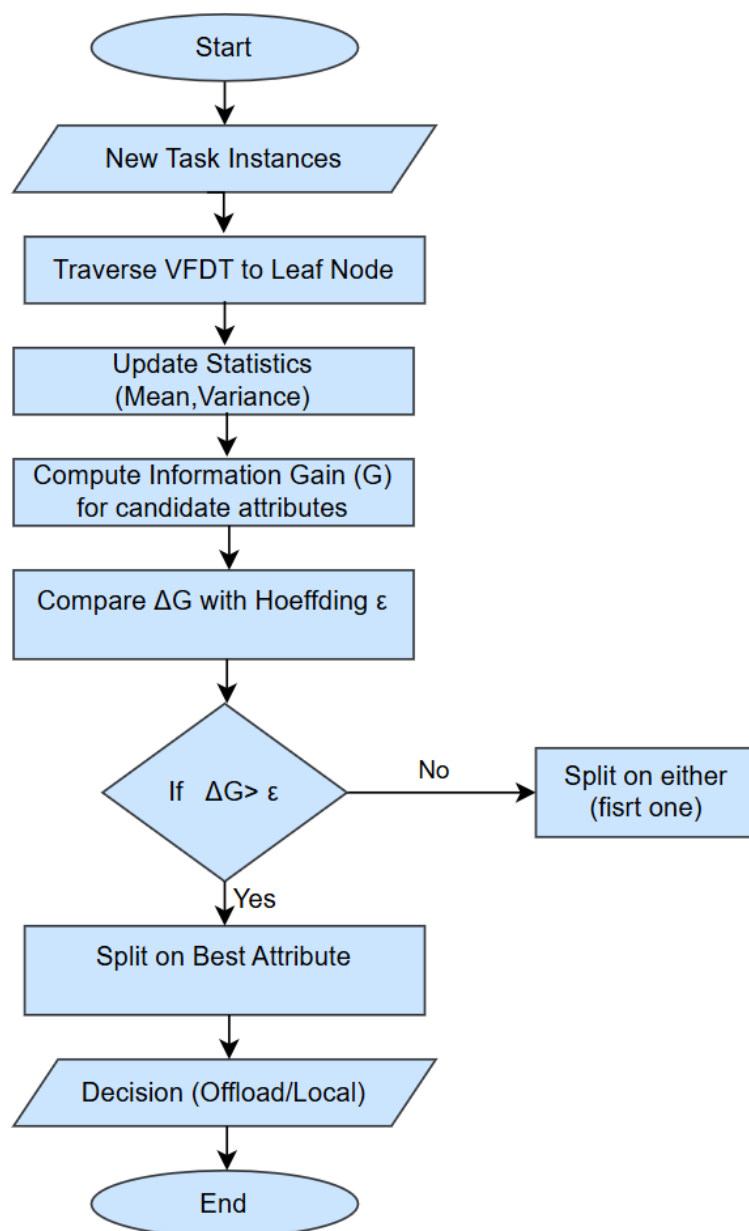


Figure 5. Local decision making using VFDT.

EDs also monitor offloaded tasks and manage responses from fog nodes. When the execution result is received, EDs integrate it into local applications. Each ED records performance metrics such as task completion time, network overhead, energy usage, and task success/failure rate, and feeds them back into the VFDT model. This mechanism allows the classifier to adapt over time to the dynamic fog computing environment. Since VFDT is an incremental learner, it improves its decision quality over time as more task instances are processed, without requiring retraining from scratch. Algorithm 2 describes the local offloading mechanism.

Algorithm 2: Local Offloading at EDs**Input:** Incoming task t_i **Output:** Local offloading decision: Local execution or offload

```

1  for each  $t_i$  do
2      Extract feature vector  $X_t$ 
3      Feed  $X_t$  to VFDT
4      while traversing VFDT do
5          if Hoeffding Bound  $\epsilon$  is satisfied then
6              Split the node
7          end
8      end
9      Classify  $t_i$ : offload or Local Execution
10     if Decision = Local Execution then
11         Schedule the task using the EDF-based queue
12     else
13         Offload securely to PN for global decision-making.
14     end
15     Update VFDT incrementally with the observed outcome
16 end
17     Send periodic summary reports from each ED to the PN, including local
    queue length, execution latency, energy level, and offloading decisions

```

3.4.4. Continual Supervised Offloading Policy with Contextual Bandit Feedback for PNs

PNs serve as the central intelligence of the fog computing network that aids in making global task offloading decisions while ensuring effective task distribution. Conventional RL methods such as Q-learning, Deep Q-Network, or the Actor-Critic method struggle in fog environments since they usually assume fixed state spaces, static task patterns, or require costly retraining when workload patterns evolve. In addition, they suffer from limited adaptability to non-stationary workloads and catastrophic forgetting. Hence, CSPL-CB is utilized, in which PN learns a policy online, preserving previously acquired knowledge. In contrast to RL-based methods, CSPL-CB does not depend on reward signals. Instead, it employs supervised oracle labels to update the offloading policy.

At each decision instant t , PN observes a compact state that summarizes network/ED/SN status and selects an offloading action that indicates which SN should receive each task. It then executes the decision by instructing SN, monitors outcomes, and updates its policy incrementally to adapt to evolving conditions. Similar to how lemurs regulate their group temperature through sun basking and huddling, the PN dynamically balances workload distribution by directing tasks toward underloaded or energy-efficient nodes while clustering tasks when latency or resource pressure rises. This adaptive mechanism reduces energy dissipation and promotes system stability under changing workloads.

In CSPL-CB, the environment is defined by a tuple $(\mathcal{A}, \mathcal{O}, \rho)$, where $\mathcal{A}$ (or ΔY) is the discrete set of possible offloading actions (candidate SNs), $\mathcal{O}$ is the set of observations $(X \times Y)$ where the observation space contains context vectors X_t and Y_t is oracle labels available only at update time, and $\rho(\cdot)$ captures the non-stationary observation dynamics. At decision time t , PN receives the observations $O_t = (X_t, Y_t)$ in which X_t is given by,

$$X_t = f(X_{mj}, X_{nls}, X_{ti}, X_l) \quad (11)$$

X_t aggregated vectors from SNs (X_{mj}), such as available resources, utilization levels, queue lengths, and current acceptance rates. By utilizing exponentially weighted moving averages (EWMA) of latency and available bandwidth for each SN, network link statis-

tics features (X_{nls}) are captured by the agent. Task-specific features (X_{ti}) in Equation (1) and short local signal features (X_l) like current PN load and an energy budget indicator are also added. $f(\cdot)$ is the function that performs normalization and compression to a fixed-size representation.

For each task t_i , PN predicts a probabilistic offloading distribution over available SNs: $a_t \sim \pi(\cdot|X_t)$, $a_t \in \mathcal{A}$, where π indicates the stochastic policy. Entropy regularization is utilized during action sampling to maintain exploration. Prior to decision-making, each sampled action is fed through a feasibility filter to ensure that constraints like deadline feasibility, queue saturation limits, and minimum resource requirements are met.

$$a_t^F = F(X_t, a_t) \quad (12)$$

If they are not met, a safe fallback is applied that assigns the task to the adjacent underloaded SN or queue task until the next offloading decision slot. This fallback can be compared to huddling behavior, in which nodes cluster computational tasks for mutual stability and minimal energy loss under constrained conditions, while ensuring overall system endurance. PN sends task assignment messages that include task metadata and deadlines to the chosen SN for execution. After execution, the PN collects an outcome tuple for each t_i :

$$O_t^{ti} = (L_t^{ti}, E_t^{ti}, S_t^{ti}, \Delta U_t^{mj}) \quad (13)$$

where L_t^{ti} is the task latency at T; $S_t^{ti} \in \{0,1\}$ is the success indicator; E_t^{ti} is the energy consumed by the task; ΔU_t^{mj} is the change in node utilization after task execution ($U_{t+1}^{mj} - U_t^{mj}$). The complete contextual feedback available to the PN at t is given by:

$$O_t = \{O_t^{ti} | t_i \in T\} \quad (14)$$

This contextual feedback, including energy, latency, success/failure signals, and utilization change, constitutes the bandit feedback required by CSPL-CB. The PN then utilizes this feedback to update its supervised policy in a continual fashion. This helps in enabling adaptive and constraint-aware task offloading under non-stationary workloads.

To enable supervised continual learning, the ground-truth SN label Y_t is generated through a best-feasible oracle that assesses all candidate SNs under energy usage, deadline feasibility, and queue delay. The oracle chooses the SN minimizing a weighted cost function, and this offers the training signal:

$$Y_t = \arg \min_{a \in \mathcal{A}} C(a | X_t) \quad (15)$$

This solves the log-loss definition by offering a clear, defensible correct action.

The policy is updated through the negative log-loss, which is given by:

$$R_{t+1} = -\log P_\pi(Y_t | X_t) \quad (16)$$

which quantifies how well the agent predicts the oracle-optimal action. The long-run performance objective is given by:

$$r^\pi = \liminf_{T \rightarrow \infty} \mathbb{E} \left[\frac{1}{T} \sum_{t=0}^{T-1} R_{t+1} \right] \quad (17)$$

Maximizing r^π obtains sustained accuracy in choosing near-optimal SNs under dynamic workloads.

To mitigate catastrophic forgetting under emerging workloads, the PN preserves a bounded rehearsal buffer of representative previous observations and periodically implements incremental updates with experience replay and regularization constraints. This mechanism enables the policy to combine new task patterns without ignoring historically valuable behaviors.

Since tasks are latency-critical, both inference and updates should satisfy decision latency per step $\leq L_{\max}$, while ensuring the PN makes timely offloading decisions while continually optimizing its policy. By utilizing this continual supervised policy learning mechanism, the PN sustains deadline-aware, energy-efficient, and utilization-balanced performance across non-stationary fog environments. Algorithm 3 illustrates the global offloading approach.

Algorithm 3: Global Offloading Using CSPL-CB at PN

Input: Offloaded tasks; current system state X_t ; policy parameters

Output: Selected SN; updated policy

```

1.  for each decision epoch  $t$  do
2.      Observe  $X_t$  from PN and SNs
3.      Sample action  $a_t$ 
4.      if feasibility  $F(X_t, a_t) = \text{false}$  then
5.          Implement safe fallback
6.      end
7.      Perform action  $a_t$  and gather contextual bandit feedback  $O_t^{ti}$ 
8.      Generate a supervised label through the best-feasible oracle  $Y_t$ 
9.      Calculate supervised loss  $L_t$ 
10.     Store  $(X_t, Y_t)$  in replay buffer  $M$ 
11.     Update policy incrementally using mini-batches from  $M$ 
12.     if decision latency  $> L_{\max}$  then
13.         Prune action space to satisfy real-time constraints
14.     end
15. end

```

Coordination Between Local and Global Decision Layers: The coordination between local and global decision layers follows a hierarchical decision refinement mechanism. ED executes a binary decision (Local vs. Offload) through VFDT on the basis of local features X_i . When the decision is Local, execution remains at the ED without any global interaction. When the decision is Offload, task descriptor X_i is sent to the PN together with local context metadata to perform multi-class global optimization using the CSPL-CB policy. Therefore, VFDT performs coarse-grained filtering, whereas CSPL-CB performs fine-grained routing amongst fog nodes. Decision conflicts are circumvented using mechanisms like role separation and feasibility filtering. EDs do not choose specific SNs; they only decide whether to offload. SN selection authority is preserved at the PN layer. Additionally, the PN applies the feasibility filter (Equation (12)) before final task assignment to enforce deadline, capacity, and utilization constraints. When the chosen SN becomes infeasible because of state changes, a safe fallback strategy allocates the task to the next best possible SN. This hierarchical authority structure guarantees global constraint compliance by avoiding contradictory routing decisions. After the execution of tasks, outcome feedback $(L_t^i, E_t^i, S_t^i, \Delta U)$ is sent back to both PN and ED. The PN updates the CSPL-CB policy, whereas the ED updates its VFDT model by utilizing the execution results. This dual-feedback mechanism confirms that local offloading decisions gradually align with global optimization trends by minimizing long-term decision divergence.

3.4.5. Scheduling Offloaded Tasks in SNs

Once the task is sent to SN from PN, SN takes full responsibility for its local scheduling and execution. Scheduling is performed with the help of a hybrid EDF-DRS (Earliest Deadline First-Dynamic Resource Scheduling) scheduler, tasks depending on resource

availability, priority, and execution efficiency. This scheduler analyses resource requirements of tasks, prioritizes tasks based on task size, deadlines, or dependencies, and orders the execution to meet deadlines. In this hybrid approach, two logical queues, such as a high-priority deadline-sensitive queue and a normal resource-balanced queue, are utilized. This two-queue split allows SN to balance strict timing guarantees and adaptive resource distribution.

Initially, tasks are analyzed for their resource requirements and deadlines. Tasks with strict deadlines are kept in a high-priority deadline-sensitive queue (Q_{EDF}) that uses EDF scheduling. In contrast, tasks that are resource-intensive or have relaxed timing requirements are kept in the normal resource-balanced queue (Q_{DRS}) that uses DRS scheduling [44]. Q_{EDF} manages urgent tasks and is frequently resorted to maintain deadline order, while Q_{DRS} manages less time-critical tasks and is resorted to only periodically to conserve computation. Like how lemurs dynamically switch between sun basking for conserving energy and huddling to maintain warmth and group balance, the hybrid scheduler switches between Q_{EDF} and Q_{DRS} to maintain thermal-equilibrium-like stability in resource allocation, prioritizing urgent processes when needed. It preserves energy efficiency during low-load conditions. This queue separation aids in minimizing redundant resorting, ensuring that the hybrid scheduler quickly prioritizes urgent tasks with reduced overhead.

For each t_i with deadline D_{t_i} and remaining workload W_{t_i} at time t , its urgency factor (UF_{t_i}) is given by:

$$UF_{t_i} = \frac{W_{t_i}}{D_{t_i} - t} \quad (18)$$

Ensure that $D_{t_i} - t > 0$. If UF_{t_i} exceeds a predefined threshold, the task is kept in Q_{EDF} ; otherwise, it is kept in Q_{DRS} . Q_{EDF} is always given priority in execution. Tasks in this queue are scheduled in increasing order of their deadlines to ensure that SN minimizes the risk of missed deadlines. For any two tasks t_i and t_j in Q_{EDF} , the ordering follows: $t_i < t_j$ when $D_{t_i} < D_{t_j}$. By considering available CPU, memory, and bandwidth at each scheduling interval, Q_{DRS} operates adaptively. Each task in Q_{DRS} is assigned a proportional share of resources depending on its requirements. When the available computational capacity of m_j is C_f , and a task t_k requires R_k units of computing, its assigned share S_k is determined by:

$$S_k = \frac{R_k}{\sum_{j \in Q_{DRS}} R_j} \times C_f \quad (19)$$

This proportional allocation guarantees fairness among non-urgent tasks, preventing resource starvation. This mechanism reflects lemur huddling behavior in which shared heat represents equitable distribution of limited energy among members, while ensuring that no individual suffers a resource deficit. Task execution proceeds by dispatching all possible Q_{EDF} tasks in order of deadline, using the required portion of the available computational resources. Based on the computed shares, the remaining computational capacity is then distributed amongst tasks in Q_{DRS} . When the EDF set surpasses the free capacity of SN, the scheduler leverages proportional reduction within Q_{EDF} , providing higher weight to tasks with the earliest deadlines. This hybrid method ensures that critical tasks are prioritized while making effective utilization of available resources for normal tasks.

During execution, SN continuously monitors task completion time, actual resource consumption, and deviations from estimates. When tasks from Q_{DRS} start to lag excessively, SN can temporarily elevate them into Q_{EDF} by maximizing their UF_{t_i} . On the other hand, DRS can expand allocations to accelerate heavier tasks when resources are underutilized. This behavior is similar to changing from huddling to sun basking when environmental energy levels allow higher activity for thermal and computational balance. After execution, task latency, success indicator, energy consumed and change in node utilization are reported back to the PN, enabling better coordination in future offloading decisions.

Algorithm 4 shows the offloading process in SNs.

Algorithm 4: Scheduling of Offloaded Tasks in SNs

Input: Tasks assigned by PN; Urgency factors UF_{t_i}

Output: Ordered execution of tasks; balanced resource distribution; outcome tuples

```

1  for each  $t_i$  at SN do
2      Calculate  $UF_{t_i}$ 
3      if  $UF_{t_i} > \text{threshold}$  then
4          Place  $t_i$  in  $Q_{EDF}$ 
5      else
6          Place  $t_i$  in  $Q_{DRS}$ 
7      end
8  end
9 end
// EDF Queue Management//
10 Sort  $Q_{EDF}$  based on deadlines
11 while  $Q_{EDF} \neq \emptyset$  do
12     Execute the earliest-deadline task while coordinating nodes through
    adaptive load balancing inspired by lemur huddling behaviour
13 end
// DRS Queue Management//
14 if the reschedule interval is reached then
15     Resort  $Q_{DRS}$ 
16 end
17 for each task  $t_k \in Q_{DRS}$  do
18     Assign proportional share  $S_k$ , considering workload equilibrium for
    sustaining node efficiency
19 end
20 if combined demand of  $Q_{EDF} > \text{available } C_f$  then
21     Proportionally reduce allocations across tasks using  $S_k$ 
22 end
23 Report outcome tuples back to PN

```

4. Evaluation Methodology

To assess the viability of the proposed LITO algorithm, the iFogSim toolkit [46], an extension of CloudSim [47], is employed, which is a widely utilized open-source Java-based toolkit developed for fog computing simulations. It depends on a distributed dataflow overview and functions based on the Sense–Process–Actuate paradigm. iFogSim allows event-driven interactions amongst edge and fog components, enabling the modeling of real-time computation offloading scenarios. It assumes distributed task generation and processing across heterogeneous IoT and fog nodes. This framework offers vital classes like Sensor, EdgeDevice, Actuator, Tuple, PhysicalTopology, and Application, which together enable the development of customized edge/fog computing environments with several heterogeneous nodes.

Workloads are synthetically generated with probabilistic distributions for memory, CPU cycles, and arrival times, allowing controlled exploration of light to high-intensity task scenarios. The network model integrates realistic bandwidth and latency parameters for IoT-to-Edge and Edge-to-Fog links, by capturing transmission delays under diverse data rates. Parameter sensitivity is analyzed systematically by changing task complexity,

workload size, data rates, and node topologies, enabling evaluation of robustness, SLA adherence, and performance of LITO under both nominal and stress conditions.

4.1. Simulated Scenarios

To evaluate the scalability and responsiveness of the LITO algorithm under varying edge workloads, we define four simulation scenarios as shown in Table 3. Each scenario increases the number of IoT devices and associated Edge Devices, while maintaining a constant fog-layer structure (PNs, SNs, and System Manager). Fog nodes remain constant since the focus is on assessing offloading efficiency of LITO under increasing edge workloads, instead of scaling fog infrastructure. This incremental topology supports systematic analysis of LITO's behaviour across small- to large-scale deployments. The scenarios are defined as follows:

- Lightweight Edge Load (Configuration 1): Models a small-scale environment with 16 IoT devices and 8 EDs, representing use cases such as room-level smart automation.
- Moderate Edge Load (Configuration 2): Doubles the edge scale to 32 IoT devices and 16 EDs, representing a smart building or floor.
- Dense Edge Load (Configuration 3): Includes 64 IoT devices and 32 EDs, approximating a complex system like a smart industrial floor or campus.
- High-Density Edge Load (Configuration 4): Stress-tests LITO with 128 IoT devices and 64 EDs, simulating dense city blocks or wide-area fog deployments.

In all configurations, the fog layer includes 1 PN, 3 SNs, and 1 System Manager, which remain constant across different scenarios. This configuration spectrum enables detailed evaluation of how LITO handles increasing device density and varying levels of workload intensity. The corresponding topology is illustrated in Figure 6. Simulation configurations detailing the number of IoT devices, EDs, PNs, SNs, and system managers are illustrated in Table 3.

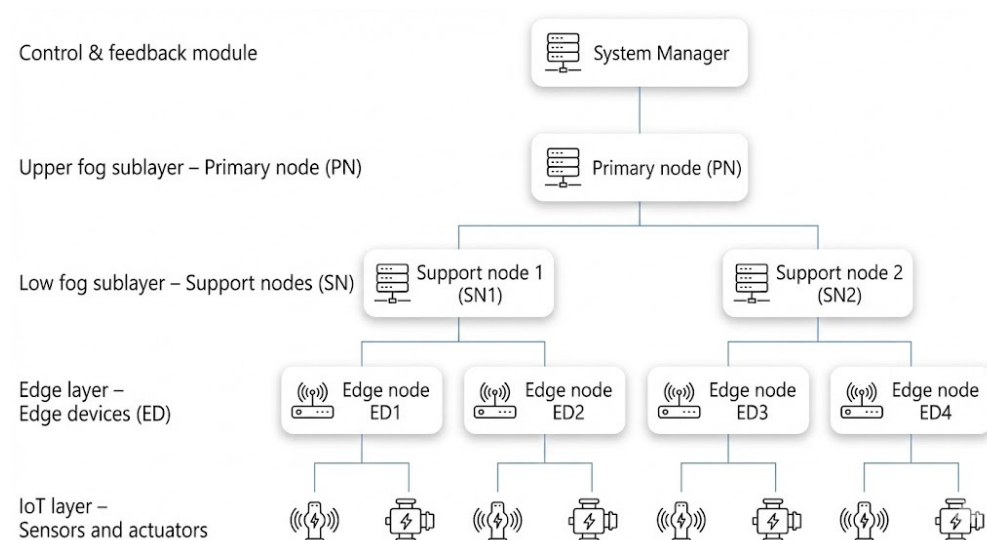


Figure 6. Topology illustrating hierarchical interactions among system devices.

The hierarchical topology utilized in all simulation scenarios is summarized as follows:

- IoT Layer: variable, ranging from 16 to 128 IoT devices based on the configuration.
- Edge Layer: variable, ranging from 8 to 64 EDs.
- Fog Layer: Fixed, with 1 PN, 3 SNs, and 1 system manager across all configurations to isolate the impact of increasing device density and edge workload on the performance of LITO.

Table 3. Simulation configurations detailing the number of IoT devices, EDs, PNs, SNs, and system managers.

Parameter/Category	IoT Devices	EDs (Offspring Nodes)	Fog Devices (SN + PN)	System Manager
Tuple Type	Sensor: ENV_SENSOR Actuator: DISPLAY	—	—	—
Distribution	Sensor: Based on workload size	—	—	—
Latency	Sensor: 1.0 ms (to gateway/edge) Actuator: 1.0 ms (response time to user)	To Parent (SN): 2 ms (Uplink)	Support → Primary: 5 ms Primary → Registry: 10 ms (Uplink)	System Manager → Nodes: 0–10 ms (Downlink)
Node Name	—	Edge	SN: support- PN: primary-	System Manager
MIPS	—	10,000	SN: 30,000 PN: 50,000	20,000
RAM	—	8000 MB	SN: 20,000 MB PN: 40,000 MB	16,000
Uplink/Downlink Bandwidth	—	4000 Mbps	SN: 8000 Mbps PN: 10,000 Mbps	5000 Mbps
Level	4	3	SN: 2 PN: 1	0
Rate per MIPS	0	0.01	0.01	0
Busy Power	2 W	100 W	SN: 150 W PN: 200 W	120 W
Idle Power	1 W	60 W	SN: 90 W PN: 100 W	80 W
CPU usage cost (G\$/s)	0.05	0.08	SN: 0.22; PN: 0.35	0.3
Memory usage cost (G\$/MB)	0.005	0.008	SN: 0.022; PN: 0.035	0.03
Bandwidth usage cost (G\$/MB)	0.01	0.015	SN: 0.025; PN: 0.03	0.02
Modules Deployed	task_generator	—	primary_planner; support_scheduler	system_manager
Functional Role	Task generators	Local executors	PN: Global controllers SN: Mid-layer schedulers	System manager

The fog layer is actively involved in task execution and coordination across all configurations. PNs act as global planners to make offloading decisions using the CSPL-CB mechanism, but do not directly complete tasks. All tasks offloaded from EDs are sent to SNs, acting as the actual execution entities in the fog layer. Therefore, PNs execute policy refinement and coordination, whereas SNs manage scheduling and computation of offloaded tasks. This separation of execution and planning roles remains consistent across all scenarios.

In the experiments, the weighting coefficients in Equation (2) are fixed as $\alpha = 0.5$, $\beta = 0.3$, and $\gamma = 0.2$ to prioritize latency, considering energy consumption and utilization imbalance. These values are chosen to ensure balanced trade-offs amongst conflicting objectives and reflect latency-sensitive fog applications. The weights are normalized so that $\alpha + \beta + \gamma = 1$.

In Table 3, the simulation parameters are chosen to emulate a realistic hierarchical fog computing environment with gradually increasing computational capacity, bandwidth, and energy consumption from EDs to PNs. The latency values (1–10 ms) show proximity-based communication delays reported in MEC and fog simulation studies. In contrast, MIPS and RAM configurations follow a tiered scaling pattern, indicating constrained EDs and high-capacity fog nodes. Power and cost coefficients are proportionally allocated to

capture realistic differences in energy consumption and pricing across different layers. The proposed framework's robustness is assessed under changing dynamic task arrivals and workload intensities, while ensuring that performance trends are not based on a single static configuration.

4.2. Synthetic Workload Generation

To evaluate the LITO framework under a wide range of conditions, the experiments employ a fully synthetic workload generator designed to model realistic, heterogeneous resource demands and dynamic task arrivals across a multi-layer IoT-fog architecture. While real data center traces offer valuable insights, they often lack the flexibility and granularity required to systematically test decentralized offloading behaviors across diverse deployment scenarios. In contrast, a synthetic approach allows controlled generation of workloads that reflect a broad spectrum of edge and fog environments, including underexplored or future-oriented configurations. Inspired by the Google Borg cluster workload traces (ClusterData2019/trace version 3; May 2019) [48], the synthetic workload is generated to reflect realistic large-scale computing patterns. Each task, node, and network state is modeled with key compute, memory, I/O, timing, bandwidth, latency, and execution outcome attributes. The simulation environment a hierarchical IoT-edge-fog topology whose computational capabilities are parameterized on the basis of typical edge and fog hardware specifications. The number of active nodes in each layer is given in Section 4.1. Tasks originating from IoT devices are either processed locally on EDs or offloaded to fog nodes based on network conditions and resource availability. Bandwidth, network latency, and energy consumption are simulated for IoT → Edge → Fog links. Performance is assessed in terms of vital metrics like task latency, energy consumption, makespan, resource utilization, throughput, success rate, and offloading ratio, thereby providing a controlled yet realistic environment to test the efficiency of the proposed framework.

Each simulation scenario was repeated 10 times using different random seeds. Service times were generated from the specified task-complexity distributions, while task arrivals followed a Poisson process with exponentially distributed inter-arrival times. Reported results are means across runs; error bars indicate ± 1 standard deviation over the 10 runs.

4.3. Experimental Variables

The experiments were designed to investigate the impact of the following main factors:

- Workload Size (Tasks per Second): Tested at 100, 200, 300, and 600 tasks per second to simulate different levels of sensor activity and user demand.
- Task Complexity (CPU cycles): Measured at 10, 20, 30, 40, and 50×10^6 CPU cycles, representing a range of task types, from lightweight sensor processing to compute-intensive video analytics or anomaly detection.
- Network Bandwidth (Mbps): Evaluated at 5, 10, 20, 50, 100 Mbps to test the system
- Node Configuration (System Topology): Four hierarchical topologies (Configurations 1 to 4) were designed, each corresponding to one of the simulation scenarios in Table 3. Workload size varies from 60 to 900 tasks per second across different configurations (Table 4). To measure TSR, workload size varies from 60 to 90 tasks per second across different configurations.

Low workload sizes (60–90 tasks/s) were utilized to measure TSR in order to ensure that the system is not saturated and to accurately capture task completion behavior under controlled conditions. On the other hand, overall experiments listed in Table 4 varied workload from 60 to 900 tasks/s (Table 4) to evaluate performance under heavier system loads.

Table 4. LITO results showing the impact of varying IoT, ED, SN, and PN configurations on diverse metrics.

Configurations/Topology	Workload Size (Tasks/s)	Throughput (Tasks/s)	Makespan (s)	Comp. Cost (G\$)	Task Completion Time (ms)	Task Success Ratio	Offloading Ratio	SLA Violation Rate
Lightweight Edge Load	60	52	8.389	139,680.0	19.4	0.97	2.4%	0.03
	120	108	9.758	141,380.0	21.1	0.96	1.2%	0.04
	180	158	9.091	144,520.0	23.5	0.95	0.8%	0.05
Moderate Edge Load	100	92	18.056	149,280.0	22.2	0.94	1.4%	0.06
	200	182	19.228	153,380.0	25.8	0.93	0.7%	0.07
	300	270	20.732	158,760.0	29.6	0.91	0.5%	0.09
Dense Edge Load	180	163	48.072	166,920.0	33.8	0.88	0.8%	0.11
	360	318	53.35	173,280.0	37.2	0.86	0.4%	0.14
	540	468	58.629	181,520.0	41.9	0.83	0.3%	0.17
High-Density Edge Load	300	245	164.929	192,440.0	49.3	0.80	0.4%	0.19
	600	465	171.872	205,760.0	56.7	0.76	0.2%	0.23
	900	675	182.767	221,840.0	63.9	0.72	0.1%	0.26

4.4. Experimental Design

The experimental evaluation was organized as a set of targeted experiments, each modeled to isolate the influence of a specific system factor on LITO's performance under controlled conditions. The experiments were organized as follows:

- Experiment E1: Impact of Workload Size (Lightweight Edge Load): It assesses the system's behavior under increasing workload intensity. The lightweight topology was fixed, whereas the workload size was varied across {100, 200, 300, 600} tasks per second. The data rate was set to 20 Mbps, while task complexity was fixed at 20×10^6 CPU cycles. Performance was evaluated with the metrics such as network usage, communication energy, computation energy, total energy consumption, and average resource utilization (Figures 7–11).
- Experiment E2: Impact of Task Complexity (Lightweight Edge Load): For examining sensitivity to computational demand, task complexity was varied from 10×10^6 to 50×10^6 CPU cycles, whereas the workload size was fixed at 100 tasks per second under Lightweight Edge Load. Moreover, the data rate was set at its default value. It evaluates task latency and offloading ratio (Figures 12 and 13).
- Experiment E3: Impact of Workload Characteristics and Topology: It examines the impact of arrival dynamics and network conditions. Task Success Ratio (TSR), throughput, makespan, computational cost, task completion time, task success ratio, offloading ratio, and SLA violation rate were calculated against varying workload sizes across four deployment configurations given in Section 4.1. Task latency was assessed under diverse data rates {5, 10, 20, 50, 100} Mbps, with workload size fixed at 100 tasks/s under Lightweight Edge Load.
- Experiment E4: Ablation Study on Execution Scenarios: An ablation analysis was performed to evaluate the contribution of different execution strategies, such as ED-only local execution, full offloading, hybrid execution at the edge, SN-only execution, and hybrid SN + ED execution. All the System parameters and workload conditions were held constant, and performance was assessed in terms of throughput and makespan.

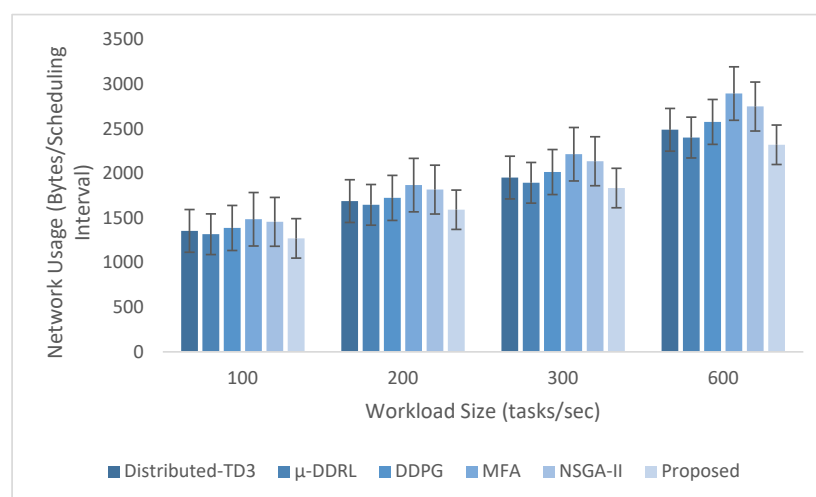


Figure 7. Results of average network usage (bytes per scheduling interval) across varying workload sizes.

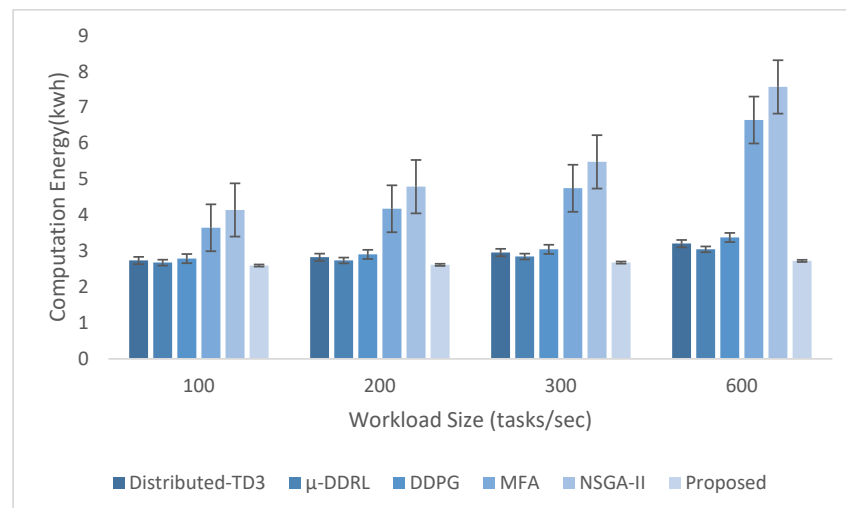


Figure 8. Comparison of the computation energy consumption (kWh) across varying workload sizes.

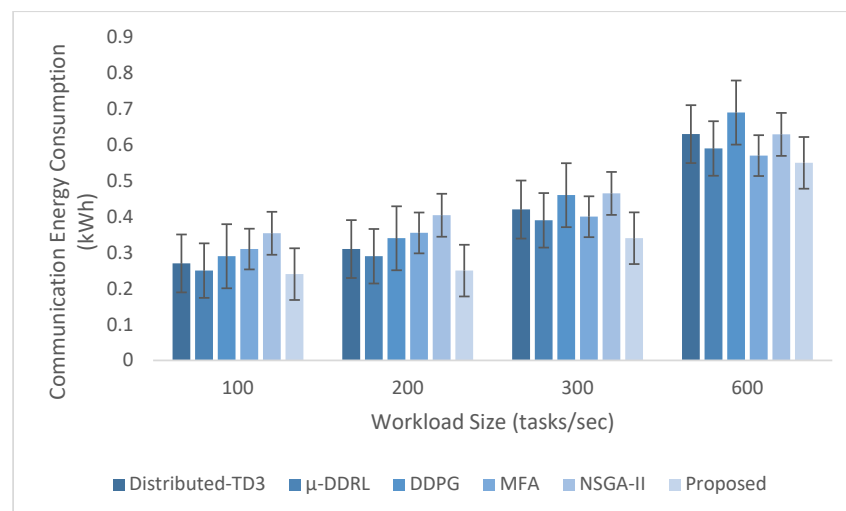


Figure 9. Comparison of the communication energy consumption (kWh) across varying workload sizes.

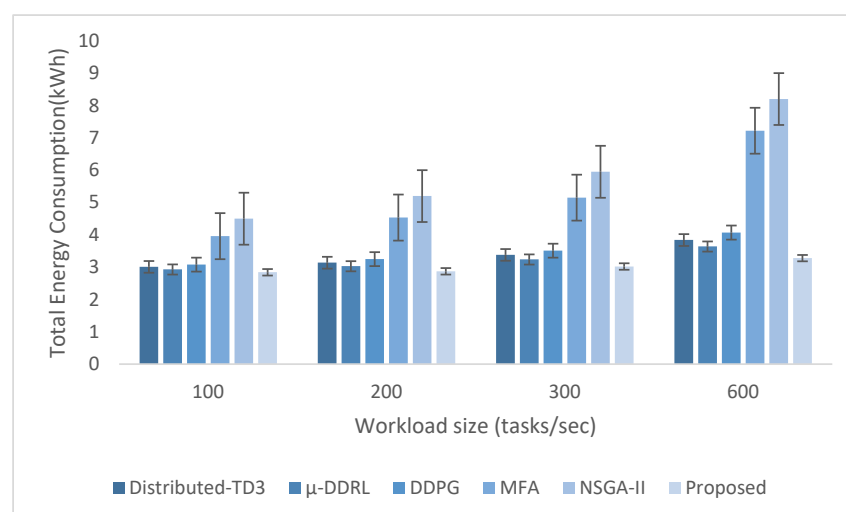


Figure 10. Comparison of the total energy consumption (kWh) across varying workload sizes.

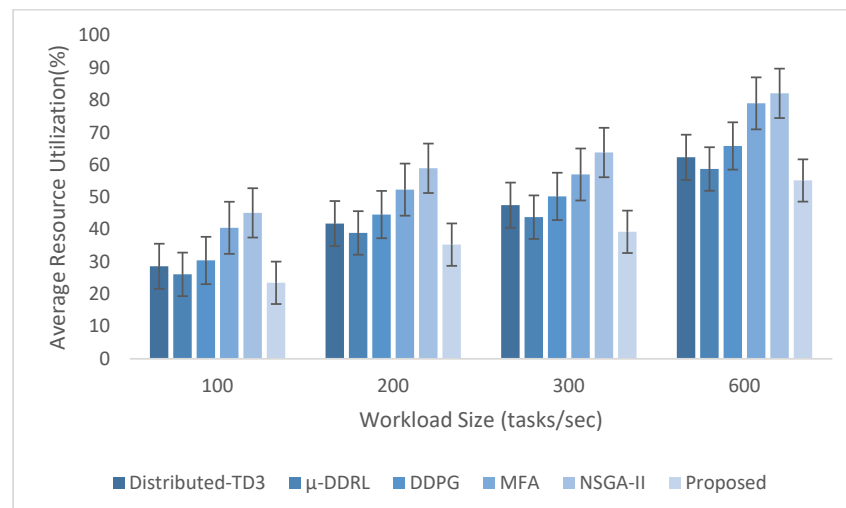


Figure 11. Comparison of the average resource utilization (%) across varying workload sizes.

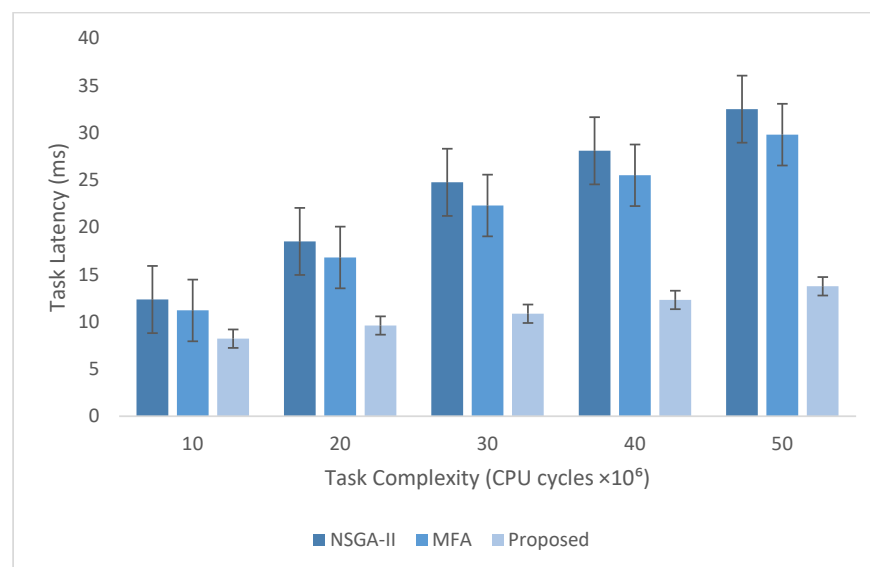


Figure 12. Comparison of the task latency across varying task complexities.

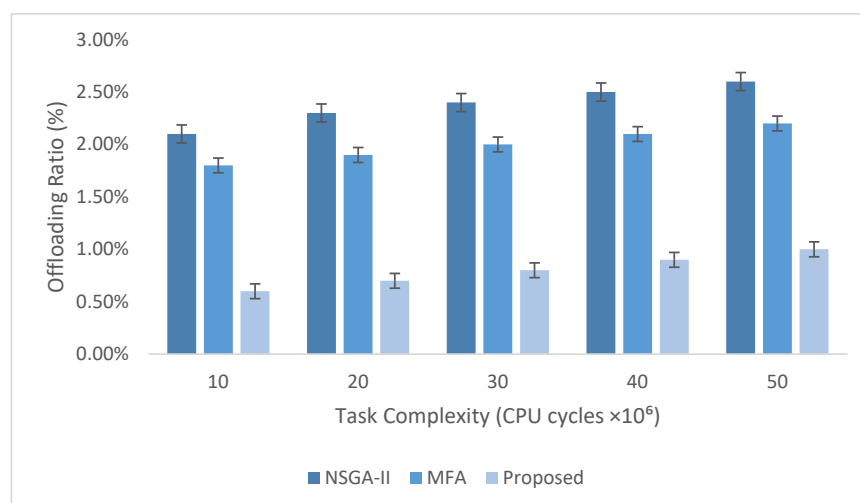


Figure 13. Comparison of the offloading ratio across varying task complexities.

This experiment-centric design aligns the evaluation methodology with the reported results. To account for stochasticity in the simulator and workload generation, each scenario

(algorithm $\times$ workload level) was executed 10 independent times using different random seeds. Reported results correspond to the mean across runs. In figures that include error bars, the bars represent ± 1 standard deviation computed over the 10 runs.

4.5. Performance Metrics

The performance metrics considered are:

- **Network Usage:** It is the average number of bytes sent per scheduling interval during the simulation. It comprises task descriptors, control messages, and data exchanged during task offloading between IoT devices, EDs, and fog nodes. The metric is normalized per fixed time window to enable fair comparison across different workload sizes.

$$\text{Network Usage} = \frac{1}{T_{sim}} \sum_{k=1}^K B_k \quad (20)$$

where B_k indicates the total bytes sent during interval k ; T_{sim} indicates the total simulation time, and K indicates the number of scheduling intervals.

- **Makespan:** It is the total time taken to complete all submitted IoT tasks across the multi-layer LITO overview. Reducing makespan is vital for efficient scheduling and faster task completion under changing workloads.

$$\text{Makespan} = \max_{i \in T} C_i \quad (21)$$

where C_i is the completion time of task i .

- **Average Resource Utilization:** It evaluates how efficiently computational and communication resources of devices are exploited during task execution. It assesses the balance between idle capacity and workload distribution across fog/edge layers.

$$\text{Average Resource Utilization} = \frac{\sum_{i=1}^N TVM_i}{\text{Makespan} \times N} \quad (22)$$

where TVM_i is the total time taken by the i th virtual machine to complete all allocated tasks, and N is the total number of resources.

- **Task Latency:** It is the overall time taken from task generation at an IoT device until its execution is completed and the output is returned. It shows the end-to-end delay caused by a task in the system.

$$L_i = L_i^Q + L_i^T + L_i^E + L_i^R \quad (23)$$

where L_i^Q is the queuing delay before scheduling or offloading the task; L_i^T is the transmission delay caused when sending the task data from an IoT device to the nearby ED; L_i^E is the processing time of the task at the allocated node; L_i^R is the return time.

- **Computation Energy Consumption:** It indicates the total energy used by processing entities like IoT devices, EDs, and fog nodes during the execution of computational tasks.

$$E_{comp} = \sum_{j \in M} \left(E_j^{busy} \cdot T_j^{exec} + E_j^{idle} \cdot T_j^{idle} \right) \quad (24)$$

where M indicates the set of processing nodes; P_j^{idle} indicates the energy consumption of node j in idle state (W); P_j^{busy} indicates the energy consumption in the busy state (W); T_j^{exec} indicates the total execution time at j ; T_j^{idle} indicates the idle time of j .

- **Communication Energy Consumption:** It indicates the total energy required to send tasks and associated data among IoT devices, EDs, and fog nodes during task offloading.

$$E_{comm} = \sum_{(i,j) \in \mathcal{L}} E_{ij}^{tx} \cdot T_{ij}^{tx} \quad (25)$$

where $\mathcal{L}$ indicates the set of communication links; P_{ij}^{tx} indicates the transmission power over the link (i, j) ; T_{ij}^{tx} indicates the transmission time over the link (i, j) .

- **Total Energy Consumption:** It is the total energy utilized by all entities during the complete life cycle of task execution that includes computation and communication activities.

$$E = \sum_{i=1}^N E_i^{Comp} + E_i^{Comm} \quad (26)$$

where E_i^{Comp} is the computation energy utilized for executing task i on IoT device, ED, or fog node E_i^{Comm} is the communication energy needed for sending task i among IoT devices, EDs, and fog nodes.

- **Throughput:** It calculates the rate at which tasks are effectively completed within a given time period, reflecting the processing efficiency of the proposed system.

$$Throughput = \frac{N_{completed}}{T_{sim}} \quad (27)$$

where $N_{completed}$ indicates the number of successfully completed tasks; T_{sim} indicates the overall simulation time. Throughput is expressed in tasks per unit time.

- **Computational Cost:** It is the total expenditure of system resources, such as CPU, memory, and bandwidth, incurred during the communication and execution of all tasks. It is expressed in Grid Dollars (G\$).

$$CC = \sum_{i=1}^N CC_i^{cpu} + CC_i^m + CC_i^{bw} \quad (28)$$

where CC_i^{cpu} is the CPU usage cost; CC_i^m is the memory usage cost; CC_i^{bw} is the bandwidth usage cost.

- **Offloading Ratio:** The offloading ratio measures the ratio of tasks sent from EDs to higher-tier fog nodes for processing, relative to the total tasks generated.

$$Offloading\ ratio\ (\%) = \frac{N_{offloaded}}{N_{total}} * 100 \quad (29)$$

- **SLA Violation Rate (%):** It directly evaluates the real-world acceptability of how frequently deadlines/latency bounds are missed.

$$SVR(\%) = \frac{N_{violated}}{N_{total}} \times 100 \quad (30)$$

where $N_{violated}$ indicates the number of tasks missing the deadline/latency constraint.

- **Task completion time (TCT):** It measures the average time needed for a task to be completed:

$$TCT = \frac{1}{N} \sum_{i=1}^N T_{i,k} \quad (31)$$

where $T_{i,k}$ is the execution time of task T_i at computing node k . When TCT is low, task execution efficiency is improved.

- **Task success ratio (TSR):** It estimates the proportion of tasks completed before their deadlines:

$$TSR = \frac{\sum_{i=1}^N \mathbb{I}(T_{i,k} \leq D_i)}{N} \quad (32)$$

where D_j indicates the deadline; $\mathbb{I}()$ indicates an indicator function. A higher TSR shows a more reliable system.

In this study, we compare LITO with two established multi-objective metaheuristics adapted to the task-offloading formulation, Non-dominated Sorting Genetic Algorithm II (NSGA-II) [39] and the Multi-Objectives Firefly Algorithm (MFA) [36], and three representative distributed DRL-based offloading approaches, including Distributed-TD3 [42],

μ -DDRL [43], and DDPG [44]. These baselines were chosen according to three criteria. First, both algorithms natively support multi-objective or composite optimization. They can be configured with the same objective vector (energy, latency, and resource-utilization balance) and constraints as LITO, which enables a fair comparison at the problem level. Second, these baselines have both been used in prior work on resource allocation and task offloading in cloud/edge/fog environments. They are widely regarded as representative state-of-the-art evolutionary, swarm-based, and distributed learning-based optimization (DRL) approaches, respectively. Third, they belong to two different algorithmic families (evolutionary and swarm-intelligence), so that the performance gains achieved by LITO are not tied to the weaknesses of a single optimization paradigm. The inclusion of distributed RL frameworks confirms that LITO is assessed against adaptive, online decision-making strategies instead of only static/population-based optimizers.

We deliberately avoid adding a large number of closely related metaheuristics (such as further NSGA-II NSGA-II variants or minor DRL modifications), since these show similar convergence behavior under similar objective settings and would not materially modify the comparative insights. Instead, we complement the comparison with external baselines by including an ablation study that isolates the effect of LITO's design choices (local execution only, full offloading, hybrid ED policy, SN-only execution, and the proposed SN + ED collaboration). This combination of strong external baselines and internal ablations provides a focused yet informative evaluation of the proposed framework.

Each baseline is simulated under similar workload distributions, network topologies, and system constraints, ensuring fair comparison. For baseline MFA and NSGA-II, the objectives are reducing task latency and energy consumption, with MFA also maximizing resource utilization. For distributed RL methods, the objective functions integrate latency–energy trade-offs using reward formulations within actor–critic frameworks for online adaptation to workload variations. Decision variables include task-to-node offloading assignments to fog or cloud nodes, subject to constraints on task deadlines, node capacity, cluster size, as well as network bandwidth. Baseline algorithms are implemented in iFogSim using hierarchical fog-cloud topologies by modeling tasks, dependencies, and control loops through AppModule, AppEdge, and AppLoop classes. Balanced resource allocation and fairness are ensured via cluster-head selection, load balancing, and equitable task scheduling across devices. Performance metrics are assessed through repeated simulations. Averaged results are utilized for comparative analysis.

5. Experimental Results

This section provides the results after the evaluation of the proposed LITO algorithm by analyzing its performance under diverse workload sizes, workload characteristics, task complexities, and system configurations. The aim was to analyze how different factors influence the performance metrics related to energy efficiency, latency, resource utilization, and offloading behavior in fog computing environments.

5.1. Impact of Workload Size

Under an increasing workload, LITO maintains controlled growth in network usage without sudden spikes or early saturation, indicating resilient scaling. As shown in Figure 7, network usage increases gradually with workload, reflecting a proportional rise in task offloading and data exchange among IoT devices, EDs, and SNs/PNs. This contrasts with baseline methods, where aggressive offloading under high load can trigger network bursts and congestion. LITO achieves smoother behavior by balancing local execution and offloading decisions through role-based ED–SN–PN coordination. In particular, feasibility filtering prevents forwarding infeasible tasks, reducing unnecessary transmissions and

retransmissions. In addition, the huddling-inspired mechanism enables SNs and PNs to share load under stress, which mitigates localized overload and avoids communication bursts. Overall, the gradual increase in normalized network usage suggests that LITO scales effectively with workload while keeping communication overhead manageable. Practically, this supports admitting larger task volumes without risking network saturation, which is critical in bandwidth-limited fog deployments.

Figures 8–11 show the comprehensive evaluation of the proposed and existing task offloading strategies in terms of computation energy, communication energy, total energy consumption, and average resource utilization under increasing workload intensity. Distributed RL methods also demonstrate adaptive computation reduction when compared to metaheuristic baselines; however, their continuous policy updates and neural network inference lead to additional processing overhead under high workload conditions. Across all workload sizes under Lightweight Edge Load, the proposed LITO framework reliably proves lower computation energy and higher resource efficiency when compared with existing NSGA-II and MFA frameworks, as shown in Figure 8. It indicates that tasks are more capably distributed and processed across EDs and fog nodes. The improvement arises since LITO combines feasibility-aware global offloading with EDF-DRS based local scheduling, minimizing idle cycles and preventing costly re-executions due to deadline violations. In addition, the role-based selection mechanism restricts oscillatory offloading between nodes, lowering processing churn, while cooperative huddling spreads workload among neighbor nodes during transient spikes. These effects together translate into reduced local processing overhead, similar to how lemurs balance metabolic expenditure via coordinated sun basking behaviors. Practically, this indicates that operators can assist higher request volumes without scaling calculate resources or breaching energy budgets.

Under increasing task load, LITO attains reduced communication energy consumption when compared to baselines. Even though distributed RL methods like μ -DDRL and Distributed-TD3 minimize centralized bottlenecks, they have increased communication overhead because of state exchange, policy dissemination, and experience synchronization between agents. As in Figure 9, baselines show increasing energy costs when workload increases, while LITO remains consistently lower even in the high-load region. This is mainly due to feasibility filtering and role-based offloading (ED/SN/PN), which avoid wasted transmissions and oscillatory task migration. Practically, this produces improved energy-efficient task handling in dense edge deployments.

The total energy consumption shown in Figure 10 consolidates these improvements, highlighting that LITO outperforms both benchmark strategies in overall system efficiency. LITO exploits CPU, memory, and bandwidth evenly across the hierarchy, in contrast to MFA which remains limited by clustering heuristics or NSGA-II which over-utilizes hotspots at high loads. The RL-based baselines attain better total energy performance when compared to NSGA-II and MFA as a result of adaptive offloading; however, exploration dynamics and cumulative training costs upsurge overall system energy under heavy load. For real deployments, this implies extended node lifetime, enhanced thermal stability, and higher QoS without hardware scaling, making LITO appropriate for dense urban fog zones and industrial IIoT edges.

In Figure 11, although baselines show high utilization, they indicate localized saturation instead of balanced efficiency. LITO maintains lower energy and network overhead while preserving balanced resource utilization across fog layers. Moreover, LITO attains stable, non-congestive utilization when compared to distributed RL methods by distributing tasks proportionally across edge and fog layers. This approach avoids premature resource exhaustion, maintaining SLA compliance.

5.2. Impact of Task Complexity

Figure 12 illustrates the comparative analysis of task latency across changing task complexities under Lightweight Edge Load with 100 tasks per second. It reveals the superior performance of the proposed LITO framework when compared to the baselines like NSGA-II and MFA. This is mainly due to LITO's mechanism design that includes feasibility filtering minimizing wasted execution and queueing delays; distributed ED/SN/PN role-based selection avoids oscillations and minimizes offloading churn, and adaptive "huddling" behavior spreads overload across nearby nodes, identical to lemur thermoregulation that stabilizes service latency under spikes. Under high-complexity stress conditions, baselines collapse because of centralized decision bottlenecks and lack of overload diffusion, while LITO sustains QoS via balanced resource allocation along with intelligent offloading. Practically, this stability makes LITO appropriate for real-time, delay-sensitive edge applications.

Under increasing task complexities, LITO maintains service reliability and energy feasibility efficiently when compared to NSGA-II and MFA, proving stronger workload resilience in high-stress regions. Figure 13 supports this by highlighting that LITO consistently attains lower offloading ratios when complexity increases. This is because LITO utilizes role-based coordination between EDs and SNs to minimize oscillation and queue churn, feasibility screening at the PN to avoid non-beneficial offloading, and localized huddling to redistribute bursts of load. It ensures that remote execution is only triggered when net performance gains are anticipated. Therefore, LITO identifies scenarios in which local execution is energy-efficient, eliminating redundant remote offloading. In stress regions where baselines depend on aggressive offloading, they experience network contention and increased queuing delays, while LITO's selective offloading of only high-impact tasks produces meaningful reductions in energy and latency. Practically, this allows stable QoS under rapidly increasing workloads without over-provisioning.

5.3. Impact of Workload Characteristics and Topology

Table 4 evaluates the proposed LITO framework under increasing workload intensity across four topologies, reporting makespan, throughput, task completion time, computational cost, TSR, offloading ratio, and SLA violation rate. Table 4 reports LITO-only results to isolate topology and workload effects; baseline comparisons are provided in Figures 7–14. The node configuration of the four topologies is given in Table 3.

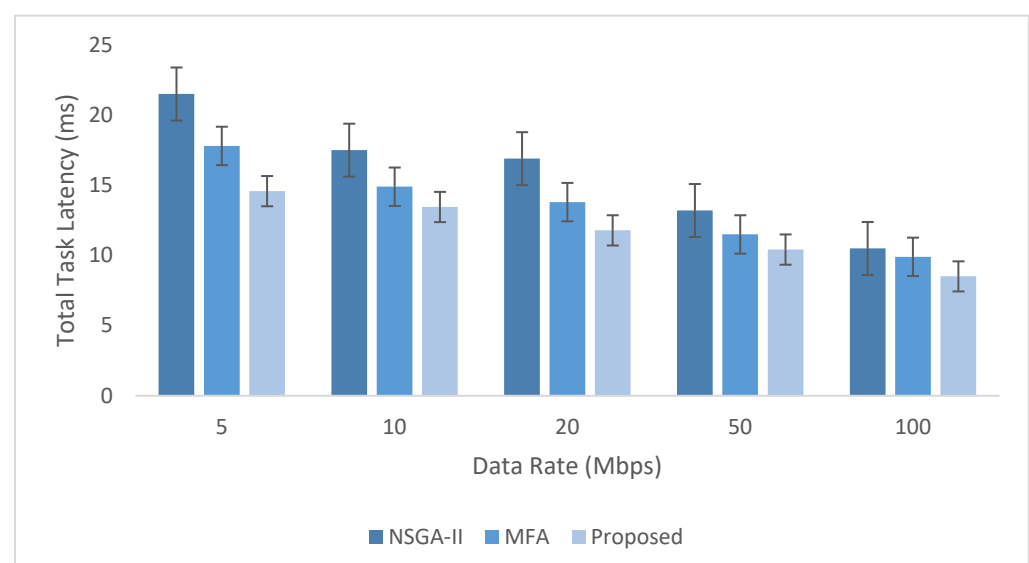


Figure 14. Comparison of task latency with respect to increasing data rates.

Overall, the results indicate that LITO can handle heterogeneous fog resources while maintaining reliable and efficient operation as workload intensity changes. The observed stability under load is consistent with LITO's mechanism-level design (e.g., feasibility-aware filtering and role-based task assignment) rather than ad hoc parameter tuning. In particular, throughput increases with workload up to moderate levels, reflecting effective scheduling and distribution of tasks across EDs and fog nodes. At extremely high workloads, throughput gains diminish in some configurations due to saturation and resource contention—an expected behavior as CPU capacity, bandwidth, and queue limits become binding constraints.

Importantly, LITO mitigates early congestion by filtering infeasible tasks at the ED level, preventing unnecessary forwarding that would otherwise amplify queue buildup and reprocessing. As a result, throughput remains stable until the system approaches hard saturation limits imposed by the underlying topology and resource capacities.

Makespan analysis shows that task completion times scale with workload size, underscoring the growing influence of processing and communication delays. The proposed LITO framework alleviates extended makespan through its hybrid scheduling strategy in which the DRS-based scheduling ensures fair resource allocation for normal tasks, and the EDF queue prioritizes latency-sensitive tasks. This hierarchical task scheduling aids the system in preserving responsiveness, particularly under high workloads, without disproportionately penalizing longer-running tasks. In addition, it reduces scheduling oscillation, prevents starvation among long-running tasks, and lowers queue time for urgent tasks. Similarly, computational cost remains controlled across different configurations, representing that the LITO algorithm effectively employs fog and edge resources to minimize overhead. This efficiency arises from the lemur-inspired huddling strategy, in which EDs implicitly collaborate by sharing overload via selective offloading instead of offloading indiscriminately. By distributing bursty workloads across multiple nodes, huddling stabilizes SLA performance during spikes, enhancing feasibility without overconsuming any single resource pool, which contrasts with traditional metaheuristic methods that may overload resources because of the lack of dynamic feasibility awareness.

Task completion time and task success ratios remain favorable when workload increases, while reflecting that tasks are completed successfully and within reasonable delay bounds. Lower task completion time is attained since LITO filters infeasible tasks at EDs and dispatches only schedulable workloads upward, eliminating redundant queuing and reprocessing at fog nodes. Task success ratios remain high under load by reflecting limited task abandonment and reliable execution. This integration indicates that tasks are both feasible and finishable instead of merely successful. This robustness leads to decentralized coordination among SNs, PNs, and EDs: SNs preserve neighborhood-level resource visibility, PNs perform global orchestration when necessary, and EDs decide feasibility locally. Such role separation suppresses oscillation between global and local decision scopes, minimizing both re-execution overhead and deadline violations.

The offloading ratio and SLA violation further emphasize the adaptability of the LITO framework. LITO minimizes offloading ratio under heavy workloads, eliminating deadline pressure and congestion at fog nodes. Configurations with very high workload sizes show modest SLA violation increases, which is expected in real deployments, yet overall rates remain low, demonstrating effective orchestration. Crucially, in stress cases where baseline methods would degrade severely due to unmanaged contention and queue saturation, LITO mitigates cascading failures through feasibility checks, role-based selection avoids churn between ED and fog layers, huddling spreads overload. This combination preserves SLA degradation by making degradation gradual rather than catastrophic, proving graceful failure behavior instead of abrupt collapse.

Task latency is measured under Lightweight Edge Load with a workload size of 100. Figure 14 illustrates that total task latency consistently decreases when data rate increases, indicating the reduced transmission time across the network. This superior performance arises from mechanism-level advantages of LITO. At low data rates, these mechanisms are vital since baselines confront sharp latency spikes because of uncoordinated task assignment and re-execution, whereas LITO preserves stable, low latency by adapting offloading decisions and task execution dynamically. Even as transmission delays reduce at higher data rates, LITO continues to surpass others, while ensuring steady end-to-end latency reduction under both stress and nominal conditions.

Attaining lower latency with LITO necessitates slightly higher energy consumption because of role-based coordination and dynamic task migrations. Likewise, improved SLA is associated with a lower offloading ratio, since feasibility checks avoid overloading edge nodes. In other words, moderate energy overhead and reduced offloading flexibility are sacrificed to gain more reliable SLA and stable latency, emphasizing LITO's balanced approach to performance optimization.

5.4. Ablation Study on Execution Scenarios

Figures 15 and 16 illustrate the effect of diverse execution strategies on makespan and throughput within a fog-computing environment. Executing tasks only at ED leads to the lowest makespan and highest throughput because local processing avoids offloading delays and offloading congestion. However, this approach risks resource saturation under high-load conditions, resulting in potential queuing and SLA violations. On the other hand, offloading all tasks from the ED results to fog nodes in a significantly higher makespan and a sharp reduction in throughput as a result of communication overhead, task oscillation across SNs and PNs, and congestion at fog nodes, demonstrating that purely centralized execution is not suitable for edge-heavy traffic. The hybrid-ED strategy, in which tasks are partially processed locally and partially offloaded, offers a balanced behavior. Makespan modestly increases when compared with pure local execution, but throughput stabilizes since tasks are dynamically distributed across nodes, mitigating overload conditions while effectively utilizing available fog resources. By utilizing role-based selection and huddling mechanism, the proposed hybrid SN + ED model further improves performance. Huddling spreads tasks amongst lightly loaded EDs and SNs, minimizing oscillation and mitigating sudden increases in queue lengths. The basking (energy-aware) rule guarantees that tasks are executed with priority in which energy consumption is minimal without affecting task latency.

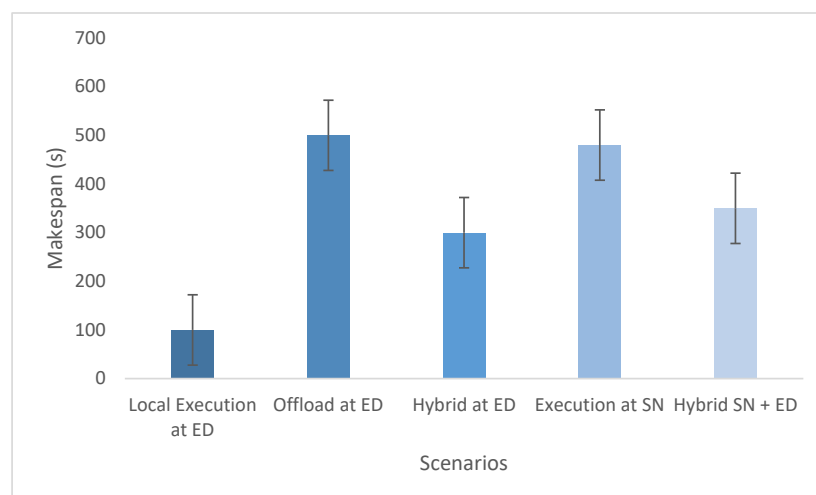


Figure 15. Comparison of makespan across different execution scenarios.

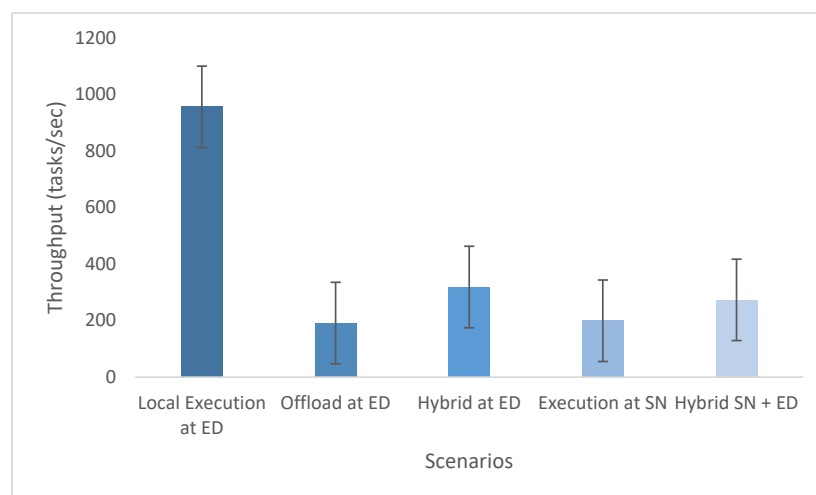


Figure 16. Comparison of throughput across different execution scenarios.

Figure 17 confirms that these mechanisms' contributions (the full LITO model) achieves the lowest latency. Removing huddling rises latency, since load is unevenly distributed, leading to queuing delays. Disabling basking further increases latency due to less energy-efficient execution decisions. In contrast, removing feasibility filtering yields the highest latency, since tasks are allocated to overloaded nodes, inducing re-execution and congestion. These results prove that the combination of lemur-inspired mechanisms allows LITO to preserve performance under high-load stress, in which baseline strategies collapse, while ensuring minimized queuing, balanced workload distribution, and stable SLA adherence.

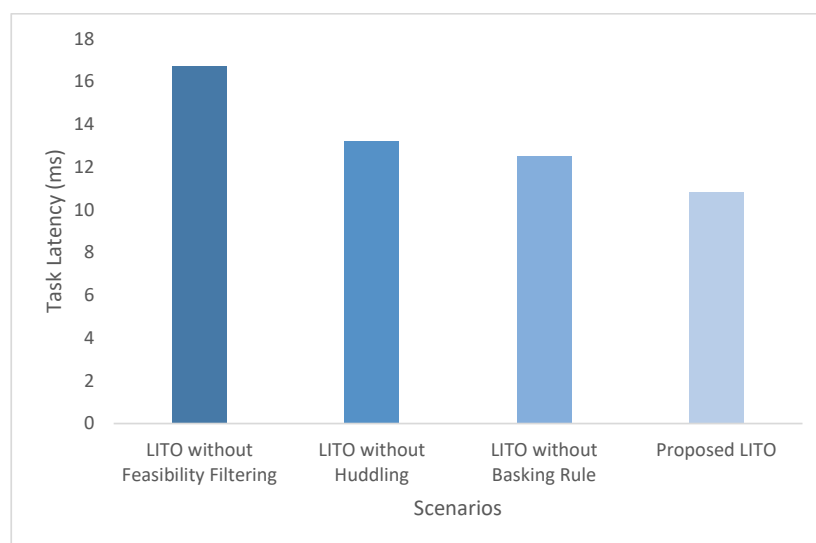


Figure 17. Comparison of task latency for the LITO ablation study.

5.5. Robustness Evaluation Under Dynamic Disturbances

To assess robustness under realistic fog disturbances, controlled disturbance events are inserted during simulation runtime. For the dynamic node failure scenario, one or two SNs are randomly deactivated at a uniformly sampled time between 30% and 60% of the total simulation period. During failure, the PN instantly eliminates the failed SN from the action set, and all newly arriving tasks are reallocated through the fallback routing mechanism and the feasibility filter. Tasks queued at the failed SN are redistributed to the following best feasible SN on the basis of CSPL-CB policy. For network perturbation experiments, link bandwidth between IoT–ED and PN–SN layers is changed dynamically within $\pm 30\%$ of nominal values through a stochastic perturbation model, and random latency

spikes of 1–5 ms are also inserted to emulate transient congestion. The PN continuously updated the EWMA-based link statistics in X_t , enabling adaptive routing under unstable network conditions.

The robustness analysis in Table 5 shows that LITO demonstrates controlled degradation during dynamic disturbance conditions. When a single SN fails, system throughput confronts only a slight decline, which is accompanied by a minor upsurge in SLA violation rate and a small rise in makespan. This limited effect shows the efficiency of feasibility filtering, fallback routing, and hierarchical coordination at the PN. Under dual SN failures, performance degradation is proportional rather than severe, with moderate drops in throughput and manageable upsurges in SLA violations and completion time, signifying its improved operational stability and reduced processing capacity. In bandwidth perturbation scenarios, reduced bandwidth results in a modest performance impact as a result of increased transmission delays, whereas bandwidth improvement lowers makespan and improves throughput. Hence, LITO preserves stable TSRs and bounded SLA violations across all disturbance settings, proving its resilience and adaptive decision-making capability under dynamic fog environments.

Table 5. Robustness evaluation of LITO under bandwidth perturbations and dynamic SN failures.

Scenarios	Throughput	SLA Violation Rate (%)	TSR	Makespan
Normal	270	0.09	0.91	20.732
1 SN Failure	263.5	0.101	0.899	21.210
2 SN Failure	257.8	0.112	0.888	21.964
Bandwidth −30%	259.2	0.105	0.895	21.650
Bandwidth +30%	278.4	0.082	0.918	20.118

6. Conclusions

This study addressed persistent challenges in fog computing, including energy inefficiency, latency, and uneven resource utilization. By modelling fog-layer coordination through cooperative lemur social behaviors, LITO adopts a decentralized architecture in which EDs, SNs, and PNs collaboratively manage workload distribution. The framework supports local decision-making via VFDT-based incremental learning to adapt to workload changes, while CSPL-CB policy refinement enables PNs to adjust task distribution across the fog hierarchy. At the SN layer, hybrid EDF-DRS scheduling prioritizes latency-sensitive tasks while maintaining efficient allocation for normal workloads.

Experimental results across multiple configurations, workload sizes, and task-complexity settings show that LITO reduces communication and computation energy consumption, improves throughput and resource utilization, and strengthens SLA adherence compared with the evaluated baselines. Under extreme workloads, performance becomes bounded by resource contention and queueing limits; however, LITO maintains stable behavior and balanced utilization relative to competing methods. Overall, LITO provides a practical biologically inspired approach to adaptive task offloading in fog environments.

Future work will extend LITO by integrating predictive workload models that forecast arrival intensity and task complexity to enable proactive feasibility filtering and earlier congestion avoidance. It will also incorporate fault-tolerant offloading under node and link failures through redundancy-aware placement, re-routing, and graceful degradation policies. Finally, we will quantify end-to-end overhead and scalability, including decision

latency and messaging cost, and validate the framework under more realistic dynamic conditions such as time-varying bandwidth and mobility.

Author Contributions: Conceptualization, A.A. and H.K.; methodology, A.A.; software, A.A.; validation, A.A. and H.K.; formal analysis, A.A.; investigation, H.K.; resources, A.A.; data curation, A.A.; writing, original draft preparation, A.A.; writing, review and editing, A.A. and H.K.; visualization, A.A.; supervision, H.K.; project administration, H.K. All authors have read and agreed to the published version of the manuscript.

Funding: This study was funded by the Ongoing Research Funding Program, King Saud University, Riyadh, Saudi Arabia (ORF-2026-204).

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

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

References

1. Laghari, A.A.; Jumani, A.K.; Laghari, R.A. Review and state of art of fog computing. *Arch. Comput. Methods Eng.* **2021**, *28*, 3631–3643. [CrossRef]
2. Saurabh, N.; Dhanaraj, R.K. A review paper on fog computing paradigm to solve problems and challenges during integration of cloud with IoT. *J. Phys. Conf. Ser.* **2021**, *2007*, 012017. [CrossRef]
3. Laroui, M.; Nour, B.; Mounghla, H.; Cherif, M.A.; Afifi, H.; Guizani, M. Edge and fog computing for IoT: A survey on current research activities and future directions. *Comput. Commun.* **2021**, *180*, 210–231. [CrossRef]
4. Kumari, N.; Yadav, A.; Jana, P.K. Task offloading in fog computing: A survey of algorithms and optimization techniques. *Comput. Netw.* **2022**, *214*, 109137. [CrossRef]
5. Chen, H.; Qin, W.; Wang, L. Task partitioning and offloading in IoT cloud-edge collaborative computing framework: A survey. *J. Cloud Comput.* **2022**, *11*, 86. [CrossRef]
6. Costa, B.; Bachiega, J.; De Carvalho, L.R.; Araujo, A.P.F. Orchestration in fog computing: A comprehensive survey. *ACM Comput. Surv.* **2022**, *55*, 1–34. [CrossRef]
7. Meticulous Research. Fog Computing Market Growth. 2025. Available online: <https://www.meticulousresearch.com/product/fog-computing-market-6224> (accessed on 31 October 2025).
8. Saeik, F.; Avgeris, M.; Spatharakis, D.; Santi, N.; Dechouniotis, D.; Violos, J.; Leivadeas, A.; Athanasopoulos, N.; Mitton, N.; Papavassiliou, S. Task offloading in edge and cloud computing: A survey on mathematical, artificial intelligence and control theory solutions. *Comput. Netw.* **2021**, *195*, 108177. [CrossRef]
9. Rezaee, M.R.; Hamid, N.A.W.A.; Hussin, M.; Zukarnain, Z.A. FOG offloading and task management in IoT-FoG-Cloud environment: Review of algorithms, networks, and SDN application. *IEEE Access* **2024**, *12*, 39058–39080. [CrossRef]
10. Sulimani, H.; Sulimani, R.; Ramezani, F.; Naderpour, M.; Huo, H.; Jan, T.; Prasad, M. HybOff: A hybrid offloading approach to improve load balancing in fog environments. *J. Cloud Comput. Adv. Syst. Appl.* **2024**, *13*, 113. [CrossRef]
11. Rahmani, A.M.; Haider, A.; Khoshvaght, P.; Gharehchopogh, F.S.; Moghaddasi, K.; Rajabi, S.; Hosseinzadeh, M. Optimizing task offloading with metaheuristic algorithms across cloud, fog, and edge computing networks: A comprehensive survey and state-of-the-art schemes. *Sustain. Comput. Inform. Syst.* **2025**, *45*, 101080. [CrossRef]
12. Agarwal, A.; Negi, P. Enhancing fog offloading and task management in IoT-fog-cloud environments: A review of cutting-edge algorithms, networks, and SDN applications. *Int. J. Commun. Netw. Inf. Secur.* **2025**, *17*, 1–48.
13. Ameena, B.; Ramasamy, L. Drawer cosine optimization enabled task offloading in fog computing. *Expert Syst. Appl.* **2024**, *259*, 125212. [CrossRef]
14. Dong, S.; Tang, J.; Abbas, K.; Hou, R.; Kamruzzaman, J.; Rutkowski, L.; Buyya, R. Task offloading strategies for mobile edge computing: A survey. *Comput. Netw.* **2024**, *254*, 110791. [CrossRef]
15. Khan, Z.A.; Aziz, I.A.; Osman, N.A.B. A review on task offloading using meta-heuristic algorithms on fog computing. In *Proceedings of the IEEE 21st Student Conference on Research and Development (SCoReD)*, Kuala Lumpur, Malaysia, 13–14 December 2023; pp. 469–475.
16. Matam, R.; Tripathy, S. Task offloading using fog analytics. In *IET eBooks*; IET: London, UK, 2024; pp. 1–16.
17. Latip, R.; Aminu, J.; Hanafi, Z.M.; Kamarudin, S.; Gabi, D. Metaheuristic task offloading approaches for minimization of energy consumption on edge computing: A systematic review. *Discov. Internet Things* **2024**, *4*, 35. [CrossRef]
18. Pilli, N.; Mohapatra, D.; Reddy, S.S. Review on meta-heuristic algorithm-based priority-aware computation offloading in edge computing system. *J. Inst. Eng. (India) Ser. B* **2025**, *106*, 1261–1286. [CrossRef]

19. Ullah, I.; Lim, H.-K.; Seok, Y.-J.; Han, Y.-H. Optimizing task offloading and resource allocation in edge-cloud networks: A DRL approach. *J. Cloud Comput.* **2023**, *12*, 112. [\[CrossRef\]](#)
20. Acheampong, A.; Zhang, Y.; Xu, X.; Kumah, D.A. A review of the current task offloading algorithms, strategies, and approaches in edge computing systems. *Comput. Model. Eng. Sci.* **2022**, *134*, 35–88.
21. Zheng, Y.; Li, A.; Wen, Y.; Wang, G. A UAV trajectory optimization and task offloading strategy based on hybrid metaheuristic algorithm in mobile edge computing. *Future Internet* **2025**, *17*, 300. [\[CrossRef\]](#)
22. Samarneh, A.A.; Alma'aitah, A.Y. Distributed task offloading in mobile edge computing using metaheuristics. In Proceedings of the 6th International Conference on Communications, Signal Processing, and their Applications (ICCSPA), Istanbul, Turkey, 8–11 July 2024; pp. 1–6.
23. Rasool, M.A.E.; Kumar, A.; Islam, A. Dynamic task offloading optimization in mobile edge computing systems with time-varying workloads using improved particle swarm optimization. *Int. J. Adv. Comput. Sci. Appl.* **2024**, *15*, 622–627. [\[CrossRef\]](#)
24. Naik, B.B.; Priyanka, B.; Ansari, S.A. Energy-efficient task offloading and resource allocation for edge computing: A quantum-inspired particle swarm optimization approach. *Clust. Comput.* **2025**, *28*, 155. [\[CrossRef\]](#)
25. Zaidi, S.; Attalah, M.A.; Khamer, L.; Calafate, C.T. Task offloading optimization using PSO in fog computing for the Internet of Drones. *Drones* **2024**, *9*, 23. [\[CrossRef\]](#)
26. Swaminathan, D.; Rajagopalan, A.; Nidumolu, V.; Alroobaea, R.; Kotb, H.; Aboras, K.M.; Elrashidi, A. OD-TRA-based task offload optimization for industrial Internet of Things: Improving efficiency and performance with digital twins and metaheuristic optimization. *IEEE Access* **2024**, *12*, 51796–51817. [\[CrossRef\]](#)
27. Thomas, P.; Jose, D.V. *Metaheuristics-Based Task Offloading Framework in Fog Computing for Latency-Sensitive Internet of Things Applications*; Lecture Notes in Networks and Systems; Springer Nature: Singapore, 2023; pp. 221–239.
28. Kumari, N.; Jana, P.K. A metaheuristic-based task offloading scheme with a trade-off between delay and resource utilization in IoT platform. *Clust. Comput.* **2023**, *27*, 4589–4603. [\[CrossRef\]](#)
29. AlShathri, S.I.; Chelloug, S.A.; Hassan, D.S.M. Parallel meta-heuristics for solving dynamic offloading in fog computing. *Mathematics* **2022**, *10*, 1258. [\[CrossRef\]](#)
30. Fard, A.F.; Ardakani, M.M.; Mirzaie, K. Multi-objective task offloading optimization in fog computing environment using INSCSA algorithm. *Clust. Comput.* **2024**, *27*, 7469–7491. [\[CrossRef\]](#)
31. Ali, A.; Azim, N.; Othman, M.T.B.; Rehman, A.U.; Alajmi, M.; Al-Adhaileh, M.H.; Khan, F.U.; Orken, M.; Hamam, H. Joint optimization of computation offloading and task scheduling using multi-objective arithmetic optimization algorithm in cloud–fog computing. *IEEE Access* **2024**, *12*, 184158–184178. [\[CrossRef\]](#)
32. Pillai, S.E.V.S.; Paramasivam, S.; Polimetla, K.; Dhanasekaran, S.; Agrawal, K.K.; Yadav, S.P.; Logeshwaran, J.; Gatto, G. Optimizing resource management using hybrid metaheuristic algorithm for fog layer design in edge computing. *Syst. Soft Comput.* **2025**, *7*, 200323. [\[CrossRef\]](#)
33. Attalah, M.A.; Zaidi, S.; Mellal, N.; Calafate, C.T. Task-offloading optimization using a genetic algorithm in hybrid fog computing for the Internet of Drones. *Sensors* **2025**, *25*, 1383. [\[CrossRef\]](#)
34. Kishor, A.; Chakarbarty, C. Task offloading in fog computing using smart ant colony optimization. *Wirel. Pers. Commun.* **2021**, *127*, 1683–1704. [\[CrossRef\]](#)
35. Shukla, P.; Pandey, S. MOTORS: Multi-objective task offloading and resource scheduling algorithm for heterogeneous fog–cloud computing scenario. *Res. Sq.* **2023**, *80*, 22315–22361. [\[CrossRef\]](#)
36. Saif, F.A.; Latip, R.; Hanapi, Z.M.; Kamarudin, S.; Kumar, A.V.S.; Bajaher, A.S. Multi-objectives firefly algorithm for task offloading in edge–fog–cloud computing. *IEEE Access* **2024**, *12*, 159561–159578. [\[CrossRef\]](#)
37. Sui, X.; Wang, J.; Zhang, S.; Zhang, S.; Zhang, Y. Multi-strategy fusion mayfly algorithm on task offloading and scheduling for IoT-based fog computing multi-tasks learning. *Artif. Intell. Rev.* **2025**, *58*, 144.
38. Zhang, S.; Wang, J.; Zhang, S.; Xing, Y.; Sui, X. Multi-strategy fusion snake optimizer on task offloading and scheduling for IoT-based fog computing multi-tasks learning. *Clust. Comput.* **2024**, *28*, 69. [\[CrossRef\]](#)
39. Keshavarznejad, M.; Rezvani, M.H.; Adabi, S. Delay-aware optimization of energy consumption for task offloading in fog environments using metaheuristic algorithms. *Clust. Comput.* **2021**, *24*, 1825–1853. [\[CrossRef\]](#)
40. PKappeler, M.; van Schaik, C.P. Evolution of primate social systems. *Int. J. Primatol.* **2002**, *23*, 707–740. [\[CrossRef\]](#)
41. Appel, M. Lemur (36453479071).jpg. [Photograph], Wikimedia Commons. 3 July 2017. Available online: [https://commons.wikimedia.org/wiki/File:Lemur_\(36453479071\).jpg](https://commons.wikimedia.org/wiki/File:Lemur_(36453479071).jpg) (accessed on 31 December 2025).
42. Baden, A.L. A description of nesting behaviors, including factors impacting nest site selection, in black-and-white ruffed lemurs (*Varecia variegata*). *Ecol. Evol.* **2019**, *9*, 1010–1028. [\[CrossRef\]](#)
43. García-Martín, E.; Lavesson, N.; Grahn, H.; Casalicchio, E.; Boeva, V. Energy-aware very fast decision tree. *Int. J. Data Sci. Anal.* **2021**, *11*, 105–126. [\[CrossRef\]](#)
44. Sahni, Y.; Cao, J.; Yang, L.; Wang, S. Distributed resource scheduling in edge computing: Problems, solutions, and opportunities. *Comput. Netw.* **2022**, *219*, 109430. [\[CrossRef\]](#)

45. Bernard, L.; Yassa, S.; Alouache, L.; Romain, O. Efficient Pareto-based approach for IoT task offloading on fog–cloud environments. *Internet Things* **2024**, *27*, 101311. [[CrossRef](#)]
46. Gupta, H.; Dastjerdi, A.V.; Ghosh, S.K.; Buyya, R. iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. *Softw. Pract. Exp.* **2017**, *47*, 1275–1296. [[CrossRef](#)]
47. Calheiros, R.N.; Ranjan, R.; Beloglazov, A.; De Rose, C.A.F.; Buyya, R. CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exp.* **2011**, *41*, 23–50. [[CrossRef](#)]
48. Google. ClusterData2019 (Trace Version 3): Borg Cluster Workload Traces (May 2019). Cluster-Data GitHub Repository, Documentation File ClusterData2019.md. Available online: <https://github.com/google/cluster-data/blob/master/ClusterData2019.md> (accessed on 29 January 2026).

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.