Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.076163

ARTICLE

A Hybrid Harmony Search–Nondominated Sorting Approach for Cost-Efficient and Deadline-Aware Fog-Enabled IoT Placement

Zahra Farhadpour[1,*], **Tan Fong Ang**[1,*] **and Chee Sun Liew**[2]

[1]Department of Computer System and Technology, Faculty of Computer Science and Information Technology, Universiti Malaya, Kuala Lumpur, Malaysia

[2]Department of Smart Computing and Cyber Resilience, School of Computing and Artificial Intelligence, Faculty of Engineering and Technology, Sunway University, Kuala Lumpur, Malaysia

*Corresponding Authors: Zahra Farhadpour. Email: s2194093@siswa.um.edu.my; Tan Fong Ang. Email: angtf@um.edu.my

Received: 15 November 2025; Accepted: 26 February 2026; Published: 09 April 2026

ABSTRACT: The heterogeneity and dynamic behavior of fog computing environments introduce major challenges to achieving optimal application placement. Limited fog resources and varying workloads often necessitate offloading applications beyond their local clusters, making it difficult to maintain the required level of service quality under varying conditions. In this context, placement methods must ensure a balanced trade-off between multiple objectives, such as time and cost, while maintaining reliable adherence to constraints like application deadlines and limited fog-node memory. Existing solutions, including heuristic, metaheuristic, learning-based, and hybrid optimization approaches, have been proposed to address these challenges. However, many of these methods rely on weighted objective formulations with limited visibility into trade-offs among conflicting objectives, emphasize local refinement over sustained exploration, or handle resource and deadline constraints implicitly, reducing their effectiveness in resource-constrained and dynamic fog environments. To overcome these limitations, this paper proposes a hybrid Harmony Search–Nondominated Sorting Genetic Algorithm (HS–NSGA), which integrates the improvisation operators of HS with the Pareto-based elitist selection of NSGA-II. The proposed algorithm advances the state of the art by enabling unbiased trade-off exploration and preserving high-quality solutions while explicitly enforcing memory constraints and deadline requirements. HS serves as the primary search mechanism, generating diverse candidate placements through memory consideration, randomization, and pitch adjustment, while nondominated sorting and crowding distance guide the selection of high-quality trade-off solutions across iterations. To further improve robustness against constraint violations in dynamic conditions, a fraction of the population is diversified using NSGA-II mutation and crossover operators. Experimental evaluations across heterogeneous fog–cloud scenarios demonstrate that HS–NSGA consistently achieves superior cost–time reduction compared with representative hybrid genetic and swarm-based methods while satisfying application deadline requirements and fog-node memory constraints. For makespan, HS–NSGA achieved reductions of 14.5% over LD-NPGA (Local Draft Niche-Pareto Genetic Algorithm), 16.8% over FSPGA (Fog Service Placement Genetic Algorithm), 19.5% over FSPPSO (Fog Service Placement Particle Swarm Optimization), and 16.1% over GA-FSA (Genetic Algorithm Flamingo Search Algorithm), with consistent improvements across varying task volumes. For cost, reductions reached 32.7%, 35.5%, 42.9%, and 30.7% over LD-NPGA, FSPGA, FSPPSO, and GA-FSA, respectively.

KEYWORDS: Fog computing; hybrid metaheuristics; harmony search; application placement; multi-objective scheduling

1 Introduction

The expansion of the Internet of Things (IoT) into critical sectors such as industrial automation, transportation, and healthcare has created substantial demands for time-sensitive data processing [1,2]. Scenarios such as remote health monitoring, autonomous driving, and predictive maintenance rely on the timely analysis of large volumes of data generated at the edge of the network. Any violation of response time or reliability constraints in these settings can directly compromise safety, efficiency, and service quality. Fog computing acts as a distributed enabler for IoT applications by deploying computational resources closer to IoT devices [3,4]. However, the heterogeneity of fog nodes, their resource constraints, and fluctuating workloads present significant challenges. In practice, applications may need to be redirected beyond their originating clusters when local fog resources are insufficient, which increases the probability of violating Quality of Service (QoS) guarantees such as deadlines and memory requirements. In this context, the application placement problem in fog computing can be formulated as a constrained multi-objective optimization problem. The goal is to determine an optimal assignment between applications and fog resources that not only optimizes multiple objectives, such as execution cost and makespan, but also ensures strict adherence to resource and QoS constraints under dynamic and uncertain conditions [5,6]. A variety of optimization methods have been explored to tackle this problem, including heuristics, deterministic algorithms machine learning techniques, and metaheuristic approaches. In this context, Integer Linear Programming (ILP) has attracted significant attention because of its effectiveness in identifying global optima [7–10]. However, ILP models suffer from poor scalability. As the number of applications and nodes increases, the solution space grows exponentially, making these models computationally impractical for large-scale and highly dynamic scenarios. To overcome this, researchers have increasingly turned to evolutionary algorithms to optimize a single objective, such as latency, bandwidth consumption, cost, or energy usage, or to approach the problem from a multi-objective optimization perspective. Despite their flexibility, many conventional evolutionary methods suffer from premature convergence, producing suboptimal solutions. More recently, deep reinforcement learning (DRL) has also been explored [11–14]. By modeling placement as a sequential decision-making problem, DRL agents learn policies that aim to minimize delay and cost through interaction with the environment. Although DRL is promising, such approaches often rely on extensive training data and long training times, and their performance may degrade when deployed in highly dynamic fog environments that differ from the training conditions. This limits their practical applicability to real-time placement decisions.

Taken together, current placement methods fall short in different ways. Classical ILP scales poorly beyond modest instances; DRL requires heavy training and degrades under non-stationary workloads; and stand-alone evolutionary schemes often suffer from unbalanced exploration vs. exploitation, premature convergence, and high repair costs when offspring violate constraints. Weighted-objective cost models further bias the search and obscure true Pareto trade-offs in this environment. These limitations motivate the use of hybrid approaches that combine the complementary capabilities of multiple algorithms to improve performance in multi-criteria scenarios.

Building upon the exploitation strength and feasibility-preserving refinements of Harmony Search, together with the diversity maintenance and trade-off awareness of the Pareto-optimal selection mechanism, this paper proposes a hybrid Harmony Search–Nondominated Sorting Genetic Algorithm (HS–NSGA). HS–NSGA combines feasibility-aware local refinement via HS with NSGA-II's elitist multi-objective selection, improving convergence speed, preserving diversity without weighting bias, and reducing infeasibility repair. In this hybrid, HS serves as the primary search engine, generating candidate placements via memory consideration, pitch adjustment, and randomization. It refines promising task–device mappings stored in the harmony memory by reusing high-quality assignments from previous generations. A feasibility-aware pitch

adjustment reduces contention, and a diversification stage applies genetic-based crossover and mutation to a subset of the population to enhance exploration of the solution space. By producing a smooth spread of trade-offs, this hybrid addresses common gaps in the placement literature, including unbalanced exploration vs. exploitation, premature convergence, costly repair of infeasible offspring, and bias introduced by weighted-objective models. This design enables HS–NSGA to achieve superior cost–time trade-offs while consistently satisfying deadline requirements and memory constraints.

The main contributions of this paper are summarized as follows:

- We introduce a novel HS–NSGA hybrid algorithm that integrates HS improvisation with Pareto-based ranking to solve the fog application placement problem.
- The proposed algorithm jointly minimizes execution cost, makespan, and deadline violation while maintaining reliable adherence to memory constraints in heterogeneous and dynamic fog environments.
- A diversification phase incorporating NSGA-II-style mutation and crossover is employed to enhance exploration and mitigate premature convergence.
- Experimental evaluation across heterogeneous fog–cloud scenarios demonstrates that HS–NSGA outperforms comparable hybrid genetic and swarm-based methods in both cost–time optimization and constraint satisfaction.

The remainder of this paper is organized as follows. Section 2 reviews related works on fog application placement and hybrid metaheuristics. Section 3 presents the system model and problem formulation, while Section 4 introduces the proposed HS–NSGA algorithm. Section 5 analyzes the time complexity of the algorithm, and Section 6 describes the experimental setup. The results of the experiments are provided and discussed in Sections 7 and 8, with the statistical test results reported in Section 9. Section 10 discusses the limitations of the study and outlines future research directions. Finally, Section 11 concludes the paper.

2 Related Work

Existing literature in application placement presents diverse solution techniques and optimization strategies, ranging from classical optimization methods to advanced machine learning and hybrid metaheuristics. These approaches can broadly be categorized into deterministic models, heuristic techniques, metaheuristics, learning-based solutions, and hybrid methods.

2.1 Classical and Deterministic Approaches

Classical application placement approaches typically formulate the problem using exact optimization techniques such as Integer Linear Programming (ILP), Mixed-Integer Linear Programming (MILP), or related mathematical programming models, aiming to derive optimal placement decisions under explicit system and quality of service constraints [7–10]. Several studies extended basic ILP and MILP formulations to improve realism or scalability. Fuzzy-enhanced MILP models have been proposed to prioritize applications based on deadline sensitivity and workload characteristics [9], while availability-aware and graph-based formulations exploit network structure to improve deadline satisfaction and service resilience. Other works extended deterministic models to capture additional system aspects such as server disaggregation, energy efficiency, or microservice replication across multi-core architectures [7,10]. More complex formulations, including mixed-integer nonlinear programming, have also been explored to jointly optimize energy consumption and latency under strict constraints [8].

Despite their mathematical rigor, deterministic approaches suffer from fundamental limitations. Their computational complexity grows rapidly with the number of tasks and nodes, making them unsuitable

for large-scale or dynamic fog environments. Even when combined with auxiliary mechanisms such as fuzzy logic, graph partitioning, or decomposition strategies, these models often require heuristic simplifications to remain tractable [7–10]. As a result, deterministic placement methods are primarily used as design-time optimizers or benchmark references, motivating the shift toward heuristic, metaheuristic, and hybrid approaches.

2.2 Heuristics

To overcome the scalability limitations of exact ILP and MILP formulations, many studies have proposed heuristic placement strategies that provide fast and computationally lightweight solutions suitable for large-scale and real-time fog environments [15–18]. These approaches typically rely on rule-based or greedy decision mechanisms that prioritize workloads based on predefined criteria like workload characteristics or resource availability. Several heuristic methods focus on reducing latency and improving quality of service by exploiting hierarchical fog-cloud architectures. For example, requirement-aware and QoS-driven heuristics prioritize delay-sensitive tasks and attempt to place them closer to end users through clustering or context-aware decision rules [15,16]. Other approaches aim to improve deadline satisfaction by exploring multiple candidate clusters or duplicating placement requests, thereby increasing the likelihood of feasible placement under resource constraints [17]. Energy-aware heuristics have also been proposed, combining module prioritization and consolidation strategies to reduce makespan and energy consumption [18].

Despite their efficiency and scalability, heuristic approaches suffer from inherent limitations. Their reliance on predefined rules and greedy decisions often leads to suboptimal trade-offs when multiple conflicting objectives must be optimized simultaneously. Moreover, these methods are typically tailored to specific system assumptions and quality of service targets, which makes them sensitive to workload fluctuations and limits their robustness in dynamic fog-cloud environments. These shortcomings have motivated the adoption of metaheuristic and hybrid approaches that offer stronger exploration capabilities and more effective trade-off management.

2.3 Metaheuristic Approaches

Metaheuristic approaches have been widely adopted for placement problem in fog environments due to their scalability and flexibility in optimizing conflicting objectives such as latency, cost, and energy consumption [19–22]. Unlike deterministic methods, which suffer from scalability issues, and heuristic approaches, which rely on greedy rules, metaheuristics employ stochastic search operators to explore solution spaces more effectively. In this category, several approaches enhance search efficiency by incorporating knowledge reuse mechanisms or diversity-preserving initialization strategies to avoid premature convergence and maintain population diversity [19,20]. Cost- and latency-aware metaheuristic designs have also been explored, demonstrating improved convergence behavior and solution quality compared to traditional heuristic baselines [21,22].

Despite their effectiveness in managing multi-objective trade-offs, metaheuristic approaches introduce higher computational overhead than heuristics and exhibit stochastic variability across runs. These limitations have motivated the development of hybrid methods that combine metaheuristics with complementary strategies to improve adaptability, convergence efficiency, and solution stability.

2.4 Learning-Based Approaches

Learning-based methods, particularly deep reinforcement learning (DRL), have been increasingly applied to fog application placement due to their ability to adapt decision-making in dynamic and uncertain environments [11–14,20,23]. By learning placement policies through continuous interaction with

the environment, these approaches are well suited to heterogeneous, mobile, and large-scale fog systems where static optimization models may fail to capture evolving system states. Several studies have explored DRL frameworks to improve placement efficiency and adaptability. Distributed and asynchronous DRL architectures have been proposed to enhance service placement success rates across heterogeneous fog colonies, often supported by pre-training or prediction mechanisms to accelerate convergence [20]. Graph-based reinforcement learning models have also been introduced to capture structural dependencies among tasks and resources, enabling efficient decision-making and near real-time inference once training is completed [14]. More recently, federated and hierarchical reinforcement learning approaches have been applied to vehicular and UAV-assisted fog environments, allowing decentralized policy learning while preserving data privacy and exploiting contextual information such as traffic density and network topology [23].

Overall, DRL-based frameworks demonstrate strong adaptability to heterogeneity and are capable of capturing complex structural dependencies. Nevertheless, they face persistent challenges, including high training overhead, reliance on representative training environments, and reduced solution diversity compared to evolutionary algorithms. These limitations highlight the need for hybrid approaches that integrate the complementary strengths of different methods, thereby improving robustness, adaptability, and decision support in fog application placement.

2.5 Hybrid Approaches

Hybrid optimization approaches have been widely explored to combine complementary algorithmic strengths and improve solution quality in complex optimization problems. In such problems, metaheuristic search is often integrated with constructive or repair-based mechanisms to ensure solution feasibility while maintaining effective exploration of large combinatorial spaces [24–27]. Building on these general optimization principles, hybrid approaches have attracted growing interest in fog application placement, and several hybrid designs have been proposed to enhance placement performance from different perspectives. Metaheuristic combinations, such as genetic algorithm (GA) hybridized with simulated annealing (SA) or local search strategies, have been explored to improve optimization quality across objectives including cost, makespan, and energy consumption, achieving better balance than standalone algorithms [28,29]. Cost-driven optimization has also been investigated through hybridizing GA with the Flamingo Search algorithm to reduce SLA violation costs compared to traditional genetic-based methods [30]. In addition, recommendation-based hybrid frameworks have been introduced to balance user-centric QoS requirements with resource availability, demonstrating improvements in waiting time and overall resource utilization relative to heuristic and metaheuristic baselines [31]. Mobility-aware hybrid approaches further integrate prediction and context-awareness with heuristic scheduling to optimize fog service allocation and reduce migration costs and latency disruptions [32].

Despite these advances, important limitations remain in existing hybrid approaches. Many studies rely on weighted objective formulations [28,30–32] that are sensitive to weight selection and provide limited visibility into true trade-offs among conflicting objectives [33]. Moreover, most existing hybrids combine evolutionary algorithms with local search mechanisms primarily to refine local solutions, placing limited emphasis on sustained exploration of the solution space [29]. In addition, explicit support for memory constraints and deadline requirements is often absent, with feasibility and quality of service considerations typically handled implicitly rather than formulated within a constrained optimization framework, which is critical in resource-constrained fog infrastructures. In the following, Table 1 presents a comparative summary of representative application placement approaches discussed in each category, underlying the key characteristics and limitations of each work.

Table 1: Comparative summary of representative application placement approaches in fog computing.

Category	Work	Highlights	Limitations
Deterministic	[7]	Explicit workflow and multi-core modeling; realistic testbed validation.	Poor scalability and limited adaptability to dynamic conditions.
	[8]	Rigorous mathematical formulation with joint task allocation and CPU usage optimization.	High computational complexity with centralized control and limited scalability.
	[9]	Fuzzy QoS-aware prioritization; hierarchical fog–cloud placement	Static fuzzy rules with limited exploration capability.
	[10]	Server disaggregation; joint energy-delay-aware modeling.	Centralized control and weighted objective modeling with limited scalability.
Heuristic	[15]	Fog-node clustering with horizontal cooperation and deadline-aware placement.	Rule-based decisions with no explicit trade-off analysis and limited adaptability.
	[16]	Service placement with improved acceptance ratio and responsiveness.	Weighted rule-based decisions with limited exploration and solution diversity.
	[17]	Decentralized cluster-based architecture with explicit focus on delay minimization.	Single-objective optimization with no consideration to constraints.
	[18]	Modular placement with energy reduction via node consolidation.	Greedy heuristic decisions based on fixed priority rules with no global search.
Metaheuristic	[19]	Improved delay minimization and load balancing; scalable to large task sets.	Centralized process with aggregated fitness and no explicit memory constraints modeling.
	[21]	Improved convergence and solution quality over genetic and swarm-based methods.	Weighted-sum objective formulation with no Pareto-based trade-off analysis.
	[22]	Joint consideration of energy, bandwidth, and runtime with improved efficiency over genetic and random search methods.	Aggregated cost function with implicit constraint handling and no trade-off analysis.
Learning-based	[14]	Captures service dependencies and task priorities through graph-based learning.	High training overhead; requires multiple trained models for different weight settings.
	[20]	Decentralized decision-making with support for dynamic service arrivals.	Weighted reward formulation requiring careful tuning; high training overhead.

(Continued)

Table 1 (continued)

Category	Work	Highlights	Limitations
	[23]	Hierarchical privacy-preserving framework using contextual clustering and personalized federated learning.	High training and communication overhead; sensitivity to hyperparameters and complex system design.
Hybrid	[28]	Population-based search with improved solution quality over greedy baselines under increasing workloads.	Limited visibility into true trade-offs due to weighted-sum formulation with no explicit Pareto-front construction.
	[29]	Explicit Pareto front generation for cost and makespan with enhanced decision flexibility.	Computational overhead due to niche comparison and drafting; no explicit modeling of memory and deadline constraints.
	[30]	Multi-cost model with improved task guarantee ratio and total cost.	Weighted cost formulation with no explicit trade-off analysis and constraint handling.
	[31]	Context-aware recommendation-based placement considering user preferences, service history, and mobility.	Similarity- and rule-based decisions rather than systematic search; solution quality depends on historical data and similarity metrics.
	[32]	Proactive mobility prediction mechanism with service dependency consideration.	Greedy placement decisions with solution quality dependent on prediction accuracy.

While the individual categories discussed above address application placement from different perspectives, the comparative analysis in Table 1 reveals several cross-cutting limitations that persist across existing approaches. Deterministic formulations lack scalability and adaptability in dynamic fog environments, whereas heuristic and learning-assisted methods prioritize fast decision making at the expense of explicit trade-off visibility and systematic exploration. Metaheuristic and hybrid solutions improve optimization capability, yet many rely on weighted objective formulations or localized refinement strategies that limit insight into Pareto-optimal trade-offs and reduce solution diversity under heterogeneous workloads. Furthermore, explicit enforcement of practical system constraints, particularly fog-node memory capacity and application deadline requirements, is frequently handled implicitly or embedded within cost or reward functions, which reduces robustness in resource-constrained settings. These observations highlight the need for a unified placement approach that supports sustained exploration, explicit Pareto-based trade-off analysis, and constraint-aware optimization.

Motivated by these gaps, this paper proposes the HS–NSGA algorithm. By formulating the placement problem as a Pareto-based constrained optimization problem, explicitly enforcing fog-node memory constraints, and minimizing deadline violations alongside cost and makespan objectives, HS–NSGA enables unbiased trade-off analysis while maintaining feasibility and solution diversity. Unlike existing evolutionary and local-search-based hybrids, HS–NSGA adopts a collaborative hybridization strategy in which multiple complementary search mechanisms operate jointly within each iteration. Harmony search [34] is integrated

as a population-level exploration mechanism to introduce diverse candidate solutions and mitigate premature convergence. In parallel, genetic algorithm crossover and mutation operators are applied to further enhance diversification and explore promising regions of the search space. The nondominated sorting and elitist selection mechanisms of NSGA-II [35] ensure that high-quality nondominated solutions discovered at each iteration are preserved and propagated across generations. This collaborative design distinguishes HS–NSGA from existing hybrid approaches that primarily rely on local refinement or scalarized objectives and allows it to balance sustained exploration with the preservation of high-quality Pareto-optimal solutions in fog–cloud environments.

3 System Architecture and Objective Functions

The system architecture, task model, and task allocation formulation build upon the framework introduced in our previous study [36], with structural modifications incorporated to support the distinct hybrid search mechanism of HS-NSGA. A three-tier IoT–Fog–Cloud architecture is therefore adopted, as depicted in Fig. 1. The subsequent sections outline the task assignment model, objective functions, and problem formulation, with Table 2 summarizing the corresponding notations.

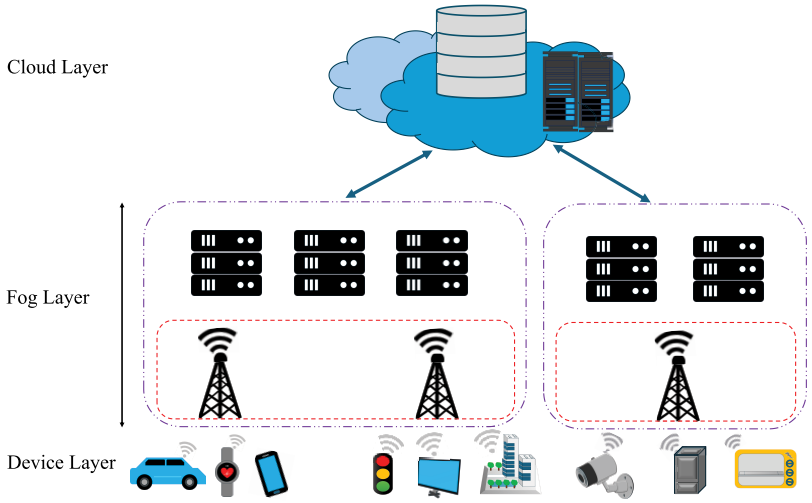

Figure 1: System architecture.

Table 2: Mathematical symbols and definitions.

Symbol	Definition
N	Binary offloading variable indicating the type of node selected for task execution
T	A group of tasks to be scheduled
τ_i	A tuple defining task i
c_i	Computational workload of task i
d_i	Completion deadline of task i
μ_i	Memory requirement of task i
s_i	Input data size for task i
s_i'	output data size for task i
n	Number of tasks concurrently executing in each time slot

(Continued)

Table 2 (continued)

Symbol	Definition
m	Number of nodes concurrently executing a set of tasks
a_{ij}	Binary mapping variable representing assignment of task i to node j
P_j^N	CPU computation rate of node j
CP_j^N	Cost of CPU usage on node j
NM_j^s	Memory capacity of node j
CM_j^N	Unit memory cost of node j
CBW_j^N	Unit bandwidth cost at node j
CT_{ij}^N	Completion time of task i on node j
Tr_{ij}^N	Input data transfer time for task i
$Tr_{ij}^{'N}$	Output data transfer time for task i
ET_{ij}^N	Execution time of task i on node j
de_j^N	Node j delay on LAN/WAN
BW_j^N	Network bandwidth of node j
dv_{ij}^N	Deadline violation for task i on node j

3.1 Task Model

Applications are represented as a set of tasks. Each task τ_i is characterized by the tuple $\tau_i = < c_i, \mu_i, d_i, s_i, s_i^{'} >$ where c_i denotes the task's computational workload, μ_i represents the memory requirement, d_i specifies the task deadline and $s_i, s_i^{'}$ denote the input and output data sizes for each transmission respectively.

3.2 Objective Functions

3.2.1 Makespan

The first objective in our model is makespan, defined as the maximum completion time across all tasks in the system. It indicates the time when the final task in the task set (T) completes execution, thereby determining the overall application completion time. Minimizing makespan is particularly critical in fog–cloud environments, where delay-sensitive IoT applications such as real-time healthcare monitoring or autonomous driving demand strict compliance with latency requirements. A shorter makespan ensures faster system response and enhanced QoS. Formally, makespan is expressed in Eq. (1) as:

$$MS\ (T) = Max\ (Et_j^N)\ \forall j\ \in\ \{1, 2, \ldots, m\}, N\ \in\ \{0, 1\}, \tag{1}$$

where, Et_j^N is computed by

$$Et_j^N = \sum_{i=1}^{n} a_{ij} \times \frac{c_i}{P_j^N} \forall j\ \in\ \{1, 2, \ldots, m\}, N\ \in\ \{0, 1\}, \tag{2}$$

where, $\frac{c_i}{P_j^N}$ represents the execution time of task i on node j of type N, computed based on the task's computational workload and the processing capacity of the selected node.

3.2.2 Total Cost Minimization

The second objective of our model is the minimization of total cost, which accounts for the monetary and resource expenses incurred when deploying IoT applications across fog and cloud resources, comprising three components:

- CPU usage cost, denoted by cpu_{cost}, reflects the processing cost incurred by executing the task on the selected node and is proportional to the task execution time and the node's processing cost rate.
- Memory usage cost, denoted by mem_{cost}, represents the cost associated with allocating memory resources required by the task on the target node.
- Bandwidth cost, denoted by BW_{cost}, accounts for the communication cost of transferring task input and output data between IoT devices, fog nodes, and cloud servers.

These three components together define $Ecost_{ij}$, the execution cost of assigning task i to node j. The overall cost objective, $COST(T)$, aggregates these costs across all tasks and computing nodes as calculated by Eqs. (3) and (4).

$$COST(T) = \sum_{i=1}^{n} \sum_{j=1}^{m} a_{ij} \times Ecost_{ij}, \tag{3}$$

$$Ecost_{ij} = cpu_{cost} + mem_{cost} + BW_{cost}. \tag{4}$$

Each cost component is computed based on the resource consumption of the task and the corresponding unit cost of the selected node, as defined in Eqs. (5)–(7).

$$cpu_{cost} = CP_j^N \times \frac{C_i}{P_j^N}, \tag{5}$$

$$mem_{cost} = CM_j^N \times \mu_i, \tag{6}$$

$$BW_{cost} = CBW_j^N \times (s_i + s_i^{'}). \tag{7}$$

3.2.3 Deadline Violation Model

In our model, deadline violation is quantified using the normalized lateness of tasks, which captures the extent to which task completion times exceed their specified deadlines. The overall deadline violation across the task set is defined as

$$DV(T) = \sum_{j=1}^{m} \sum_{i=1}^{n} a_{ij} \times dv_{ij}^N, \tag{8}$$

where dv_{ij}^N represents the normalized deadline violation of task i when executed on node j of type N, and is computed as

$$dv_{ij}^N = \max\left(\frac{CT_{ij}^N}{d_i} - 1, 0\right). \tag{9}$$

Deadline violations are treated as a soft optimization objective in this work, and as defined in Eq. (9), tasks exceeding their deadlines are explicitly penalized through the normalized lateness metric.

Here, d_i denotes the deadline of task i, and CT_{ij}^N is the end-to-end completion time of task i executed on node j of type N, calculated by

$$CT_{ij}^N = Tr_{ij}^N + ET_{ij}^N + Tr_{ij}^{'N} + de_j^N, \tag{10}$$

$$Tr_{ij}^{N} = \frac{s_i}{BW_j^N}, \tag{11}$$

$$ET_{ij}^{N} = \frac{c_i}{P_j^N}, \tag{12}$$

$$Tr_{ij}{}'N = \frac{s_i{}'}{BW_j^N}, \tag{13}$$

where Tr_{ij}^{N} and $Tr_{ij}{}'N$ denote the input and output data transmission times respectively, ET_{ij}^{N} represents the execution time of task i on server j of type N, and de_j^N accounts for the network delay associated with the selected offloading node.

3.2.4 Problem Formulation

The application placement problem is modelled as a multi-objective constrained optimization problem that simultaneously minimizes makespan, total cost, and deadline violations, subject to system resource and feasibility constraints.

$$Min\ F(T) = [MS(T), COST(T), DV(T)] \tag{14}$$

Subject to

$$CT1: \sum_{i=1}^{n} a_{ij}\ \mu_i\ \leq\ M_j^N,\ \forall j\ \in\ \{1, \ldots, m\} \tag{15}$$

$$CT2: \sum_{j=1}^{m} a_{ij} = 1, \forall i \in \{1, \ldots, n\} \tag{16}$$

In this formulation, memory capacity is imposed as an explicit constraint, as it directly determines the feasibility of task placement on computing nodes. Deadline requirements, on the other hand, are modeled as a quality-of-service objective and are minimized through a normalized lateness metric, as calculated in Eq. (9). CPU processing capacity and network bandwidth are implicitly captured through the execution time, transmission time, and cost models defined in Eqs. (2)–(13). This modeling choice reflects common practice in the related literature on constrained multi-objective system design [37,38], where feasibility-defining conditions are enforced explicitly, while other resource effects and service-level requirements are incorporated through performance and cost formulations.

Since the objectives are often conflicting, meaning that reducing one objective may result in increasing the other one, the Pareto-based optimization is employed. *A* solution $A^{\star}$ is considered Pareto-optimal if no other feasible allocation A exists such that:

$$F(A) \preceq F(A^{*}) \text{ with at least one strict inequality.} \tag{17}$$

The set of all such non-dominated solutions constitutes the Pareto front, from which a final placement configuration can be selected based on system requirements and trade-off preferences.

4 Proposed Method: Proposed Hybrid Harmony Search Nondominated Sorting Genetic Algorithm (HS-NSGA)

To address the placement challenges outlined earlier, this section presents the proposed hybrid HS–NSGA algorithm. Given the need for both comprehensive exploration and targeted local refinement in fog computing scenarios, the algorithm is designed to discover promising trade-offs while ensuring that placements satisfy deadline requirements and memory constraints. To achieve this, we integrated the

memory-based search dynamics of HS with the evolutionary structure of NSGA-II, enabling HS-NSGA to enhance global exploration and apply pitch adjustment for localized refinement. In this framework, each chromosome encodes a feasible mapping of tasks to fog or cloud nodes, along with auxiliary variables required to satisfy system constraints. Feasibility is preserved by embedding memory constraints within the solution encoding and by applying repair procedures to maintain valid decision-space bounds, ensuring that the search process focuses on realistic placement configurations, while deadline violations are addressed within the objective formulation. An overview of the HS–NSGA workflow is illustrated in Fig. 2, followed by the detailed pseudocode presented in Algorithm 1. The HS-NSGA algorithm operates through three main phases: initialization, hybrid offspring generation, and final Pareto solution generation. The algorithm begins by initializing a diverse population, then enters an iterative process in which new candidate solutions are produced through a combination of HS-based harmonies, GA crossover, and mutation. These solutions are merged and evaluated collectively, after which nondominated sorting and crowding distance mechanisms determine the subset that progresses to the next iteration. This cycle repeats until the maximum iteration count is reached, after which the first nondominated set (*F1*) is returned as the Pareto front, illustrating the trade-offs among makespan, cost, and constraint satisfaction. The following sections describe each phase in more detail.

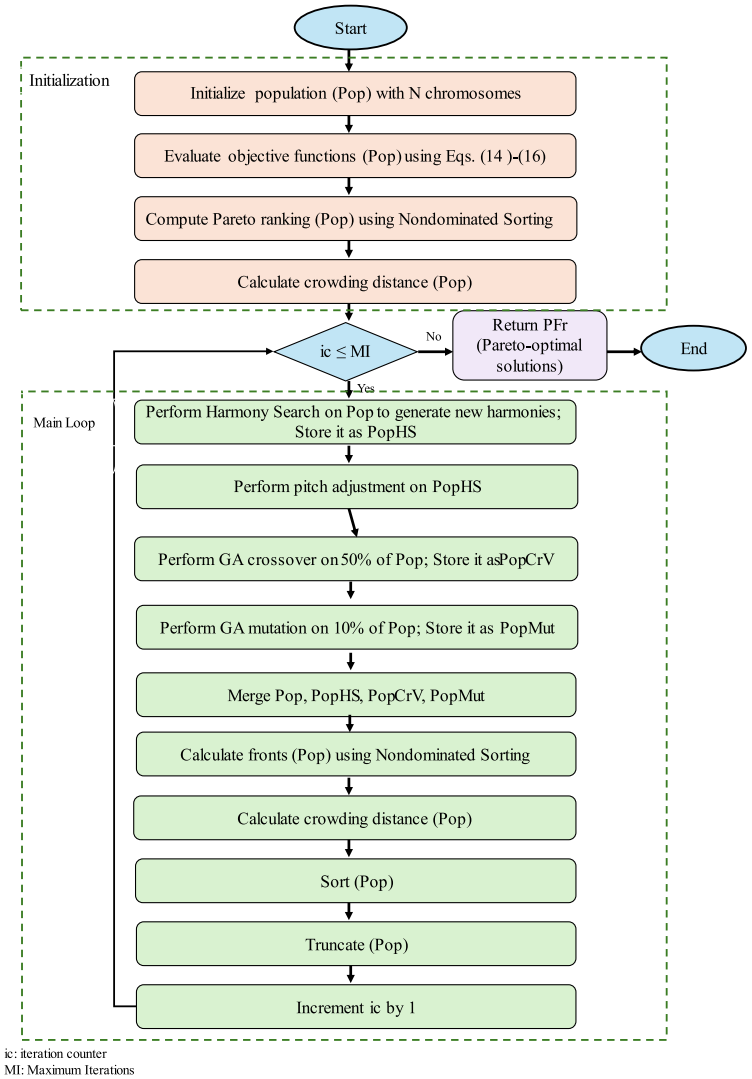

Figure 2: HS-NSGA flowchart.

Algorithm 1: HS-NSGA

Input: Task allocation requests and required computational resources.
Output: PFr (Final Pareto front containing optimal task allocation configurations).
/* Pop: Population of candidate solutions, N: Population size, F: Set of non-dominated fronts, MI: Upper bound on iterations, ic: Current iteration index, PFr: Final Pareto front set, PopHS: Buffer for HS-derived solutions, PopCrV: Buffer for crossover offspring, PopMut: Buffer for mutation offspring*/
//Population initialization
1: Pop ← Construct an initial set of N chromosomes through random generation.
2: Evaluate the objective function values of the population using Eqs. (14)–(16).
3: Compute Pareto ranking (Pop) using Nondominated Sorting.
4: Compute Crowding Distance (Pop).
//Hybrid offspring generation
5: **While** ic <= MI **do**
6: Generate PopHS & Evaluate the objective function values using Algorithm 2.
7: PopCrV ← Apply GA-based crossover to 50% of Pop and Compute the objective function values using Eqs. (14)–(16).
8: PopMut ← Apply GA-based mutation to 10% of the Pop & Compute the objective function values using Eqs. (14)–(16).
9: Pop ←Merge (Pop, PopHS, PopCrV, PopMut).
10: F← Compute Pareto ranking (Pop) using Nondominated Sorting.
11: Compute Crowding Distance (Pop).
12: Sort (Pop) by dominance hierarchy and crowding distance measures.
13: Pop ← Select the best-ranked solutions up to the population limit.
14: F← Compute Pareto ranking (Pop) using Nondominated Sorting.
15: Compute Crowding distance (Pop).
16: Sort (Pop) by dominance hierarchy and crowding distance measures.
17: Increment the iteration counter by one.
18: **end while**
//Final solution generation
19: PFr ← F 1.
20: **return** PFr.

4.1 HS-NSGA—Initialization Phase

The algorithm is initialized with a population *Pop* of size *N*, where each chromosome encodes a mapping of tasks to devices under the placement constraints. Chromosomes are generated randomly to maximize initial diversity, and each candidate solution is evaluated using Eqs. (14)–(16). Nondominated sorting and crowding distance are computed using the standard NSGA-II mechanisms [35]. Through this process, dominance tiers are identified, and a spread-preserving density measure is assigned. At the end of this phase, a diverse population, its Pareto ranking *F*, and the associated diversity scores are established to guide selection in the main loop.

4.2 Hybrid Loop for Offspring Generation and Selection

The second phase iterates until the maximum number of iterations *MI* is reached and performs complementary variation routes and elitist selection, as described in the following subsections.

4.2.1 Harmony-Search Intensification

At the beginning of each iteration, a set of candidate chromosomes *PopHS* is produced by the HS algorithm (Algorithm 2), which performs the HS improvisation procedure used to generate new harmonies.

Algorithm 2: HS

Input: Pop (Population of N Chromosomes)
Output: PopHS (Population Produced by Harmony Search)
/* MS: Memory Size, NHS: New harmony (chromosome) generation size, NH: New harmony population, HMCR: Harmony memory consideration rate, PAR: Pitch adjustment rate, FW: Fret bandwidth, D: Chromosome size */

```
1: for i ← 1 to NHS do
2:         NH(i) ← Randomly Generate New Harmony (chromosome)    //provides random selection case
3:         for j ← 1 to D do
4:                 r_j ← Random float between 0 and 1
5:                 if (rj ≤ HMCR) then
6:                 j_0 ← Random integer between 1 and MS
7:                     NH (i, j) ← Pop (j_0, j)                    //copy j-th gene from memory
8:                 end if
9:                 r_0 ← Random float between 0 and 1
10:                r_1 ← Random float between –1 and 1
11:                if (r_0 ≤ PAR) then
12:                    NH (i, j) ← NH(i, j) + FW * r_1          //pitch adjustment
13:            end if                                          //else: keep the random gene from line 2
14:        end for
15: end for
16: PopHS ← Evaluate Fitness (NH) using Eqs. (14)–(16).
17: return PopHS
```

The algorithm constructs each new harmony by selecting and adjusting gene values through three main mechanisms: memory consideration, pitch adjustment, and random selection. For each new harmony of size *D*, every gene is either drawn from the harmony memory, which consists of high-quality chromosomes in the current population, with probability *HMCR*, or sampled randomly to maintain exploration. Pitch adjustment is applied with probability *PAR*, where a small perturbation is added based on the fret bandwidth *FW* and a random factor in the range [–1, 1]. This process allows localized refinement around promising solutions while preserving feasibility. Pitch adjustment may perturb chromosome elements outside the valid decision space. To ensure feasibility, each modified chromosome is repaired prior to fitness evaluation. Specifically, after the pitch adjustment operation, each chromosome element, denoted by *NH(j)*, is bounded to the valid range of the search space as

$$NH(j) = min\left(max\left(NH\left(j\right), LB\right), UB\right) \tag{18}$$

where *LB* and *UB* specify the minimum and maximum boundaries of the search space. This repair step ensures that all candidate solutions passed to the evaluation stage remain within the admissible decision space. After all *NHS* new harmonies are improvised, they are evaluated using the objective functions in Eqs. (14)–(16), producing the set *PopHS* for use in the hybrid evolutionary loop.

4.2.2 Genetic Exploration via Crossover and Mutation

At this stage, genetic operators are applied to the current population to promote broader exploration. Crossover is applied to 20% of the population to produce *PopCrV*, promoting exploration in the search space by exchanging compatible sub-assignments between parents. Mutation is applied to 10% of the population to produce *PopMut*, injecting randomized but valid edits that help escape local optima and increase diversity. All offspring are evaluated using the objective functions given by Eqs. (14)–(16).

4.2.3 Elitist Selection and Diversity Preservation

In this stage, all candidate solutions are merged into a single population *Pop*, and nondominated fronts *F* are recomputed using nondominated sorting. Crowding distance is then computed with boundary points assigned infinite distance and intermediate points assigned the sum of normalized objective differences to promote a well-spread approximation of the Pareto set. Solutions are sorted by crowding distance and dominance rank, after which the population is truncated back to size *N*.

4.3 Termination and Pareto Solution Generation

After the main hybrid loop terminates at iteration *MI*, the current population has been ranked into nondominated fronts F. The final Pareto set is obtained by selecting the first nondominated front, *F*1, which contains all solutions for which no other solution in the population is strictly better in every objective. This front forms the Pareto-optimal set *PFr* of task-to-device mappings at termination. In practice, *PF* provides decision makers with a portfolio of trade-offs across the objectives defined in Eqs. (14)–(16), without bias toward any single objective. Finally, *PFr* is returned as the algorithm's output.

5 Complexity Analysis

The computational complexity of HS-NSGA can be evaluated with respect to the population size *N*, the number of iterations *MI*, and the problem dimension *D*, which is defined as the chromosome length corresponding to the number of tasks. During the initialization phase, population generation and fitness evaluation require $O(N.D)$, while non-dominated sorting incurs $O(N^2)$, which dominates this phase. In each iteration of the main loop, three offspring subpopulations are generated: HS-based harmonies, genetic crossover, and mutation, each contributing $O(N.\ D)$ including fitness evaluation. These offspring are merged and re-evaluated through non-dominated sorting of approximately $2N$ solutions, giving $O(N^2)$, while crowding distance calculation and sorting add $O(N \log N)$. The HS subroutine itself operates in $O(N.D)$, since each new harmony requires $O(D)$ operations across all decision variables. Therefore, the overall complexity per iteration is $O(N^2+N.D)$, and across all iterations the complexity of HS-NSGA is $O(MI.\ (N^2+N.D))$. In typical placement scenarios where $N \gg D$, the complexity is effectively dominated by $O(MI.\ N^2)$, consistent with classical multi-objective genetic-based methods.

6 Experimental Setup

To validate the effectiveness of the proposed HS-NSGA algorithm, a series of experiments were conducted under diverse fog-cloud configurations. These experiments aimed to examine the algorithm's ability to balance exploration and refinement while satisfying memory constraints and deadline requirements and to benchmark its performance against comparable hybrid methods. For all experiments, the system configuration was created by synthesizing communication and resource characteristics from recent research on fog-cloud environments [36,39–41]. Communication parameters were set to reflect both local- and wide-area connectivity. End-user devices were assumed to connect to fog servers through gateways using LAN

links based on Gigabit Ethernet technology, while fog–cloud connections were modeled over WAN links. Latency and bandwidth values for these links are listed in Table 3.

Table 3: Communication parameters.

Link Type	Bandwidth (Mbps)	Latency (ms)
LAN	1000	0.5
WAN	100	30

The computing resources of fog and cloud nodes were modeled with heterogeneous capacities to capture realistic variations. Fog nodes were assigned CPU processing rates, memory sizes, and associated costs by sampling values from specified intervals, while cloud resources were provisioned with higher performance and associated costs. This process was repeated across all nodes to build a diverse system configuration. Table 4 lists the parameter ranges used for fog and cloud resource.

Table 4: Resource specifications of fog and cloud nodes.

Attribute	Fog Nodes (Range)	Cloud Nodes (Range)	Unit
CPU processing rate	[500–1500]	[1000–5000]	MIPS
CPU cost per unit	[0.1–0.4]	[0.7–1.0]	G$
Memory capacity	2–8	8–64	GB
Memory cost per unit	[0.01–0.03]	[0.02–0.05]	G$
Bandwidth cost per unit	[0.01–0.02]	[0.05–0.1]	G$

Task instances were generated using the same parameter ranges and sampling strategy reported in [36] that represents a diverse set of IoT workloads. All task attributes were sampled independently from a uniform distribution within their corresponding intervals, enabling the generation of heterogeneous task profiles ranging from lightweight sensing tasks to computation- and data-intensive applications without bias toward specific workload characteristics.

The adopted parameter ranges are selected to cover a broad and representative spectrum of IoT workloads and fog–cloud resource configurations commonly considered in the literature. This design allows meaningful evaluation and comparison of algorithmic performance under identical conditions.

To ensure fairness, HS–NSGA and the baseline algorithms were executed under identical experimental conditions, using a population size of 100 and a maximum of 300 iterations. All executions were performed sequentially, without parallelization. The baseline algorithms were reimplemented by the authors based on their respective publications, and reported parameter settings were adopted from the original studies wherever available. For parameters not explicitly specified, theoretically optimal values reported in the literature were adopted to avoid tuning bias and ensure fairness and consistency in the comparative evaluation. The parameter settings used for HS–NSGA in all experiments are summarized in Table 5.

The HS control parameters were selected based on preliminary tuning experiments conducted to balance global exploration and local refinement. In these experiments, the key HS parameters, HMCR, PAR, and FW, were varied independently while keeping the remaining parameters fixed. The impact of these parameters on optimization performance was evaluated in terms of best execution cost and makespan. Fig. 3a–c shows the results of these experiments.

Table 5: Parameter settings.

Parameter	Description	Value
Population size (N)	Number of solutions per generation	100
Max iterations (MI)	Termination criterion	300
Crossover probability	GA crossover rate	0.5
Mutation probability	GA mutation rate	0.1
HMCR	Harmony memory consideration Rate	0.99
PAR	Pitch adjustment rate	0.10
FW	Fret bandwidth	0.10

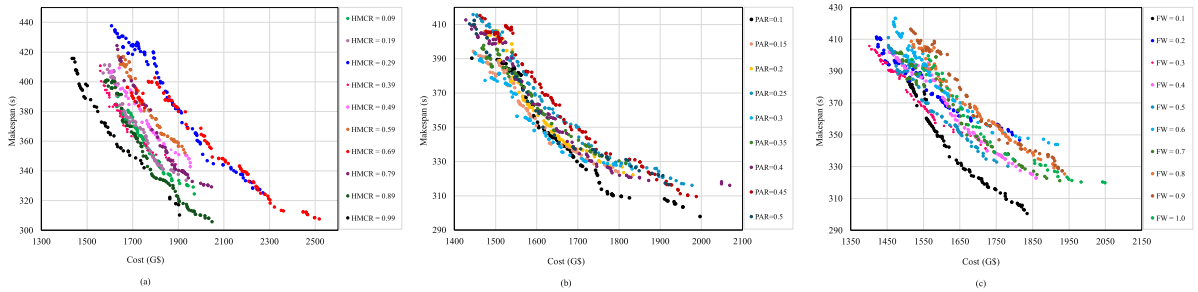

Figure 3: Sensitivity analysis used for tuning HS parameters (**a**) HMCR; (**b**) PAR (**c**) FW; under a setup involving 120 tasks, 10 fog and 3 cloud nodes.

The Pareto-front curves in Fig. 3a–c indicate that higher HMCR values improve solution quality by promoting effective exploitation, while moderate values of PAR and FW provide controlled local refinement. Based on these observations, HMCR, PAR, and FW were set to 0.99, 0.10, and 0.10, respectively, and fixed across all experimental scenarios. In contrast, the genetic algorithm crossover and mutation probabilities were determined during the algorithm design phase as part of the collaborative hybrid framework, with the objective of balancing computational effort between the HS-based exploration component and the GA-based diversification operators. These parameters were therefore not subject to sensitivity tuning and were kept fixed across all experiments.

7 Results and Discussion

7.1 Pareto Front Generation and Trade-off Analysis

As the efficiency of fog–cloud placement is influenced by workload intensity and resource distribution, the initial experiment focuses on evaluating HS-NSGA across multiple task volumes. In the first scenario, 40 tasks were mapped onto an infrastructure comprising 10 fog nodes and 3 cloud nodes, representing a moderately constrained search space. The task count was then increased to 80 and 120 while keeping the underlying fog-cloud configuration unchanged. These higher workloads enlarge the search space and heighten competition for limited resources, making it more difficult to obtain feasible solutions that satisfy both deadline and memory constraints. The resulting Pareto fronts produced by HS-NSGA are shown in Fig. 4a–c, for task sizes of 40, 80, and 120, respectively.

As shown in Fig. 4a–c, HS-NSGA produces smooth, well-distributed Pareto fronts across all scenarios, demonstrating a consistent inverse relationship between cost and makespan. As the number of tasks increases, both objectives shift upward, indicating higher computational demand and intensified resource

contention. The wider spread observed in the 80- and 120-task cases reflects greater variability in feasible allocations, while the maintained front shape highlights the algorithm's ability to preserve solution diversity and convergence quality even under heavier workloads.

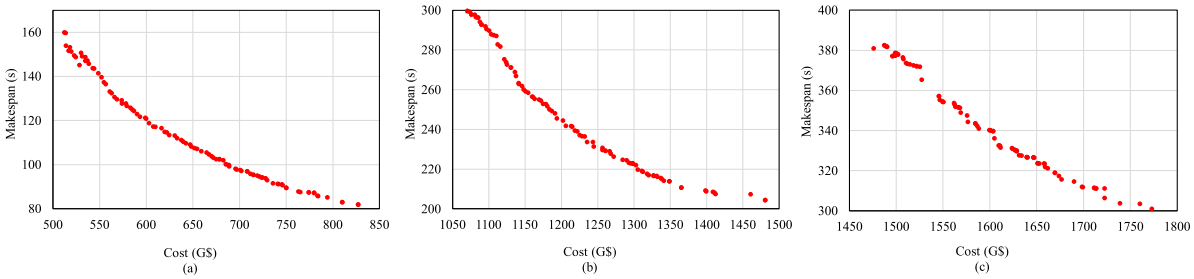

Figure 4: HS-NSGA Pareto fronts for (tasks–fog–cloud) scenarios (**a**) 40–10–3; (**b**) 80–10–3; (**c**) 120–10–3.

To further evaluate the performance of HS-NSGA, its convergence behavior of HS-NSGA was examined by increasing the workload to 200 tasks mapped on the same system of 10 fog nodes and 3 cloud nodes and tracking the reduction in cost and makespan over the iterations. Fig. 5, plots the objective values against the number of iterations and shows consistent reductions for both objectives as the algorithm progresses. The cost curve in Fig. 5a exhibits a rapid decline during the early iterations, followed by gradual stabilization as the search approaches convergence. A similar pattern is observed for makespan Fig. 5b, where fluctuations in the initial stages reflect exploration of the solution space before stabilizing to improved values.

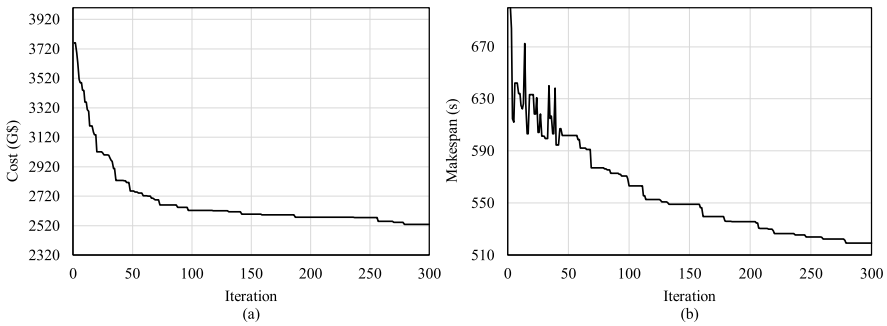

Figure 5: Convergence of (**a**) cost; (**b**) makespan under a setup including 200–10–3 (tasks–fog–cloud).

7.2 Node Scaling Analysis

The second set of experiments was conducted to investigate the behavior of HS-NSGA under different fog-cloud infrastructure sizes, evaluating how the algorithm adapts to larger search spaces and whether it can exploit additional resources to improve the cost-makespan trade-off. In this scenario, the number of fog and cloud nodes was gradually increased while task volumes were fixed at 200 and 300. The Pareto fronts obtained for multiple node configurations are shown in Fig. 6a,b for task sizes of 200 and 300, respectively, where F indicates the total number of fog nodes and C specifies the total number of cloud nodes. As shown in Fig. 6a,b, increasing the number of fog and cloud nodes consistently shifts the Pareto fronts downward and leftward, demonstrating clear improvements in both cost and makespan for the 200- and 300-task scenarios. The well-distributed fronts and the clear dominance ordering among them indicate that HS-NSGA effectively exploits additional resources as the search space expands. However, the performance gains show diminishing returns at the largest configurations, suggesting that beyond a certain scale, additional nodes provide incremental

rather than substantial improvements. Fig. 7a,b provides further insight into this scenario by illustrating the behavior of the best and median solutions across different node configurations for task sizes of 200 and 300, respectively. As expected, both cost and makespan decrease steadily with the addition of more fog and cloud nodes.

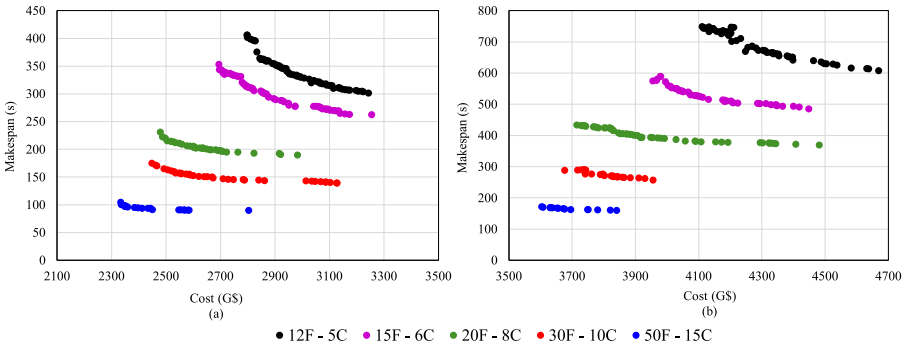

Figure 6: Pareto fronts of HS-NSGA under different fog–cloud node configurations for (**a**) 200-tasks; (**b**) 300-tasks.

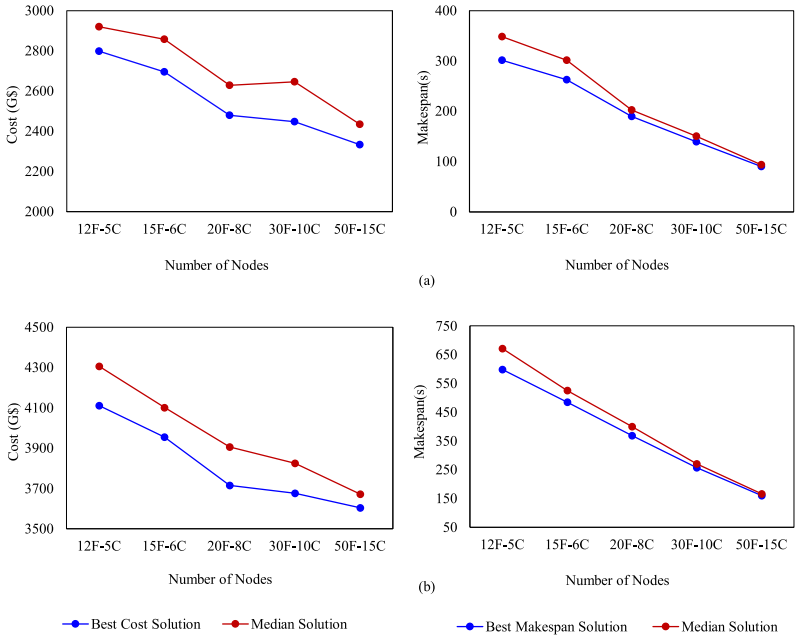

Figure 7: Cost and makespan trends across fog–cloud node configurations for the (**a**) 200-tasks; (**b**) 300-tasks scenarios.

7.3 Analysis of Task Distribution across Fog and Cloud

Beyond evaluating objective values and trade-off behavior, task distribution across fog and cloud resources was examined for a configuration involving 200 tasks, 10 fog nodes, and 3 cloud nodes, using representative solutions from the Pareto set, specifically the best cost solution, the best makespan solution, and a median solution. Fig. 8 summarizes the percentage of tasks executed on fog and cloud nodes in this scenario. The cost-oriented solution allocates nearly all tasks to fog nodes to minimize execution cost, whereas the makespan-oriented solution shifts a larger share of tasks to cloud nodes to exploit their higher processing capacity. The median solution exhibits a more balanced allocation between the two

objectives. Fig. 9a–c provides a node-level view of task allocations corresponding to three representative solutions as best cost, best makespan and median solution, respectively.

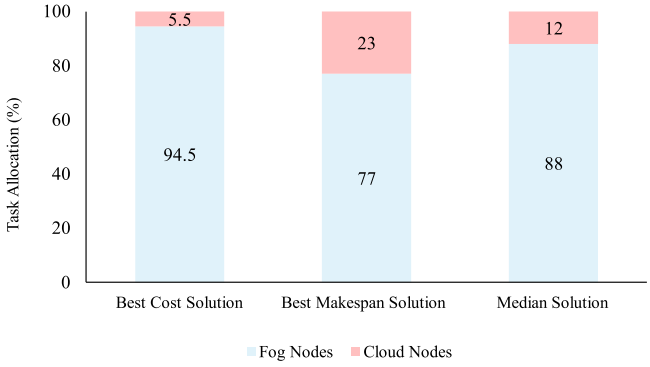

Figure 8: Task allocation percentages between fog and cloud in representative solutions.

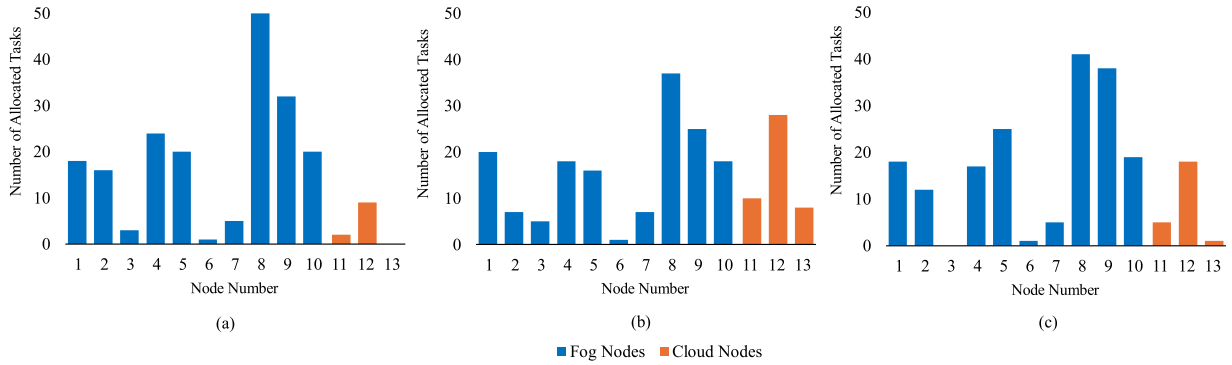

Figure 9: Node-level allocation for three representative solutions (**a**) best cost solution; (**b**) best makespan solution; (**c**) median solution.

7.4 Comparative Analysis

To assess the relative performance of HS-NSGA, we compared it against four genetic and swarm-based placement algorithms: FSPGA and FSPPSO [28], LD-NPGA [29], and GA-FSA [30]. This set of experiments was conducted under varying task volumes (80–500 tasks) with identical fog-cloud configurations. This setup provides a direct comparison of cost and makespan performance while simultaneously examining the scalability of the algorithms. Fig. 10a reports the makespan results obtained by each algorithm across increasing workloads. The results show that HS-NSGA consistently outperforms the baseline methods, achieving a lower makespan across all task sizes. For the cost metric, Fig. 10b shows the total execution cost for each algorithm as the number of tasks increases. As shown in this figure, across all workload sizes, HS–NSGA achieves the lowest cost, outperforming LD-NPGA, GA-FSA, FSPGA, and FSPPSO, providing a more cost-efficient solution while maintaining scalability. To further quantify these results, Table 6 presents the percentage performance improvements obtained by HS-NSGA relative to the baselines across all workload sizes. The table presents per-workload improvements against each comparator.

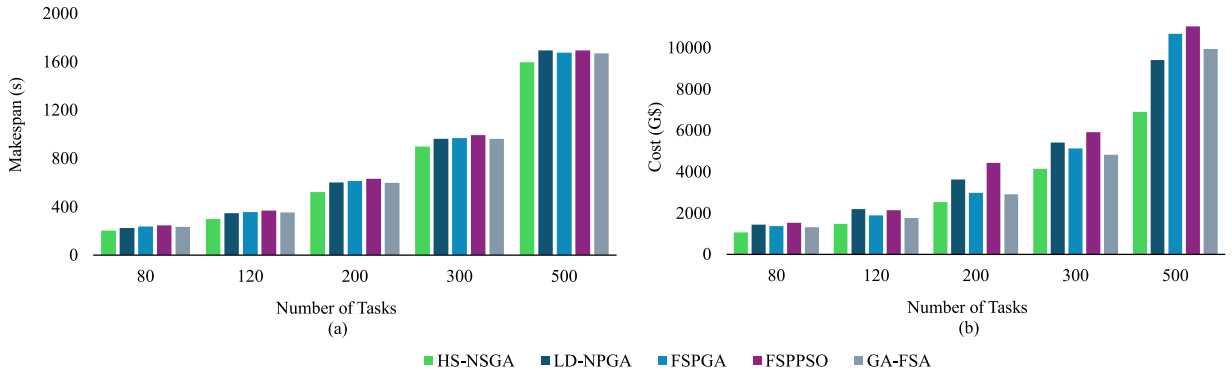

Figure 10: Comparative makespan-cost reduction under varying task sizes (**a**) makespan; (**b**) cost.

Table 6: HS-NSGA percentage improvements in makespan and cost over baseline algorithms.

	Percentage Reduction in Makespan				Percentage Reduction in Cost			
Number of Tasks	**LD-NPGA**	**FSPGA**	**FSPPSO**	**GA-FSA**	**LD-NPGA**	**FSPGA**	**FSPPSO**	**GA-FSA**
80	8.93%	13.81%	17.25%	12.36%	25.77%	22.35%	30.30%	19.17%
120	14.49%	16.75%	19.52%	16.11%	32.69%	21.41%	30.83%	15.98%
200	13.50%	15.03%	17.83%	13.05%	30.14%	14.85%	42.91%	12.81%
300	6.88%	7.40%	9.83%	6.69%	23.40%	19.13%	29.96%	14.18%
500	5.80%	4.63%	5.76%	4.37%	26.73%	35.48%	37.49%	30.66%

8 Discussion

The comprehensive performance evaluation of HS-NSGA demonstrated its effectiveness across diverse fog-cloud scenarios with varying workload and infrastructure scales. Beginning with smaller configurations, HS-NSGA effectively generated Pareto fronts that captured a wide spectrum of trade-offs between makespan and cost. In the initial 40-task scenario shown in Fig. 4a, the algorithm produced a well-distributed front, demonstrating its ability to balance exploration and refinement within a moderately constrained search space. As the number of tasks increased to 80 and 120, as shown in Fig. 4b,c, the Pareto fronts shifted outward as expected due to higher workload demands. Nevertheless, HS-NSGA maintained both solution diversity and convergence quality, confirming its ability to scale to larger workloads. The convergence behavior analysis further highlighted the algorithm's ability to steadily improve solution quality across iterations for both cost and makespan, as shown in Fig. 5. In the early stages, rapid improvements were observed, followed by gradual stabilization as the search approached convergence. This pattern indicates that the hybridization of HS with genetic operators enables the algorithm to balance global exploration with local exploitation, avoiding premature convergence while still achieving stable results within the maximum iteration period.

In terms of infrastructure scalability, the Pareto fronts generated across larger systems showed consistent improvements in both cost and makespan as additional resources became available. Importantly, HS-NSGA maintained front diversity regardless of the system size, underscoring its ability to adapt to broader search spaces. Analysis of best and median solutions across scaling scenarios confirmed that the algorithm systematically benefits from additional resources, demonstrating its suitability for deployment in heterogeneous and evolving fog-cloud environments (Figs. 6 and 7).

Beyond objective values, the task allocation patterns were analyzed to gain deeper insights into how HS-NSGA distributes workloads across fog and cloud resources (Figs. 8 and 9). Examination of representative solutions revealed clear differences in placement strategies. Cost-oriented solutions concentrated tasks on fog nodes to minimize execution cost, whereas makespan-oriented solutions shifted tasks toward cloud nodes to exploit their higher computational capacity. The median solution exhibited a balanced allocation between fog and cloud, illustrating the algorithm's flexibility in adapting placement strategies according to optimization objectives and resource utilization decisions.

Finally, comparative evaluations against LD-NPGA, FSPGA, FSPPSO, and GA-FSA demonstrated the superiority of HS-NSGA across both cost and makespan metrics. As shown in Fig. 9 and Table 7, HS-NSGA consistently achieved lower objective values across increasing task volumes. The percentage reduction analysis further quantified these improvements, with makespan reductions of up to 14.5% over LD-NPGA, 16.8% over FSPGA, 19.5% over FSPPSO, and 16.1% over GA-FSA, and cost reductions of up to 32.7% over LD-NPGA, 35.5% over FSPGA, 42.9% over FSPPSO, and 30.7% over GA-FSA, confirming consistent advantages across all baseline algorithms. These results emphasize that integrating memory-driven search with evolutionary operators in a low-level collaborative structure yields a more effective optimization process, enabling cost-efficient and deadline-aware placement in resource-constrained fog computing environments.

Table 7: Statistical comparison between HS-NSGA and baseline algorithms.

System Size	LD-NPGA		FSPGA		FSPPSO		GA-FSA	
	p-value		*p*-value		*p*-value		*p*-value	
	Best Cost	Best Makespan	Best Cost	Best Makespan	Best Cost	Best Makespan	Best Cost	Best Makespan
80 × 10 × 3	1.15E−06	3.27E−04	2.55E−05	4.81E−04	1.69E−06	3.11E−05	2.08E−05	1.24E−04
120 × 10 × 3	1.72E−06	2.83E−04	1.36E−05	7.16E−04	1.71E−06	2.60E−05	1.13E−05	7.71E−04
200 × 10 × 3	1.73E−06	3.33E−02	1.73E−06	3.25E−02	1.70E−06	3.16E−03	1.74E−06	3.06E−03
300 × 10 × 3	1.23E−06	3.18E−02	1.01E−06	3.83E−02	1.27E−06	6.30E−03	1.78E−06	1.58E−03
500 × 10 × 3	1.49E−06	9.69E−03	2.02E−06	1.01E−02	1.47E−06	9.43E−03	1.93E−06	6.23E−03

9 Wilcoxon Statistical Signed-Rank Test

To statistically validate the performance differences between the proposed HS–NSGA algorithm and the baseline algorithms, a Wilcoxon signed-rank test was conducted. For each system configuration, paired samples were constructed using the best cost and best makespan values obtained by HS–NSGA and each baseline algorithm across independent runs. The null hypothesis assumes that there is no statistically significant difference between the performance of HS–NSGA and the compared baseline algorithm. A significance level of 0.05 was adopted for all tests. The results of the test are summarized in Table 7.

As shown in Table 7, the *p*-values for both the best cost and best makespan metrics are consistently below the 0.05 significance threshold across all comparisons, providing evidence against the null hypothesis and supporting the statistical validity of the observed performance differences as the problem size increases. The boxplots in Fig. 11a,b further support these findings by illustrating the distribution of best cost and best makespan obtained by each algorithm for a representative scenario involving 120 tasks, 10 fog nodes, and 3 cloud nodes. As shown in the figure, HS–NSGA exhibits lower median values and reduced dispersion compared to the baseline methods, indicating more stable and consistent performance. In contrast, the baseline algorithms show wider interquartile ranges, reflecting higher variability and less consistent solution quality.

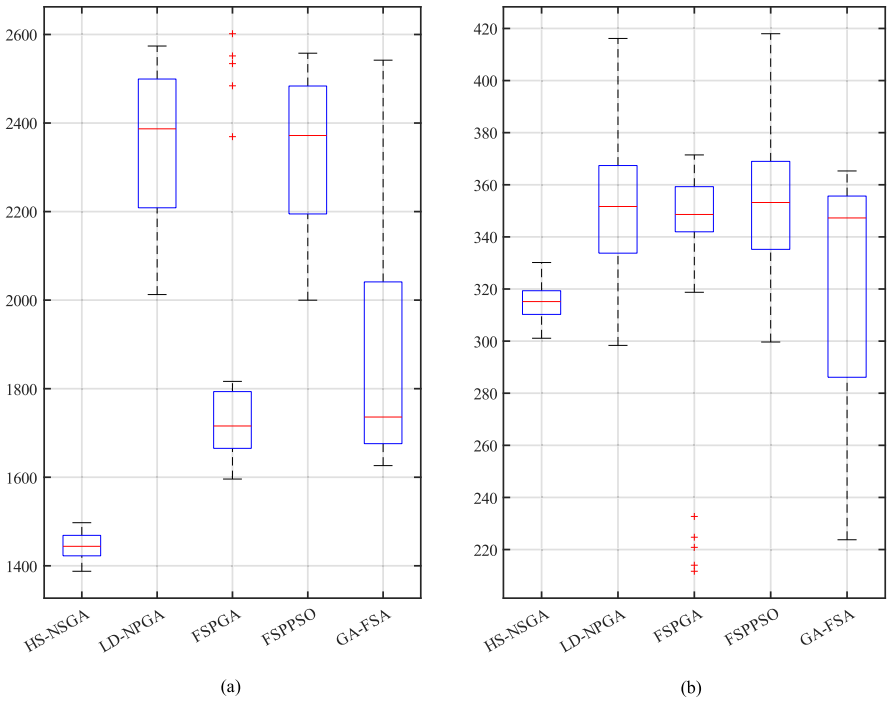

Figure 11: Boxplots of the (**a**) best cost; (**b**) best makespan; obtained by HS–NSGA and the baseline algorithms across independent runs for a deployment scenario involving 120 tasks, 10 fog nodes, and 3 cloud nodes.

10 Limitations and Future Directions

While the experimental results demonstrate the effectiveness of HS-NSGA, certain limitations should be acknowledged. First, the evaluation was conducted on synthetically generated datasets derived from parameter ranges reported in prior studies. While this approach ensures comparability with the literature, validation on real-world IoT workloads would be appropriate for assessing practical applicability and deployment. Second, HS-NSGA was evaluated without incorporating parallel execution strategies. Although this evaluation ensured consistency with the baseline methods, parallel and distributed implementations in real fog-cloud infrastructures could be explored to fully utilize system resources. Third, the current study focused primarily on makespan and cost, whereas other aspects such as energy efficiency and security were not considered and remain open challenges. As a future direction, the model can be extended to incorporate energy consumption at both the device and fog layers, as well as security mechanisms at the physical IoT level, thereby broadening its applicability to practical deployment scenarios.

11 Conclusion

This paper introduced HS-NSGA, a hybrid multi-objective algorithm that integrates HS with genetic operators in a low-level collaborative evolutionary loop for IoT application placement in fog-cloud environments. The design was motivated by the need for algorithms that balance comprehensive exploration with targeted local refinement while satisfying memory and deadline constraints. By combining the memory consideration and pitch adjustment features of HS with the evolutionary operators and selection mechanisms of NSGA-II, HS-NSGA achieves both effective global exploration and localized refinement. Experimental results demonstrated the effectiveness of HS-NSGA across diverse scenarios. The algorithm consistently produced well-distributed Pareto fronts under varying task volumes, exhibited robust convergence behavior, and preserved solution quality as the system scale increased. Analysis of resource allocation patterns revealed

that HS-NSGA adapts its placement strategy to the optimization objective, prioritizing fog nodes in cost-oriented solutions, cloud nodes in makespan-oriented solutions, and balanced allocations in median cases. Comparative evaluations further confirmed that HS-NSGA outperforms comparable hybrid algorithms such as LD-NPGA, FSPGA, FSPPSO, and GA-FSA in both cost and makespan across a range of workload intensities. Future work will focus on extending the algorithm to dynamic and real-world scenarios and on incorporating objectives such as energy efficiency and security, thereby strengthening the applicability of HS-NSGA in practical fog-cloud deployments.

Acknowledgement: Not applicable.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: The authors confirm contribution to the paper as follows: Conceptualization, Zahra Farhadpour, Tan Fong Ang and Chee Sun Liew; methodology, Zahra Farhadpour, Tan Fong Ang and Chee Sun Liew; validation, Zahra Farhadpour, Tan Fong Ang; formal analysis, Zahra Farhadpour, Tan Fong Ang; investigation, Zahra Farhadpour, Tan Fong Ang and Chee Sun Liew; writing—original draft preparation, Zahra Farhadpour; writing—review and editing, Zahra Farhadpour, Tan Fong Ang and Chee Sun Liew; visualization, Zahra Farhadpour; supervision, Tan Fong Ang and Chee Sun Liew. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: Data available on request from the authors.

Ethics Approval: Not applicable.

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

References

1. Al-Tarawneh MAB. Bi-objective optimization of application placement in fog computing environments. J Ambient Intell Humaniz Comput. 2022;13(1):445–68. doi:10.1007/s12652-021-02910-w.
2. Afrin S, Rafa SJ, Kabir M, Farah T, Bin Alam MS, Lameesa A, et al. Industrial Internet of Things: implementations, challenges, and potential solutions across various industries. Comput Ind. 2025;170(4):104317. doi:10.1016/j.compind.2025.104317.
3. Sarrafzade N, Entezari-Maleki R, Sousa L. A genetic-based approach for service placement in fog computing. J Supercomput. 2022;78(8):10854–75. doi:10.1007/s11227-021-04254-w.
4. Kashani MH, Mahdipour E. Load balancing algorithms in fog computing. IEEE Trans Serv Comput. 2023;16(2):1505–21. doi:10.1109/TSC.2022.3174475.
5. Ghafari R, Mansouri N. Swarm intelligence techniques and their applications in fog/edge computing: an in-depth review. Artif Intell Rev. 2025;58(11):359. doi:10.1007/s10462-025-11351-2.
6. Smolka S, Mann ZÁ. Evaluation of fog application placement algorithms: a survey. Computing. 2022;104(6):1397–423. doi:10.1007/s00607-021-01031-8.
7. Herrera JL, Scotece D, Galán-Jiménez J, Berrocal J, Di Modica G, Foschini L. MUMIPLOP: optimal multi-core IoT service placement in fog environments. Computing. 2025;107(4):107. doi:10.1007/s00607-025-01460-9.
8. Kopras B, Bossy B, Idzikowski F, Kryszkiewicz P, Bogucka H. Task allocation for energy optimization in fog computing networks with latency constraints. IEEE Trans Commun. 2022;70(12):8229–43. doi:10.1109/TCOMM.2022.3216645.
9. Tavousi F, Azizi S, Ghaderzadeh A. A fuzzy approach for optimal placement of IoT applications in fog-cloud computing. Clust Comput. 2022;25(1):303–20. doi:10.1007/s10586-021-03406-0.
10. Ajibola OO, El-Gorashi TEH, Elmirghani JMH. Disaggregation for energy efficient fog in future 6G networks. IEEE Trans Green Commun Netw. 2022;6(3):1697–722. doi:10.1109/TGCN.2022.3160397.
11. Abdulazeez DH, Askar SK. Offloading mechanisms based on reinforcement learning and deep learning algorithms in the fog computing environment. IEEE Access. 2023;11(4):12555–86. doi:10.1109/ACCESS.2023.3241881.

12. Goudarzi M, Palaniswami M, Buyya R. A distributed deep reinforcement learning technique for application placement in edge and fog computing environments. IEEE Trans Mob Comput. 2023;22(5):2491–505. doi:10.1109/TMC.2021.3123165.
13. Wang Z, Goudarzi M, Gong M, Buyya R. Deep Reinforcement Learning-based scheduling for optimizing system load and response time in edge and fog computing environments. Future Gener Comput Syst. 2024;152(6):55–69. doi:10.1016/j.future.2023.10.012.
14. Lera I, Guerrero C. Multi-objective application placement in fog computing using graph neural network-based reinforcement learning. J Supercomput. 2024;80(19):27073–94. doi:10.1007/s11227-024-06439-5.
15. Khosroabadi F, Fotouhi-Ghazvini F, Fotouhi H. SCATTER: service placement in real-time fog-assisted IoT networks. J Sens Actuator Netw. 2021;10(2):26. doi:10.3390/jsan10020026.
16. Zhao D, Zou Q, Boshkani Zadeh M. A QoS-aware IoT service placement mechanism in fog computing based on open-source development model. J Grid Comput. 2022;20(2):12. doi:10.1007/s10723-022-09604-3.
17. Aljohani A, Sakellariou R. Improved application placement in fog environments through parallel collaboration. IEEE Access. 2024;12(4):177379–97. doi:10.1109/ACCESS.2024.3505797.
18. Alwabel A, Swain CK. Deadline and energy-aware application module placement in fog-cloud systems. IEEE Access. 2024;12:5284–94. doi:10.1109/ACCESS.2024.3350171.
19. Ali Khan Z, Aziz IA. Dynamic OBL-driven whale optimization algorithm for independent tasks offloading in fog computing. High Confid Comput. 2025;5(4):100317. doi:10.1016/j.hcc.2025.100317.
20. Zare M, Elmi Sola Y, Hasanpour H. Towards distributed and autonomous IoT service placement in fog computing using asynchronous advantage actor-critic algorithm. J King Saud Univ Comput Inf Sci. 2023;35(1):368–81. doi:10.1016/j.jksuci.2022.12.006.
21. Singh S, Vidyarthi DP. A hybrid model using JAYA-GA metaheuristics for placement of fog nodes in fog-integrated cloud. J Ambient Intell Humaniz Comput. 2024;15(7):3035–52. doi:10.1007/s12652-024-04796-w.
22. Altunay R, Bay OF. Using metaheuristic OFA algorithm for service placement in fog computing. Comput Mater Contin. 2023;77(3):2881–97. doi:10.32604/cmc.2023.042340.
23. Wei Z, Mao J, Li B, Zhang R. Privacy-preserving hierarchical reinforcement learning framework for task offloading in low-altitude vehicular fog computing. IEEE Open J Commun Soc. 2025;6:3389–403. doi:10.24433/CO.2570132.v3.
24. Uzer MS. New hybrid approaches based on swarm-based metaheuristic algorithms and applications to optimization problems. Appl Sci. 2025;15(3):1355. doi:10.3390/app15031355.
25. Almasarwah N. Optimized green unrelated parallel machine scheduling problem subject to preventive maintenance. Designs. 2025;9(2):26. doi:10.3390/designs9020026.
26. Yaghtin M, Javid Y. Genetic algorithm based on greedy strategy in unrelated parallel-machine scheduling problem using fuzzy approach with periodic maintenance and process constraints. Int J Supply Oper Manag. 2023;10(3):319–36. doi:10.22034/ijsom.2023.109377.2359.
27. Yan T, Lu F, Wang S, Wang L, Bi H. A hybrid metaheuristic algorithm for the multi-objective location-routing problem in the early post-disaster stage. J Ind Manag Optim. 2023;19(6):4663–91. doi:10.3934/jimo.2022145.
28. Apat HK, Sahoo B, Goswami V, Barik RK. A hybrid meta-heuristic algorithm for multi-objective IoT service placement in fog computing environments. Decis Anal J. 2024;10(1):100379. doi:10.1016/j.dajour.2023.100379.
29. Bernard L, Yassa S, Alouache L, Romain O. Efficient pareto based approach for IoT task offloading on Fog–Cloud environments. Internet Things. 2024;27(4):101311. doi:10.1016/j.iot.2024.101311.
30. Alwabel A, Swain CK. Hybrid approach for cost efficient application placement in fog-cloud computing environments. Comput Mater Contin. 2024;79(3):4127–48. doi:10.32604/cmc.2024.048833.
31. Ben Rjeb H, Sliman L, Zorgati H, Ben Djemaa R, Dhraief A. Optimizing Internet of Things services placement in fog computing using hybrid recommendation system. Future Internet. 2025;17(5):201. doi:10.3390/fi17050201.
32. Sharma A, Thangaraj V. A lightweight trajectory aware application placement in IoT-fog-cloud environment. J Grid Comput. 2025;23(1):8. doi:10.1007/s10723-025-09794-6.
33. Farhadpour Z, Ang TF, Liew CS. Decision-making in fog computing: a comparative analysis of Pareto-based optimization and weighted objective models for application placement. In: 2025 17th International Conference

on Computer and Automation Engineering (ICCAE); 2025 Mar 20–22; Perth, Australia. p. 314–20. doi:10.1109/ICCAE64891.2025.10980559.
34. Geem ZW, Kim JH, Loganathan GV. A new heuristic optimization algorithm: harmony search. Simulation. 2001;76(2):60–8. doi:10.1177/003754970107600201.
35. Deb K, Pratap A, Agarwal S, Meyarivan T. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput. 2002;6(2):182–97. doi:10.1109/4235.996017.
36. Farhadpour Z, Ang TF, Liew CS. Hybrid evolutionary optimization for efficient placement of IoT applications in fog computing environments. Peerj Comput Sci. 2026;12(3):e3603. doi:10.7717/peerj-cs.3603.
37. Goudarzi M, Wu H, Palaniswami M, Buyya R. An application placement technique for concurrent IoT applications in edge and fog computing environments. IEEE Trans Mob Comput. 2021;20(4):1298–311. doi:10.1109/TMC.2020.2967041.
38. Yaghtin M, Javid Y, Ardakan MA. Multi-objective optimization in the design of load sharing systems with mixed redundancy strategies under random shocks. J Comput Sci. 2025;85(1):102495. doi:10.1016/j.jocs.2024.102495.
39. Pallewatta S, Kostakos V, Buyya R. Reliability-aware proactive placement of microservices-based IoT applications in fog computing environments. IEEE Trans Mob Comput. 2024;23(12):11326–41. doi:10.1109/TMC.2024.3394486.
40. Yousefpour A, Patil A, Ishigaki G, Kim I, Wang X, Cankaya HC, et al. FOGPLAN: a lightweight QoS-aware dynamic fog service provisioning framework. IEEE Internet Things J. 2019;6(3):5080–96. doi:10.1109/JIOT.2019.2896311.
41. Nguyen BM, Thi Thanh Binh H, The Anh T, Bao Son D. Evolutionary algorithms to optimize task scheduling problem for the IoT based bag-of-tasks application in cloud-fog computing environment. Appl Sci. 2019;9(9):1730. doi:10.3390/app9091730.