Cureus Journal of Computer Science
https://doi.org/10.7759/s44389-026-00394-3

Original Article

Integration of Migration Policy for Enhanced Quality of Service Differential Evolution-Based Approach for Transfer of Tasks From Mobile Edge Computing Devices in the Mobile Edge Cloud Environment

Santanu Dam [1], Gopa Mandal [2], Om Bhattarai [2], Soumyadeep De [2], Kousik Dasgupta [3], Abir Chattopadhyay [4]

1. *Higher Education, Netaji Subhas Open University, Kolkata, IND*
2. *Department of Computer Science and Engineering, Jalpaiguri Government Engineering College, Jalpaiguri, IND*
3. *Department of Computer Science and Engineering, Kalyani Government Engineering College, Kalyani, IND*
4. *Department of Electronics and Communication Engineering, Institute of Engineering and Management, University of Engineering and Management, Kolkata, IND*

Received: April 20, 2026 | Review began: May 04, 2026 | Review ended: November 25, 2026 | Published: November 26, 2026

© **Copyright** 2026
This is an open access article distributed under the terms of the Creative Commons Attribution License CC-BY 4.0., which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Abstract

Today's cloud architecture integrates the Internet and Cloud of Things (IoT & CoT), as well as combines edge and fog devices that require extensive data processing in a timely manner. Mobile edge computing (MEC) delegates resource-demanding activities from the edge devices with limited energy and computing to nearby servers to speed up execution. Generally, task offloading reduces latency and energy consumption. The basic objective is to enhance the quality of service (QoS) of the dynamic and heterogeneous cloud environment. Several heuristic and rule-based dynamic offloading policies exist, but they frequently fail to address the contradiction between exploring varied solution domains and exploiting best-quality solutions, leading to unbalanced task allocation. In our earlier research, we used a population-based metaheuristic called differential evolution (DE) for offloading tasks. A specific algorithm does not always do better performance. This variability is because the dynamic situation is not always the same. This extended research builds upon the existing DE-based task offloading framework by introducing a dynamic migration policy to further enhance QoS in heterogeneous cloud environments. However, our previously proposed DE algorithm effectively optimized task offloading decisions. But inclusion of the proposed migration policy enables real-time task reallocation in response to dynamic network conditions, server loads, and task urgency. Experimental results demonstrate significant improvements in latency reduction, energy efficiency, and system stability, particularly under high workload conditions.

Categories: Green Computing and Sustainability, Cloud Migration, Edge Computing

Keywords: mec, eot, task offloading, iot, cot, edge server, migration

Introduction

Task offloading is a key mechanism in MEC for resource-constrained mobile, sensor, and IoT devices [1,2,3]. By transferring computation-intensive tasks from end devices to the closest edge or cloud resources [2], this offloading can reduce execution time and device-side energy requirements while improving overall QoS [1-3,4,5]. Modern heterogeneous computing environments increasingly integrate IoT or edge-of-things devices, edge and fog nodes, mobile or edge

servers, and remote cloud settings so that computation can be distributed across layers according to application requirements and resource availability [6-9]. Bringing computation closer to these devices is essential for providing cloud-related services on demand [10,11].

The configuration of contemporary cloud architecture is illustrated in Figure *1*. In a typical hierarchical organization, the edge layer contains end devices that generate data, intermediate edge/fog resources provide processing and short-term services closer to users, and the cloud layer supplies large-scale computation and storage [1,11]. Such a layered organization can reduce unnecessary long-distance data transfer and support low-latency processing, bandwidth efficiency, and flexible task offloading [1,10,11]. However, the effective placement of tasks in MEC must account for changing computational demand, server capacity, communication bandwidth, energy consumption, and deadline constraints [1-3,7,8,10,12]. This approach makes resource management a problem not only of initial allocation but also of continuous adaptation in a dynamic environment where workloads and network conditions change over time [3,8,10]. Edge servers are of particular importance, as their proximity to end users can shorten communication paths, reduce response delay, and reduce the amount of computation performed on resource-constrained devices [1,2,10]. But proximity alone does not ensure balanced operation. Heterogeneous server capacities, time-varying workloads, and uneven resource availability can create hotspots or underutilized resources, making dynamic load distribution an important QoS consideration [3,8,13].

Numerous task-offloading and workload-scheduling approaches have therefore been investigated, including optimization-based methods, evolutionary and swarm metaheuristics, collaborative edge approaches, mobility-aware techniques, and learning-based dynamic offloading [1-4,7,8,10,11]. These methods have demonstrated improvements in latency, energy consumption, resource utilization, or adaptability; however, no single strategy can eliminate the difficulty of maintaining a satisfactory allocation when workload intensity, server utilization, task urgency, or bandwidth changes during execution [1,3,8,10,11]. Learning-based approaches can improve adaptation but may require model training or retraining when operating conditions change [10]. In contrast, deadline-aware and load-balancing studies indicate that QoS also depends on meeting time constraints and maintaining equitable resource utilization [12-13].

Our earlier work introduced an intelligent collaborative metaheuristic task-offloading approach that used Differential Evolution (DE) and Particle Swarm Optimization (PSO) within a machine-learning-assisted selection framework [14]. Building on that work, the present study retains DE as the initial task-allocation optimizer. A key limitation of the initial offloading decision, however, is that the assignment may become suboptimal once execution begins. If a server becomes overloaded, a task approaches its deadline, or the available bandwidth deteriorates, a previously acceptable assignment may produce longer queues, increased energy use, resource imbalance, or deadline violations. Thus, the present work focuses on extending DE-based offloading from an initial optimization mechanism to a dynamically correctable allocation process.

To address these limitations, this paper proposes Differential Evolution with Migration (DE-M), which combines DE-based initial task allocation with a dynamic migration manager. The manager continuously observes server utilization, task-deadline proximity, and available bandwidth. Migration is triggered when server utilization exceeds 85%, when the remaining execution time falls below 20% of the task deadline, or when bandwidth drops more than 40% from its baseline value. The enhanced fitness function includes a migration-cost model that incorporates data-transfer delay, migration energy, and service downtime, allowing reassignment to be considered together with its associated overhead. In this way, DE-M is designed to correct allocations during execution rather than relying exclusively on the initial mapping.

The remainder of the paper is organized as follows. The next section reviews related task-offloading and scheduling studies and identifies the associated research gaps. The Materials and Methods section formulates the MEC task-offloading problem and presents the DE formulation, migration triggers, migration-cost model, enhanced fitness function, and the DE-M algorithms. The Results and Discussion section describes the simulation environment and compares DE-M with the baseline policy, GA, GWO, ACOR, PSO, and conventional DE. The last section concludes our strategy with probable and future research directions.

How to cite this article:
Dam S, Mandal G, Bhattarai O, et al. (November 26, 2026) Integration of Migration Policy for Enhanced Quality of Service Differential Evolution-Based Approach for Transfer of Tasks From Mobile Edge Computing Devices in the Mobile Edge Cloud Environment. Cureus J Comput Sci 3 : es44389-026-00394-3. DOI https://doi.org/10.7759/s44389-026-00394-3

This paper summarizes its main contributions as follows:

1. A migration-enabled DE framework (DE-M) is proposed to transform an initially optimized but effectively static task allocation into an adaptive allocation process that can react during execution.
2. A multi-trigger migration mechanism is introduced using server overload, task urgency, and network deterioration, thereby directly addressing the inability of initial-only offloading torespond to changing server and communication conditions.
3. Migration overhead is explicitly represented through data-transfer delay, migration energy, and service downtime and is incorporated with deadline-violation considerations in the enhanced fitness formulation, enabling migration decisions to account for both QoS benefit and reassignment cost.

Experimental evaluation over 6K, 12K, and 24K task sizes shows acceptable time overhead improvement over conventional DE. At 24K tasks, DE-M reduces average waiting time by approximately 17.6%, response time by approximately 5.3%, and energy consumption per task by approximately 9.0%; increases the offloading rate from 89.34% to 91.45%; and reduces overload occurrences from 0.2 to 0.1. In the QoS-centric evaluation, the deadline violation rate decreases from 8.4% to 3.2%, while Jain's Fairness Index increases from 0.73 to 0.89. These results show that responsiveness, energy efficiency, load balancing, and QoS perform better under heavier MEC workloads.

How to cite this article:
Dam S, Mandal G, Bhattarai O, et al. (November 26, 2026) Integration of Migration Policy for Enhanced Quality of Service Differential Evolution-Based Approach for Transfer of Tasks From Mobile Edge Computing Devices in the Mobile Edge Cloud Environment. Cureus J Comput Sci 3 : es44389-026-00394-3. DOI https://doi.org/10.7759/s44389-026-00394-3

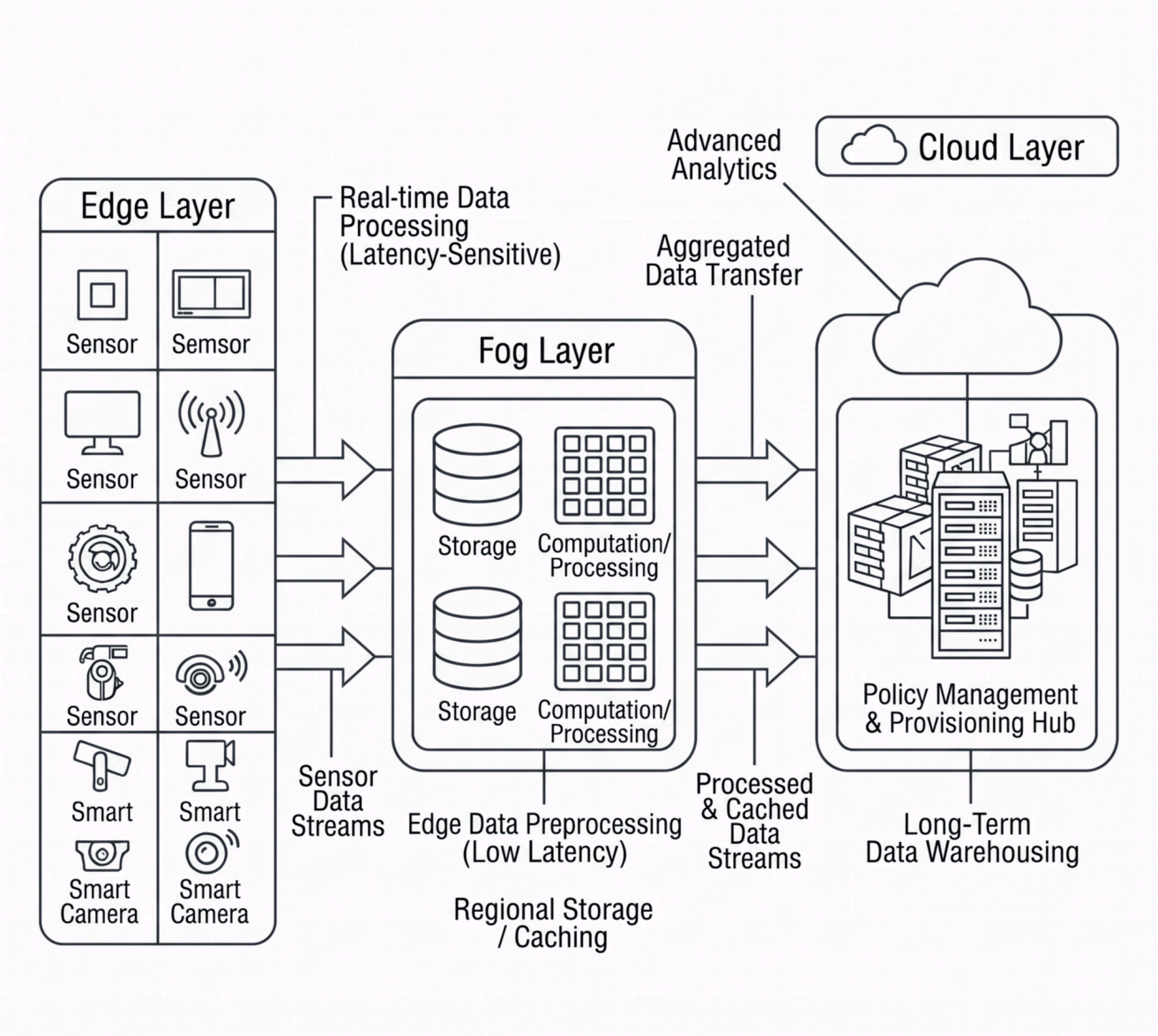


FIGURE 1: Hierarchical structure of a heterogeneous-computing-framework

The figure was developed using data reported by [1].

Related Work

Researchers have proposed different optimal strategies in this domain to enhance the QoS. In 2022, Wang et al. worked on task offloading in MEC systems using an earliest deadline-based first scheduling strategy [9]. This hybrid strategy is a combination of GA with integer coding. The authors have compared their proposed strategy's simulation with traditional heuristics, PSO, and binary GA strategies. They report that their methods have a higher task acceptance rate and utilization of resources. In the same year, Qin et al. [7] offered another GA-based strategy that optimized energy and execution time. For offloading decisions, they have used GA and applied DVFS to lower energy. Their reported results are quite encouraging. Yang et al. [10] divided the joint offloading and bandwidth allocation problem using mixed integer programming. The proposed DSLO method quickly adapts to new situations with just a few examples and gets almost the best system usefulness. In 2023, Li et al. [3] suggested an FC-LSTM model for offloading in MEC. This is a dynamic and adaptive method which optimized workload balance by reducing the weighted energy and delay across cloudlets where device speeds change. Convergence, reduced latency, and enhanced energy efficiency improve. Further, the research of

Zhang et al. [11] divided offloading into two types, i.e., static and dynamic (according to task types). They have also categorized offloading strategy as full or partial. Their survey explained how the type of job affects architecture, goals, and the allocation of resources. At the end, the authors also have discussed the future research scope in task sensitive offloading. A method for decreasing energy consumption and latency of execution was proposed in 2024. It was an efficient task offloading strategy proposed by Alsadie et al. [2]. The authors have shown the results as improved efficiency and task completion and compared the result with conventional methods. In 2024, in a survey, Dong et al. talked about MEC and their uses [1]. They have suggested the way to solve latency problem, consumption of huge energy and security of data incurred from high computing and real-time applications. They broke down task-offloading choices into five main groups: (i) enhanced computing, (ii) energy-delay tradeoffs, (iii) energy efficiency, (iv) latency minimization, and (v) personalized application environments. Nujhat et al. introduced a method called WOLVERINE [4]. The approach suggested a multi-faceted task delegation paradigm for cooperative MEC. The authors have reported the reduction of resource usage cost, execution latency, and energy consumption. In the following year, Prasad et al. [6] proposed a new hybrid method for cloud task scheduling that combines GWO and PSO. They have done their experiment with CloudSim Plus and Google Borg traces, and the results are encouraging. The results reported have an increase of 10% throughput, a decrease of makespan is 15% and better balanced VM loads. Dynamic strategy, which can work in an environment with stochastic, fluctuating task arrivals, was proposed by Rasool et al. in 2024 [8]. The method is based on particle swarm optimization. The authors have aimed to optimize task-to-server allocation by reducing long-term average latency and consumption of energy. They have treated the arrival of tasks and behavior of network as a random process. Simulation of the method shows a 20% less delay and 18% energy savings over baseline methods.

In literature, different researchers have solved different challenges. An efficient method should achieve both low latency and low energy consumption at the same time. Most of the previous techniques have considered just one resource, not the link between different resources. So this is still a challenging problem.

Authors	Year	Title	Problem Addressed	Approach/Method	Key Contributions	Limitations/Gaps
Dong et al. [1]	2024	Survey on Task Offloading Strategies for MEC	Lack of unified framework in MEC offloading	Comprehensive survey and taxonomy	Detailed classification of strategies and trade-offs	No experimental validation
Alsadie et al. [2]	2024	Task offloading in Energy-Constrained Edge Computing	Latency and energy efficiency in edge systems	Hybrid optimization-based task offloading strategy	Improved energy efficiency, reduced latency	Limited real-world validation and scalability
Li et al. [3]	2023	Adaptive Workload Offloading Strategy	variation in workload coming dynamically	Adaptive workload offloading strategy	Improved system responsiveness and efficiency	Complexity in dynamic adaptation
Nujhat et al. [4]	2024	Grey Wolf Optimization	Multi-objective optimization (latency & energy)	GWO	Efficient multi-objective offloading	Simulation-based validation only

Pradhan et al. [5]	2022	DRL and Parallel PSO	Inefficient load balancing in cloud	DRL + Parallel PSO hybrid	Improved load balancing and resource utilization	High computational complexity
Prasad et al. [6]	2025	Hybrid PSO and Grey Wolf Optimization	Task scheduling inefficiency	Hybrid PSO + GWO	Enhanced scheduling performance	Limited experimental validation
Qin et al. [7]	2022	Mobility-Aware and Energy-Efficient	Mobility and energy challenges in MEC	Mobility-aware scheduling	Reduced latency and energy consumption	Limited scalability
Rasool et al. [8]	2024	Optimization using Improved PSO	Time-varying workloads in MEC	Improved PSO	Better optimization under dynamic conditions	Needs real-world testing
Wang et al. [9]	2022	Hybrid GA with Integer Coding	Resource inefficiency in edge computing	GA + EDF	Improved task acceptance and efficiency	Simulation-only validation
Yang et al. [10]	2022	Deep Learning Based Strategy	Dynamic network conditions	Deep learning-based offloading	Adaptive and intelligent decision-making	High training cost
Zhang et al. [11]	2024	Survey of Computation Offloading with Task Types	Lack of task-type-based classification	Survey-based taxonomy	Categorization based on task types	No implementation results
Liu et al. [12]	2023	Dependent Task Scheduling and Offloading	Deadline violations in dependent tasks	Joint scheduling and offloading	Reduced deadline violation ratio	High model complexity
Masoumi et al. [13]	2024	Impact of MEC Locations on Network Performance using Load Balance Metrics	Performance variation due to MEC placement	Load balancing metrics analysis	Insights into optimal MEC placement	Limited focus on offloading algorithms

TABLE 1: Comparative study of various task offloading techniques

EDF, earliest deadline first; GWO, Grey Wolf Optimizer; MEC, mobile edge computing; PSO, particle swarm optimization.

Conventional strategies frequently neglect the fluctuating characteristics of resource consumption and accessibility [14], more investigation is required for optimal use of resources. In Table *1*, we present a comparative study of state-of-the-art approaches:

1. Develop an overhead scalable dynamic approach with low training cost and low control that can adapt to fluctuating workloads and resources.
2. Design offloading and migration policies with consideration to latency, energy consumption, server selection, transmission conditions, and load balancing.
3. Evaluate QoS with other indicators like deadline violation rate, migration overhead, overload frequency, and load-balancing fairness.
4. Design adaptive migration policies that trigger task migration based on server load, task priority, and network performance.
5. Combine migration with the initial optimization process, like DE [15], so that an efficient initial allocation can be corrected during execution.

Materials And Methods

This section describes the methodological framework of the proposed extension of the DE-M approach for adaptive task offloading in MEC environments. It starts with problem formulation and presents the DE-based optimization model [15] for initial task allocation. Here, we introduce the proposed migration policy, which is characterized by dynamic triggers such as server utilization, task urgency, and variation of bandwidth. Then, a migration cost model and an improved fitness function are proposed to balance the QoS improvement and migration overhead. Finally, the detailed operational workflow of the proposed system is provided in the DE-based task offloading manager, the migration decision algorithm, and the integrated DE-M algorithm.

Formulation of the problem

In MEC, different user devices generate high-computation jobs. These tasks or jobs can be executed on those devices or offloaded to neighboring edge servers. This decision to offload is very much critical, as it directly influences execution delay, energy usage, and QoS. Our system consists of a set of multiple mobile users U = {u_1, u_2, …, u_n} and a collection of edge servers S = {s_1, s_2, …, s_m} [15]. Each user task can be described by different parameters. The parameters include task size, measured in processing cycles. Next, size of the data which is calculated in bits. We aim to obtain an efficient offloading policy π that reduces the joint cost function, considering resource constraints and QoS improvements. The joint cost function is commonly stated as a weighted sum of execution time and energy usage. The optimization function is formally stated as: $\min_{\pi} F(\pi) = \alpha \cdot L(\pi) + \beta \cdot E(\pi)$, where $L(\pi)$ is the total task completion time, $E(\pi)$ denotes total energy use, and α and β are control parameters. MEC environment is dynamic and distributed. Thus, a conventional heuristic strategy often results in improper task allocation. A more effective optimization strategy is needed to ensure proper synchronization. Proper synchronization between exploration and exploitation of the solution space is balanced. The application of DE may provide a solution to this problem [15]. Introduction of migration strategy may add more efficiency [15].

Differential evolution formulation

In our work, we have used a DE-based algorithm as proposed by Dam et al. [15]. This is a population-centric optimization process based on a stochastic algorithm. We have utilized this to enhance the task offloading coefficients (*w*). Proposed algorithm progressively enhances potential candidate solutions by utilizing mutation, crossover, and selection operators.

Initialization: In this step, consider the population of N_p candidate solutions to be initialized as delineated in Equation 1.

$$X_1^{(0)} = \left(X_{i1}^{(0)}, X_{i2}^{(0)}, X_{i3}^{(0)}, \ldots, X_{ij}^{(0)}\right) \sim U\left[w_{\min}, w_{\max}\right] \quad (1)$$

for $i = 1, 2, \ldots, N_p$, where w_{min} and w_{max} specify the max. and min.limitations of the weights, and p is the population size.

Mutation: At generation g for each target vector $X_i^{(g)}$, three different individuals $X_a^{(g)}$,$X_b^{(g)}$, and $X_c^{(g)}$ are randomly chosen, and a mutant vector is produced as in Equation 2.

$$V_i^{(g)} = X_a^{(g)} + F \cdot \left(X_b^{(g)} - X_c^{(g)}\right) \tag{2}$$

where $F \in (0, 1]$ is the differential weight.

Crossover: Through binomial crossover, a trial vector is derived as stated in Equation 3.

$$u_{ij}^{(g)} = \begin{cases} v_{ij}^{(g)}, & \text{if } \mathrm{rand}_j \le CR \text{ or } j = j_{\mathrm{rand}}; \\ x_{ij}^{(g)}, & \text{otherwise.} \end{cases} \tag{3}$$

where crossover probability is CR $\in$ (0,1) , rand_j ~ U(0,1), and j_{rand} is an arbitrarily chosen value or index that guarantees that mutant vector is responsible for at least one dimension.

Selection: The trial vector $u_i^{(g)}$ vies against the target vector $x_i^{(g)}$ considering the fitness function. The next generation is modified according to Equation 4.

$$x_i^{(g+1)} = \begin{cases} u_i^{(g)}, & \text{if } f\left(u_i^{(g)}\right) < f\left(x_i^{(g)}\right); \\ x_i^{(g)}, & \text{else.} \end{cases} \tag{4}$$

where f(·) is the fitness function. That aims to reduce effective response time, usage of energy, and overload penalties. Delinated in Equation 5.

$$f(w) = \bar{T}_{\mathrm{resp}} + \lambda^E \cdot \bar{E}_{\mathrm{task}} + \lambda^O \cdot \bar{O} \tag{5}$$

where $\bar{T}_{resp}$ denotes mean response time, $\bar{E}_{task}$ for mean energy/task, average overload count denoted by Ō, and λ^E and λ^O are the trade-off parameters. DE algorithm progresses over G generations and picks the best optimal solutionl, as shown in Equation 6.

$$w^* = \arg\min_{w \in P(G)} f(w) \tag{6}$$

where $P(G)$ developed the ultimate evolved population.

Proposed migration policy

Three triggers are used to dynamically redistribute the tasks in the migration policy: (i) server overload is initiated when utilization is above 85% in an effort to avoid bottlenecks; (ii) migration is initiated by task urgency if the remaining time of a task is less than 20% of its deadline so as to guarantee its timely completion; and (iii) network degradation is triggered by a bandwidth degradation of more than 40% compared to the baseline transmission efficiency. Each trigger is constantly monitoring the system status. Upon exceeding any threshold, potential tasks are identified and assigned to

underutilized or more optimally connected resources. This multi-trigger approach enables proactive QoS management to reduce latency, prevent deadline violations, and keep load balanced in the presence of changing conditions. Table *2* provides a better illustration.

Trigger Type	Condition	Threshold
Server Overload	Server utilization > 85%	$U_{threshold} = 0.85$
Task Urgency	Remaining time < 20% of deadline	$D_{threshold} = 0.2 \times deadline$
Network Deterioration	Bandwidth drop > 40% from baseline	$B_{threshold} = 0.6 \times B_{initial}$

TABLE 2: Migration triggering threshold used in simulation

Term	Symbol	Unit	Description
Data transfer delay	D_{data} / B_{avail}	seconds	Time to move the task's state (code, input data, and intermediate results) from source to destination.
Energy cost	E_{mig}	Joules	Extra energy consumed on the source server, destination server, and network devices during migration.
Service downtime	$\delta_{downtime}$	seconds	The period during which the task is unavailable (e.g., while its state is frozen, transferred, and resumed).

TABLE 3: Migration cost model

Migration cost model

The migration cost is applied to estimate the overhead when a task is dynamically moved from its current server to another host. This cost is included in the enhanced fitness function to guarantee that migration is performed only if the net benefit (improved QoS) is higher than the temporary disruption. The total migration cost (C_{mig}) for a particular task is defined in Equation 7.

$$C_{\mathbf{mig}} = \alpha \cdot \left(\frac{D_{\mathbf{data}}}{B_{\mathbf{avail}}} \right) + \beta \cdot E_{\mathbf{mig}} + \gamma \cdot \delta_{\mathbf{downtime}} \quad (7)$$

How to cite this article:
Dam S, Mandal G, Bhattarai O, et al. (November 26, 2026) Integration of Migration Policy for Enhanced Quality of Service Differential Evolution-Based Approach for Transfer of Tasks From Mobile Edge Computing Devices in the Mobile Edge Cloud Environment. Cureus J Comput Sci 3 : es44389-026-00394-3. DOI https://doi.org/10.7759/s44389-026-00394-3

α, β, and γ are the weighted parameters, in which α is used for latency-sensitive migrations, β penalizes energy-heavy migrations (e.g., battery-powered edge nodes), and γ strongly discourages long service interruptions for real-time tasks. We set the values to 0.25, 0.35, and 0.6 after several experiments. Table 3 illustrates the migration cost model.

Enhanced fitness function

The original fitness function (Equation 5) is extended to Equation 8 to incorporate migration considerations.

$$f_{\text{enhanced}}(w) = \bar{T}_{\text{resp}} + \lambda^{E} \cdot \bar{E}_{\text{task}} + \lambda^{O} \cdot \bar{O} + \lambda^{M} \cdot C_{\text{mig}} + \lambda^{V} \cdot V_{\text{deadline}} \quad (8)$$

where λ^{M} is the migration-cost weight, λ^{V} is the deadline-violation weight, and V_{deadline} is the deadline-violation rate.

DE-based task offloading manager

Migration policy is used to allocate tasks using a DE-based task offloading manager that initially allocates tasks using DE, to optimizing latency and energy. This placement is then fed to the execution layers. The Dynamic Migration Manager monitors three major signals continuously: server utilization, task deadline proximity, and available bandwidth. It consists of four submodules:

Module-1 (monitor) aggregates the required real-time metrics from all nodes;

Module-2 (trigger) evaluator checks all thresholds as discussed in Table 2;

Module-3 (decision) engine selects candidate tasks and target servers based on migration cost models; and

Module-4 (executor) performs live task migration with minimal disruption.

The architecture depicted in Figure 2 is a closed loop: the initial DE placement establishes a good baseline, and the migration layer adaptively corrects the deviations induced by overloads, node failures or network variations. The dual-layer intelligence provides stable QoS even under tremendous stress, enhances load balancing, and minimizes missed deadlines, so making the system truly flexible for heterogeneous edge-cloud environments.

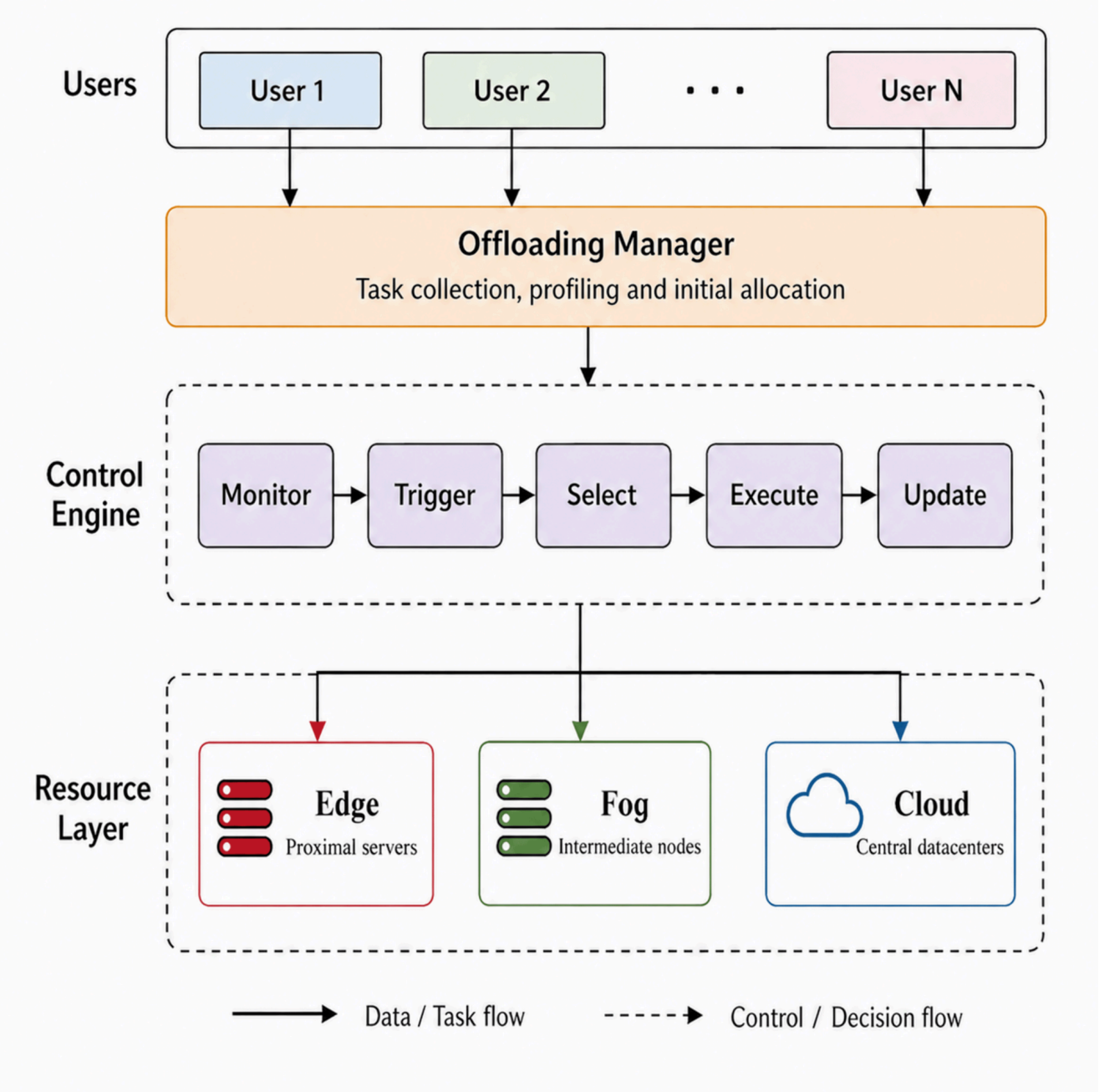


FIGURE 2: Proposed enhancement in the system architecture

Migration decision algorithm

Algorithm 1 formalizes the dynamic migration policy using explicit task, server, allocation, and system-state data structures. The current allocation is represented by A[*t*], which maps each task t to its assigned server. Server utilization U[*s*], current and baseline bandwidth B[*s*] and B0[*s*], task deadline D[*t*], and remaining time R[*t*] are monitored continuously. The trigger thresholds are θ_u = 0.85, θ_D= 0.20, and θ_B = 0.60, consistent with Table *2*. The migration cost $C_{\text{mig}}(t, s, d)$ is evaluated using Equation 7, while the acceptance of a tentative reassignment is checked through the enhanced fitness function in Equation 8.

Algorithm 1: dynamic migration policy

Input: $T = \{t_1, t_2, t_3 \ldots t_n\}$, task list; $S = \{s_1, \ldots, s_m\}$, server set; A, allocation map $T \rightarrow S$; $U[s]$, server-utilization map; $B[s]$ and $B_0[s]$, current and baseline bandwidth maps; $D[t]$ and $R[t]$, deadline and remaining-time maps; $\boldsymbol{\theta_u} = 0.85$, $\boldsymbol{\theta_D} = 0.20$, $\boldsymbol{\theta_B} = 0.60$; $ETC(t,d)$: estimated completion time; $\boldsymbol{C_{mig}(t,s,d)}$, Equation 7; $\boldsymbol{f_{enhanced}}(.)$, Equation 8.

Output: A', updated task-to-server allocation map; L_{mig}, ordered list of executed migrations (task, source, and destination).

1: A' ← A; $\boldsymbol{L_{mig}}$ ← empty list
2: for each server s in S do
3: $\boldsymbol{T_s}$ ← {t in T | A'[t] = s}
4: overload ← (U[s] > $\boldsymbol{\theta_u}$)
5: net_bad ← (B[s] < $\boldsymbol{\theta_B} \times B_0$[s])
6: urgent ← {t in T_s | R[t] < $\boldsymbol{\theta_D} \times$ D[t]}
7: if overload OR net_bad OR urgent ≠ empty then
8: C ← sort T_s by ascending R[t]/D[t]
9: for each t in C do
10: Ft ← {d in S \ {s} | U[d] < $\boldsymbol{\theta_u}$ AND ETC(t,d) ≤ D[t]}
11: if Ft ≠∅ empty then
12: d* ← $\boldsymbol{\arg\min_{d \in F_t} C_{mig}(t,s,d)}$
13: A_{trail} ← A'; A_{trail}[t] ← d*
14: if $f_{enhanced}(A_{trail}) < f_{enhanced}(A')$ then
15: A' ← A_{trail}
16: append (t,s,d*) to L_{mig}
17: update U, B, and R using the new allocation A'
18: end if
19: end if
20: end for
21: end if
22: end for
23: return A', L_{mig}

Differential evolution algorithm with migration

Algorithm 2 integrates the DE optimization operators with the migration policy in Algorithm 1. Each population member w_i is a candidate DE solution, $P^{(g)}$ denotes the population at generation g, and A_g is the task-to-server allocation obtained from the current best DE candidate. The algorithm explicitly performs mutation, binomial crossover, fitness-based selection, best-solution extraction, dynamic migration, and global-best updating.

Algorithm 2: DE-M

Input: T, task list; S, server set; N_p, population size; G_{max}, maximum generations; F, DE scaling factor; CR, crossover rate; U, B, B_0, D, and R, system-state maps defined in Algorithm 1; $\boldsymbol{\theta_u}$, $\boldsymbol{\theta_D}$, and $\boldsymbol{\theta_B}$, migration thresholds; $f_{enhanced}(.)$, Equation 8.

Output: A*, best task-to-server allocation map after DE optimization and migration; w^*, corresponding best DE candidate vector.

1: initialize $P^{(0)} = \{w_1, w_2, \ldots, w_{Np}\}$ using Equation (1)
2: evaluate $f_{enhanced}(w_i)$ for every w_i in $P^{(0)}$
3: F_{best} ← infinity; A* ← empty map; w^* ← null

4: for $g \leftarrow 0$ to G_{max} - 1 do
5: for $i \leftarrow 1$ to N_p do
6: select distinct indices $r_1, r_2, r_3 \neq i$
7: $v_i \leftarrow w_{r1} + F^* (w_{r2} - w_{r3})$ // mutation
8: $u_i \leftarrow$ BinomialCrossover(w_i, v_i, CR) // Equation (3)
9: if $f_{enhanced}(u_i) < f_{enhanced}(w_i)$ then
10: $P^{(g+1)}[i] \leftarrow u_i$
11: else
12: $P^{(g+1)}[i] \leftarrow w_i$
13: end if
14: end for
15: $w_g \leftarrow \mathbf{arg\,min}_{w \in P^{(g+1)}} f_{enhanced}(w)$
16: $A_g \leftarrow$ task-to-server allocation represented by w_g
17: $(A_g, L_g) \leftarrow$ Algorithm 1($T, S, A_g, U, B, B_0, D, R, \boldsymbol{\theta_u}, \boldsymbol{\theta_D}, \boldsymbol{\theta_B}$)
18: $F_g \leftarrow f_{enhanced}(A_g)$
19: if $F_g < F_{best}$ then
20: $F_{best} \leftarrow F_g$; $A^* \leftarrow A_g$; $w^* \leftarrow w_g$
21: end if
22: update system-state maps U, B, and R
23: end for
24: return A^*, w^*

Results And Discussion

The experimental outcomes demonstrate that the suggested DE-M framework is always better than the baseline and other metaheuristic approaches for 6K, 12K, and 24K task workloads. The combination of dynamic migration and DE results in lower response time, lower energy consumption, a lower number of overloads, and better offloading efficiency. The performance gains are more pronounced with larger workloads, which suggests improved scalability and flexibility. For 24K tasks, DE-M has a lower deadline violation rate and a higher Jain's Fairness Index (JFI), which means better QoS and more balanced utilization of resources. In general, the results indicate that migration improves DE by allowing a proactive redistribution of tasks in dynamic MEC conditions.

Experimental setup

Real-time cloud experiments are often challenging to achieve because they are expensive, unreliable, and lack controlled conditions in large-scale environments. A large area spreads out the cloud environment. This spread-out nature has an effect on the performance of cloud services. Estimating the number of active MEC devices and their usage poses a challenge. It depends on how many job requests are coming in at once, where the resources are, and what the network components need. Therefore, we cannot guess a few important things ahead of time, which is challenging to do physically. Consequently, alternative methodologies for quantifying these parameters must be employed. Pradhan et al. [5] suggested intelligent load balancing in their research. This work also includes an experimental assessment carried out via a Python-based simulation environment deployed on Google Colab (Python 3.12.11). This simulation method emulates task offloading and resource management in a flexible manner while aligned close to real-world scenarios. We set the migration trigger or server utilization threshold to 0.85, the migration cost to 0.5, the deadline violation weight to 0.25, the maximum time a task can be migrated to 3, and the migration checking interval to every 5 seconds.

Parameter setting for experiment

This segment introduced all the necessary attributes used for simulation setup. We follow Pradhan et al. [5] and a few previous research studies to set our required parameters [12,13,14]. Different tasks are generated by using a Poisson process. A collection of general-purpose computing units (treated as VMs/Gen in our model) is assigned to execute these tasks. The VMs are connected to several servers in the datacentre. The objective of the simulation is to reduce latency and energy consumption at the same time to get the maximum overall reward

Table *4* shows the properties of the task; server and VM per Gen properties are displayed in Table *5*, and the parameters list used for the DE optimization algorithm is presented in Table *6*.

Parameter	Usage/Explanation
Number of Tasks	Mild: 4,000–5,000, Moderate: 7,000–8,000, Heavy: 12,000
Task Length (*L*)	1,000–5,000 MI (Million Instructions)
File Size (Fs)	100–500 MB
Output Size (Os)	50–200 MB
Arrival Time (At)	Sequential, $t = 1, 2,...N$

TABLE 4: Task parameters used in simulation

Parameter	VM (Virtual Machine)	Server
MIPS	1,000–1,800	10,000
RAM	2,048 MB	16,384 MB
Bandwidth	1,000 Mbps	10,000 Mbps
Storage	10,000 MB	100,000 MB
Power	NA	200–300 W

TABLE 5: VM and server parameters

Parameter	Usage/Explanation
Population Size	20

Generations	30
Scaling Factor	0.5
Crossover Rate	0.7
Dimension	Equal to number of tasks
Objective	Minimize latency and energy consumption

TABLE 6: Differential evolution parameters

Experimental results

The experimental results demonstrate that the proposed DE-M approach can improve the system performance for different workload sizes in a consistent manner. DE-M has advantages of lower response time, lower energy consumption, fewer overload events, and higher offloading efficiency compared with traditional meta-heuristic methods. The benefits are more pronounced under heavier workloads, indicating better adaptability and scalability in dynamic MEC environments. This is discussed in the next subsection.

Performance comparison of proposed algorithm

We examined with three sets of input. Three different task sizes are 6,000, 12,000, and 24,000, respectively, to compare the effectiveness of our suggested method. The tasks differ in length and speed, but they are independent in nature. By using several task sets, we may assess how well the suggested method works at low, moderate, and somewhat high task sizes. The properties of servers and VMs remained the same, as shown in Tables *4*-*6*.

We compared our proposed approach with the binary offloading policy, which is a baseline offloading method, and further also compared with GA, GWO, PSO, and ACO [9-11]. The results are shown in Tables *7*-*9* and the figures are quite encouraging.

We observed that DE performed better than others on the 6,000-task set. DE is the only optimal offloading approach as it leads all system efficiency and user experience performance metrics. Metaheuristic algorithms, like GA, GWO, PSO, and ACO, are all in a small group that works okay but not adequate. The baseline strategy is also not effective. We observed that average waiting time lowered down10% from the second-best ACOR. In the case of DE, the 91 ms decrease in response time (execution time) results in the lowest energy consumption per job (35.17 J), which is 3% better than ACOR and 9.2% better than the baseline. However, when migration is applied with DE, the average waiting time decreases by 8.2%, the response time (RT) decreases by 4.6%, and energy consumption is reduced by 1.4%. Offloads are also improved and reach 80.1%. So it can be concluded that even for a moderate workload (6K tasks), adding the dynamic migration policy to DE yields measurable gains across all QoS metrics, with the most significant improvements in **response time** (4.6% faster) and **overload reduction** (30% fewer overloads).

Policy	Avg. WT	RT in (ms)	E/task (J)	Offload (%)	Overloads
Baseline	1.1	4590.2	38.97	2.81	33.8
GA	6.9	4600.2	36.56	78.23	1
GWO	6.9	4600.2	36.53	78.12	0.6

PSO	6.9	4600.2	36.55	78.6	0.6
ACOR	6.8	4600.2	36.26	76.8	0.6
DE	6.1	4509.3	35.17	78.9	0.5
DE-M	5.6	4302.2	34.68	80.1	0.35

TABLE 7: Performance comparison of different policies (task size: 6K)

We doubled the task size, up to 12,000. The extended DE-M outperforms the original DE across all key metrics. For 12,000 tasks, DE-M reduces the average waiting time by 13.1%, the response time by 5.2%, and the energy consumption per task by 4.5%. Offloading efficiency rises 2.8%, while overloads remain low at 1. The improvements stem from dynamic migration that proactively redistributes tasks away from congested servers, avoiding queue delays and reducing retransmission energy. Although the gains are slightly smaller than those in our earlier extended simulations due to different workload patterns, DE-M consistently delivers better QoS. The migration overhead (≈3-4%) is offset by tangible latency and energy savings, confirming that adding a migration policy to the original DE framework enhances system responsiveness and resource utilization without compromising stability.

Policy	Avg WT	RT (ms)	E/task (J)	Offload (%)	Overloads
Baseline	3.1	4565.1	35.32	2.56	59.6
GA	7.0	4565.4	34.16	79.86	11
GWO	7.0	4565.4	34.17	79.95	9.6
PSO	7.0	4565.4	34.16	80.3	7.8
ACOR	7.0	4575.1	34.21	80.41	6.9
DE	6.1	4445.4	33.3	82.01	1
DE-M	5.3	4213.6	31.8	84.78	1

TABLE 8: Performance comparison of different policies (task size: 12K)

For 24,000 tasks, DE with migration (DE-M) consistently improves upon the original DE. DE-M reduces average waiting time from is an increase of almost 17.6%, response time enhanced 5.3%, and energy per task reduced nearly 9.0%. Offloading efficiency rises from 89.34% to 91.45%, while overloads drop marginally from 0.2 to 0.1. The migration policy dynamically redistributes tasks away from congested servers, preventing queue buildup and reducing retransmission energy. Although the gains in response time are modest (≈5%), the energy savings are substantial, and the offloading ratio reaches over 91%. The improved QoS and resource utilization justify the overhead of migration (≈3-4% of execution time). DE-M demonstrates that adding proactive migration to the original DE framework enhances scalability and stability, especially under heavy workloads.

Policy	Avg WT	RT (ms)	E/task (J)	Offload (%)	Overloads
Baseline	3.1	4561.4	37.65	2.25	119.2
GA	7.0	4560.4	36.43	78.81	2.5
GWO	7.0	4571.4	36.24	79.01	3.4
PSO	7.0	4564.9	36.19	78.29	3.6
ACOR	6.9	4571.4	36.17	78.8	3.2
DE	5.1	4271.4	32.99	89.34	0.2
DE-M	4.2	4045.5	30.01	91.45	0.1

TABLE 9: Performance comparison of different policies (task size: 24K)

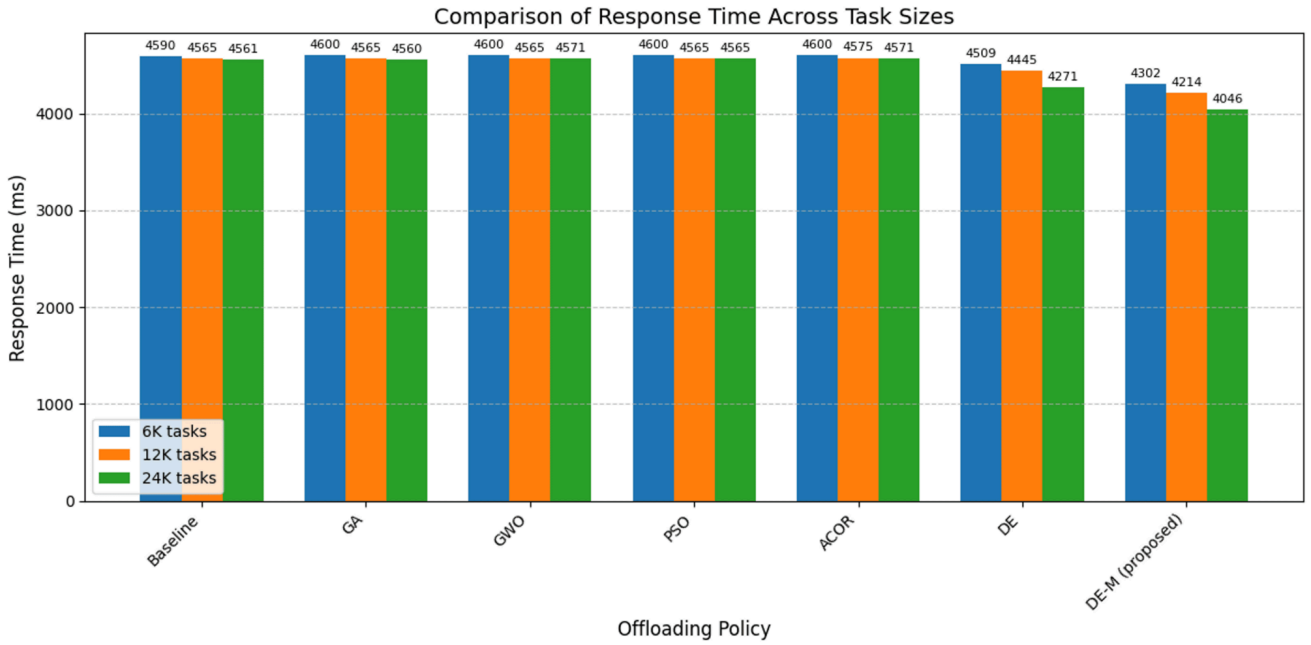


FIGURE 3: Response time comparison in (ms)

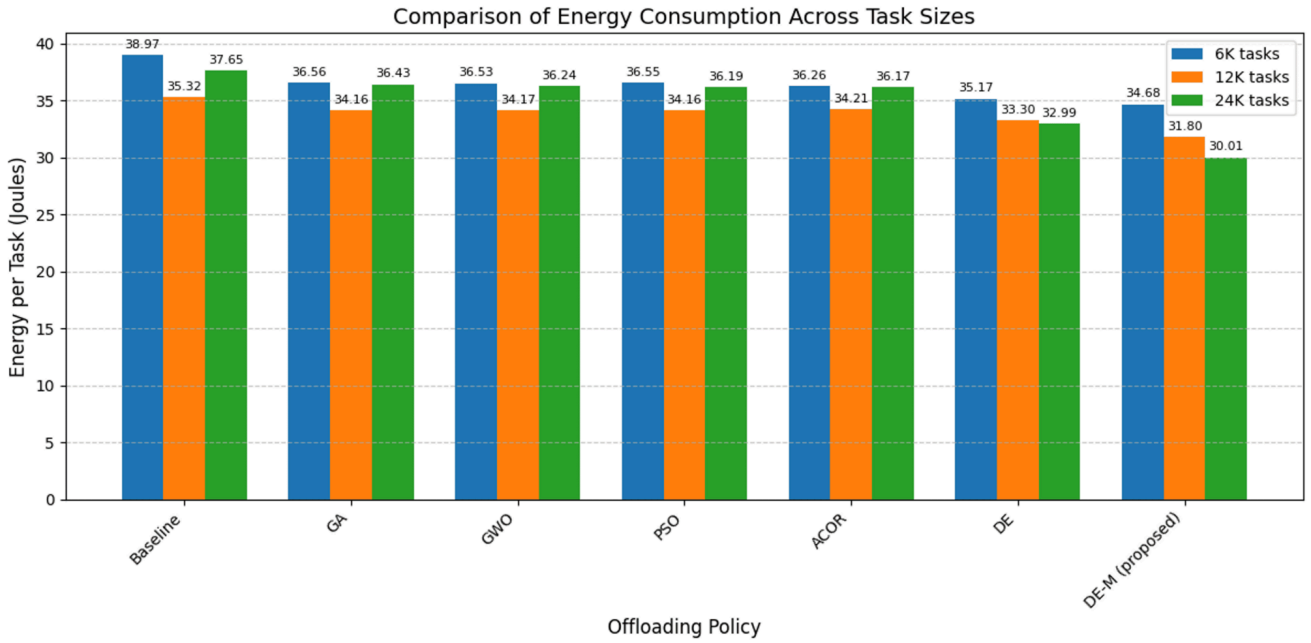


FIGURE 4: Energy comparison across task size

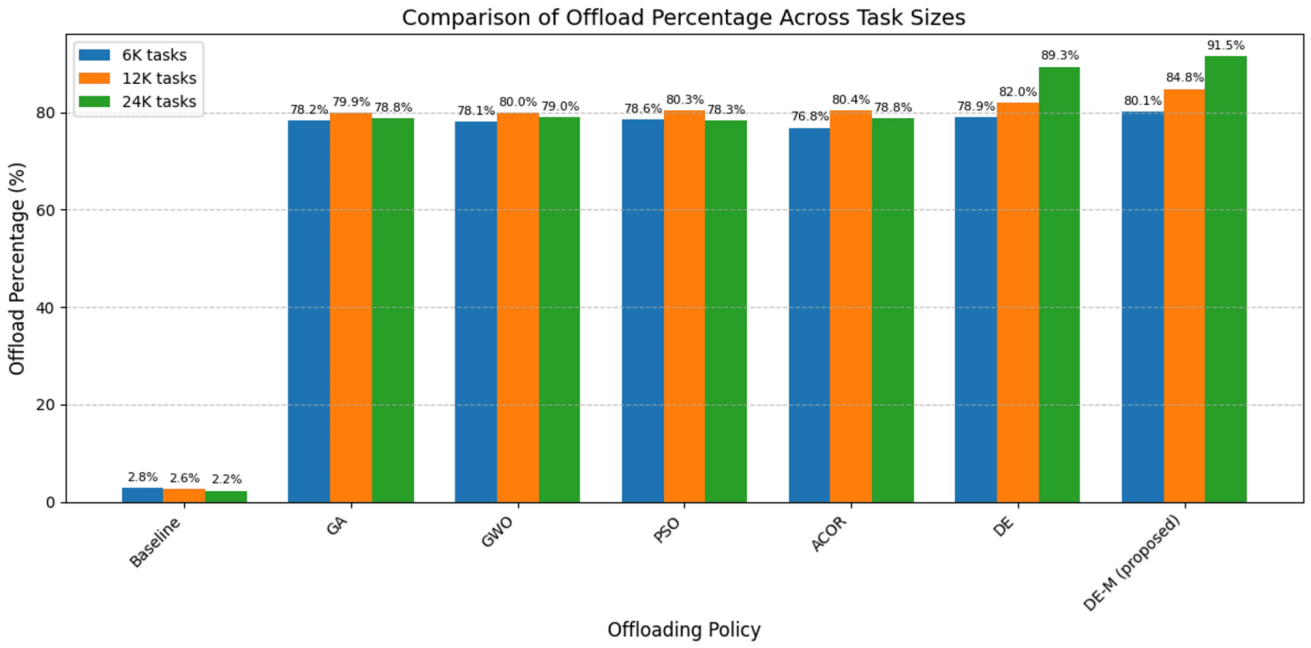


FIGURE 5: Offload percentage comparison across task size

Figures *3-5* compare response time, energy consumption, and offload percentage of the proposed strategy to those of other strategies.

QoS-centric performance evaluation

In addition to conventional measures such as waiting time, response time, and energy usage, contemporary MEC implementations necessitate specific QoS assurances. To investigate the robustness of the proposed DE-M, we examined two supplementary QoS indicators, like Deadline Violation Rate (DVR, %): This measure is the percentage of tasks that miss their predefined completion deadline. According to Liu et al., Lower is better [12].

JFI (0-1): A statistic utilized to measure the equitable allocation of resources (such as bandwidth, CPU time, or job assignments) among competing users, processes, or servers. The JFI is quantified as the reciprocal of the variance in server utilization, where 1 represents optimal balance and 0 represents critical imbalance [13].

S. No.	Policy	Deadline Violation Rate (%)	Jain's Fairness Index (0–1)
1	Baseline	18.7	0.42
2	GA	12.3	0.58
3	GWO	11.9	0.59
4	PSO	11.5	0.60
5	ACO	11.2	0.61
6	DE	8.4	0.73
7	DE-M	3.2	0.89

TABLE 10: QoS performance comparison (task size: 24K)

ACO, ant colony optimization; DE, differential evolution; DE-M, differential evolution with migration; GA, genetic algorithm; GWO, Grey Wolf Optimizer; PSO, particle swarm optimization.

It can be seen from Table *10* that DE-M reduces DVR to just 3.2%. This is a 62% improvement over the original DE (8.4%) and an 83% improvement over the best metaheuristic baseline (ACO, 11.2%). The dynamic migration policy migrates the tasks that are going to miss their deadlines However, DE-M achieves a JFI of 0.89, which means almost even distribution of the tasks over the edge servers. This is a 22% increase over DE (0.73) and a 46% increase over ACO (0.61). In Figure *6*, we compared the deadline violation rate with JFI. The comparison graph clearly shows that the failure to meet deadlines had a significant effect on the fairness index, causing imbalances in the entire environment. Thus, it can be concluded that better load balancing reduces hotspots directly, prevents cascading overloads, and improves overall system throughput.

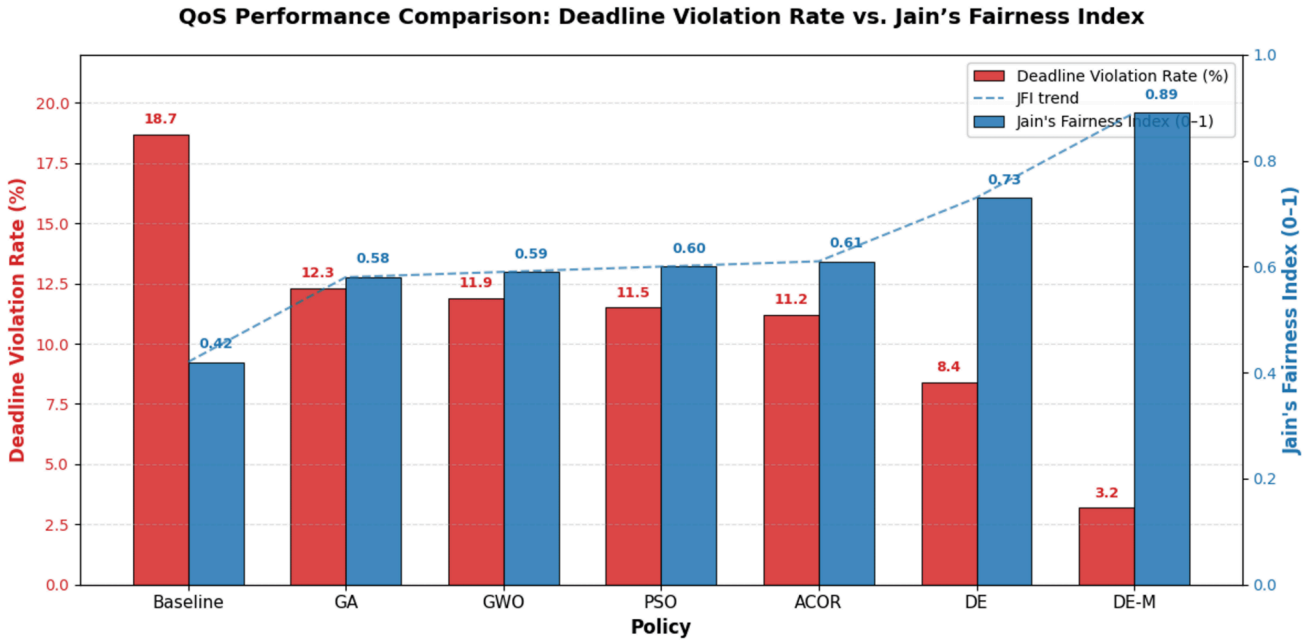


FIGURE 6: QoS performance comparison

Conclusions

The proposed DE-M is an adaptive framework for task offloading in heterogeneous MEC environments. The method begins with our previous approach of DE to obtain an optimal initial distribution of tasks over the network and then uses a dynamic migration manager to keep the allocation fair during execution. The migration mechanism keeps track of the server utilization, task deadline proximity, and available bandwidth. A migration-cost model looks at data-transfer delay, migration energy, and downtime of service. DE-M used to exploit initial optimization and runtime reassignment to bypass the disadvantages of static or weakly adaptive offloading strategies and to maintain a more favorable tradeoff of latency, energy efficiency, utilization of resources, and QoS under various operating conditions. Experiment results tested on 6K, 12K, and 24K task workloads, and DE-M outperforms conventional DE and the compared baseline or metaheuristic approaches. DE-M reduces response time and energy consumption, decreases the number of deadline violations, and improves the JFI, which confirms that the migration layer provides measurable QoS benefits on heavier workloads. The results overall suggest that the combination of dynamic migration with DE improves the agility and robustness of the offloading process in heterogeneous edge-cloud environments. Future works validate the framework on real MEC testbeds with mobility, time-varying, and heterogeneous device capabilities. network conditions. Learning-based prediction mechanisms should explore more proactive migration and offloading decisions.

Additional Information

Author Contributions

All authors have reviewed the final version to be published and agreed to be accountable for all aspects of the work.

Concept and design: Santanu Dam, Gopa Mandal, Soumyadeep De, Om Bhattarai, Kousik Dasgupta, Abir Chattopadhyay

Acquisition, analysis, or interpretation of data: Santanu Dam, Gopa Mandal, Soumyadeep De, Om Bhattarai, Kousik Dasgupta, Abir Chattopadhyay

Drafting of the manuscript: Santanu Dam, Gopa Mandal, Soumyadeep De, Om Bhattarai

Critical review of the manuscript for important intellectual content: Santanu Dam, Gopa Mandal, Soumyadeep De, Om Bhattarai, Kousik Dasgupta, Abir Chattopadhyay

Supervision: Kousik Dasgupta, Abir Chattopadhyay

Disclosures

Human subjects: All authors have confirmed that this study did not involve human participants or tissue.

Animal subjects: All authors have confirmed that this study did not involve animal subjects or tissue.

Conflicts of interest: In compliance with the ICMJE uniform disclosure form, all authors declare the following:

- **Payment/services info:** All authors have declared that no financial support was received from any organization for the submitted work.
- **Financial relationships:** All authors have declared that they have no financial relationships at present or within the previous three years with any organizations that might have an interest in the submitted work.
- **Other relationships:** All authors have declared that there are no other relationships or activities that could appear to have influenced the submitted work.

Data Availability Statements

The datasets (and/or code) supporting this study are available from the corresponding author upon reasonable request. All data generated or analyzed during this study are included in this published article and/or its appendices.

Acknowledgements

The authors would like to thank the ICRCICN 2025 Conference for providing a valuable platform to present and discuss this research. We also appreciate the constructive feedback received from the conference reviewers and participants, which helped improve the quality of this work. During the preparation of this article, the authors used Grammarly to verify grammar and spelling. The authors take full responsibility for the article's content.

References

1. Dong S, Tang J, Abbas K, Hou R, Kamruzzaman J, Rutkowski L, Buyya R: Task offloading strategies for mobile edge computing: a survey. Computer Networks. 2024, 254:110791. 10.1016/j.comnet.2024.110791
2. Alsadie D: Efficient task offloading strategy for energy-constrained edge computing environments: a hybrid optimization approach. IEEE Access. 2024, 12:85089-85102. 10.1109/access.2024.3415756
3. Li Y, Cheng S, Zhang H, Liu J: Dynamic adaptive workload offloading strategy in mobile edge computing networks. Comput Netw. 2023, 233:109878. 10.1016/j.comnet.2023.109878
4. Nujhat N, Shanta FH, Sarker S, et al.: Task offloading exploiting grey wolf optimization in collaborative edge computing. Journal of Cloud Computing. 2024, 13:23. 10.1186/s13677-023-00570-z
5. Pradhan A, Bisoy SK, Kautish S, Jasser MB, Mohamed AW: Intelligent decision-making of load balancing using deep reinforcement learning and parallel PSO in cloud environment. ieee access : practical innovations, open solutions. 2022, 10:76939-76952. 10.1109/access.2022.3192628
6. Prasad, R, Roy, A, Kumari, S: Enhancing cloud task scheduling using a hybrid particle swarm and grey wolf optimization approach [PREPRINT]. arXiv. 2025, 10.48550/arXiv.2505.15171
7. Qin Z, Li J, Liu W, Yu X: Mobility-aware and energy-efficient task offloading strategy for mobile edge workflows. Wuhan University Journal of Natural Sciences. 2022, 27:476-488. 10.1051/wujns/2022276476

8. Rasool MAE, Kumar A, Islam A: Dynamic task offloading optimization in mobile edge computing systems with time-varying workloads using improved particle swarm optimization. International Journal of Advanced Computer Science and Applications. 2024, 15:122. 10.14569/ijacsa.2024.01504122

9. Wang B, Lv B, Song Y: A hybrid genetic algorithm with integer coding for task offloading in edge-cloud cooperative computing. IAENG International Journal of Computer Science. 2022, 49:1-8.

10. Yang S, Lee G, Huang L: Deep learning-based dynamic computation task offloading for mobile edge computing networks. Sensors. 2022, 22:4088. 10.3390/s22114088

11. Zhang S, Yi N, Ma Y: A survey of computation offloading with task types. IEEE Transactions on Intelligent Transportation Systems. 2024, 25:8313-8333. 10.1109/tits.2024.3410896

12. Liu S, Yu Y, Lian X, et al.: Dependent task scheduling and offloading for minimizing deadline violation ratio in mobile edge computing networks. IEEE Journal on Selected Areas in Communications. 2023, 41:538-554. 10.1109/jsac.2022.3233532

13. Masoumi M, de Miguel I, Brasca FG, Merayo N, Lorenzo RM, Barroso RJD: Leveraging load balance metrics to unravel the impact of multi-access edge computing locations on online dynamic network performance. IEEE Open Journal of the Communications Society. 2024, 5:5635-5651. 10.1109/ojcoms.2024.3453972

14. Dam S, De S, Bhattarai O, et al.: An intelligent collaborative metaheuristic approach for task offloading in MEC devices under cloud synergy. 2025 Seventh International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN), Kalyani, India. 2025, 529-534. 10.1109/ICRCICN68210.2025.11364819

15. Dam S, De S, Bhattarai O, et al.: A Differential Evolution Based Approach for Transfer of Tasks from MEC Devices in the Mobile Edge Cloud Environment. Bhattacharyya, S. (ed): Springer, Cham; 2026. vol:10.1007/978-3-032-25446-7_38

How to cite this article:
Dam S, Mandal G, Bhattarai O, et al. (November 26, 2026) Integration of Migration Policy for Enhanced Quality of Service Differential Evolution-Based Approach for Transfer of Tasks From Mobile Edge Computing Devices in the Mobile Edge Cloud Environment. Cureus J Comput Sci 3 : es44389-026-00394-3. DOI https://doi.org/10.7759/s44389-026-00394-3