SvedbergOpen
DISSEMINATION OF KNOWLEDGE
International Journal of Artificial Intelligence and Machine Learning
Publisher's Home Page: https://www.svedbergopen.com/
Research Paper Open Access

Optimal Task Allocation Using Manta Ray Foraging And Genetic Algorithms In Heterogeneous Edge Environments

Neelima Pilli[1], Debasis Mohapatra[2], and Shiva Shankar Reddy[3]

[1]Department of Computer Science and Engineering, Biju Patnaik University of Technology, Rourkela, Odisha, INDIA
neelima.p47@gmail.com

[2]Department of Computer Science and Engineering, Parala Maharaja Engineering College, Berhampur, Odisha, INDIA
debasis.cse@pmec.ac.in

[3]Department of Computer Science and Engineering, Sagi Rama Krishnam Raju Engineering College (A), Bhimavaram, Andhra Pradesh, INDIA shiva.shankar591@gmail.com

Abstract
Edge computing has evolved as a solution for latency-sensitive IoT applications, yet efficient task allocation is challenging due to dynamic workloads, heterogeneous resources, and fluctuating network conditions. This paper proposes a novel integration of Manta Ray Foraging Optimization with Genetic Algorithm (MRFO-GA) for optimal task allocation. The fuzzy membership functions and rules prioritize tasks based on CPU utilization, storage utilization, bandwidth availability, network latency, and deadline constraints. We employ Markov Decision Process (MDP) and Lyapunov drift theory to achieve effective offloading decisions and load balancing, respectively. We compare the proposed MRFO-GA approach against four state-of-the-art approaches using five evaluation metrics: Energy (J), Latency (ms), Fitness Score, Packet Loss (%), and Reliability (%). The proposed MRFO-GA demonstrates superior performance across task loads of 50, 300, and 450.

Keywords Edge Computing, Fuzzy rules, Markov decision process, Lyapunov drift, Genetic Algorithm, Manta Ray Foraging Optimization.

1. Introduction

Task offloading is an important task in Mobile Edge Computing (MEC). Addressing computational challenges and optimal resource utilization is essential in the MEC environment [1]. Constraints related to priority, dynamic needs, dependencies, etc., have made the task allocation problem complex. Local information and decisions play a major role in task offloading in decentralized systems [2]. In this context, load balancing is required to distribute loads evenly across the available computational resources. Towards this end, Lyapunov optimization is widely used [3]. At the same time, deciding whether to offload a task to an edge/cloud server or execute it locally is essential for real-time response. Sometimes, hard decisions are ineffective. Fuzzy-based task characterization helps achieve efficient task-offloading decisions [4]. Machine learning and metaheuristic optimization are highly useful for optimizing resources. Computationally light tasks are preferred for execution on the edge server, whereas computationally intensive tasks are executed on the cloud server. Combining cloud and edge resources is required to meet task execution needs with enhanced efficiency, scalability, and speed [5-8]. Although several contributions have addressed the research issue, the problem remains open and requires optimal resource utilization and dynamic constraint satisfaction [9].
The proposed Manta Ray Foraging Optimization and Genetic Algorithm (MRFO-GA) framework combines several steps to address these issues. The hybridization of a fuzzy rule base, Lyapunov drift, and metaheuristic modeling has yielded a novel solution to these issues.

The vital contributions of this paper are:

- Fuzzifying the task parameters and leveraging a Markov Decision Process (MDP) for effective offloading decisions.
- Implementing Lyapunov drift to achieve load balancing.
- Implementation of a hybrid Manta Ray Foraging Optimization and Genetic Algorithm (MRFO-GA) framework to ensure effective task offloading.

The rest of this paper is organized as follows. Section 2 presents the literature review. Section 3 discusses the

proposed methodology. Section 4 presents the experimental results and analysis. Section 5 concludes the paper and outlines future research directions.

2. Literature Review

Enormous contributions have been made to efficient resource allocation and task offloading in MEC [10-20]. Among these contributions, Non-Orthogonal Multiple Access (NOMA) with Time Division Multiple Access (TDMA), Software Defined Network (SDN), reinforcement learning, machine learning, and blockchain technology are prevalent [10-14]. Applications of edge and cloud computing include satellite communication, UAVs, and vehicular networks. Table 1 discusses the methodologies, limitations, and strengths of several important contributions in the field.

Table 1: Review of a few prominent contributions

Paper	Methods Used	Limitations	Strengths
[21]	Distributed ML across Things, Edge, Cloud	App- specific; generalization needed	High latency & energy reduction; throughput gain
[22]	Federated Learning + scheduling	Simulation only	Enhanced offloading, optimization
[23]	Edge processing, local execution	Hardware- specific; limited scope	Major latency bandwidth and reduction
[24]	Queued task scheduling + base stations	Simulation- based	↑ Completion rate & ↓ delay
[25]	Energy-Aware vs. Edgeward Scheduling	No ML/AI integration	Good runtime performance and power savings
[26]	ANN-MLP on Edge vs Cloud	No latency metrics	Good accuracy and lower energy
[27]	DRLIS Scheduler	High model overhead	Response & load gains
[28]	DRL + task partitioning	Algorithmic complexity	High fairness, low delay
[29]	Scalable IoT building system	Focused on buildings	Scalability tested up to 1000 nodes
[30]	Lyapunov-based dynamic provisioning	Simulated	Strong latency/resource tradeoff
[31]	Stochastic+ Lyapunov	Limited to mobility use cases	Balanced energy & delay
[32]	Model benchmarking	Use-case- specific	Balanced comparison
[33]	Serverless platforms comparison (Kubeless, funcX)	ML-focused only	Evaluates under concurrency/task loads
[34]	Adaptive edge task offloading with service deployment; uses Approximate Deployment Graph (AD-graph) and task Scheduling based on service popularity	Focused on service popularity; may not generalize to all task types	Higher edge offloading rate; lower resource consumption in massive task scenarios
[35]	SDG-task offloading algorithm (DEFT) that duplicates key subtasks for Execution on multiple devices	Slight increase in energy consumption due to task duplication	Significantly reduces overall latency; maintains task dependencies

3. Proposed Methodology

Our suggested approach mainly emphasizes a fuzzy rule-based task queuing model and a task offloading/assignment strategy using MRFO-GA. The architecture in Figure 1 starts at the IoT Layer, where tasks are generated and submitted by IoT applications (Phase-I). Next, a Priority-Aware Fuzzy Rule-Based Model (Phase-II) classifies tasks as extremely urgent, urgent, or non-urgent based on latency, CPU utilization, and job deadlines. Next, the hybrid algorithm MRFO-GA (Phase-III) transfers the task to the edge layer, which

dynamically manages computing resources and optimizes load balancing between edge nodes based on Lyapunov drift. Finally, the system allocates computationally intensive tasks to the cloud.

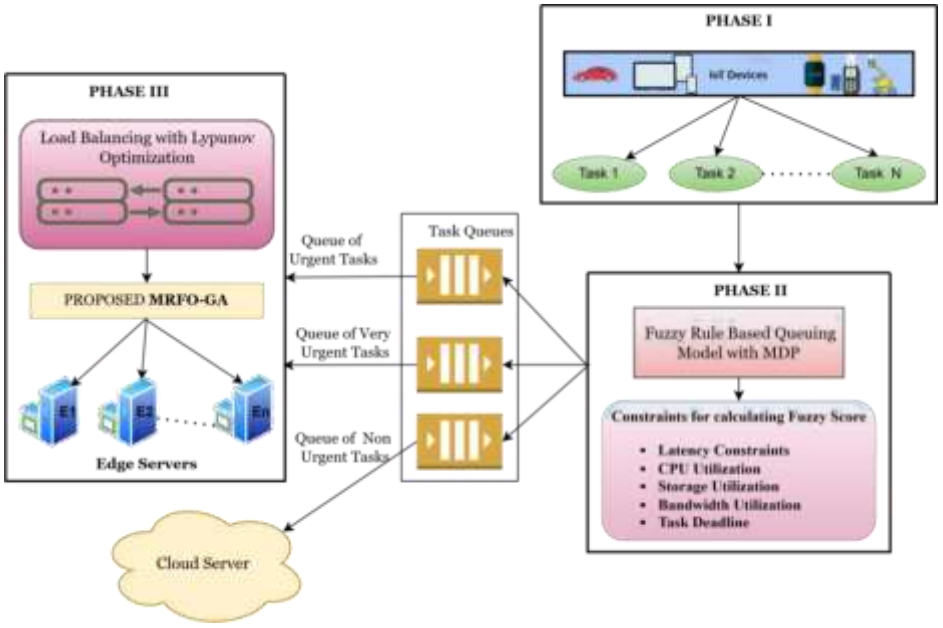


Figure 1: System Architecture

3.1 Fuzzy Rule-Based Queuing Model with Markov Decision Process

For offloading tasks, fuzzy logic determines whether to execute them locally on a device or offload them to a more powerful edge or cloud server. To balance the workload, fuzzy logic controls traffic and resource usage in cloud and edge networks. Fuzzy membership functions are designed based on factors such as traffic load, delay sensitivity, energy use, and link saturation. The proposed system fuzzifies five constraints: network latency, storage utilization, CPU usage, task deadline, and bandwidth usage. Memory utilization and processing power focus on CPU and storage utilization, whereas latency concerns and bandwidth focus on the speed and reliability of task communication. A fuzzy inference system makes intelligent, flexible scheduling decisions under uncertainty and ambiguity in real-time scenarios.

3.1.1 CPU Utilization Membership Functions

Equations 1, 2, and 3 define three membership functions corresponding to the linguistic terms Low, Medium, and High. These membership functions are defined as follows:

Low CPU Utilization:

$$\mu_{Low\_cpu}(x) = \begin{cases} 1 & x \leq 30, \\ \frac{70 - x}{40} & 30 < x < 70, \\ 0 & x \geq 70. \end{cases} \quad (1)$$

Medium CPU Utilization:

$$\mu_{Medium\_cpu}(x) = \begin{cases} 0 & x \leq 30, \\ \frac{x-30}{40} & 30 < x \leq 70, \\ \frac{100-x}{30} & 70 < x < 100, \\ 0 & x \geq 100. \end{cases} \quad (2)$$

High CPU Utilization:

$$\mu_{High\_cpu}(x) = \begin{cases} 0 & x \leq 70, \\ \frac{x-70}{30} & 70 < x < 100, \\ 1 & x \geq 100. \end{cases} \quad (3)$$

Where, x is the current CPU usage percentage (0–100%)

3.1.2 Task Deadline Membership Functions

The deadline is divided into tight, moderate, and relaxed states that specifies the three membership functions, which are defined by equations 4, 5 and 6 as follows:

$$\mu_{Tight\_deadline}(x) = \begin{cases} 1 & x \le 0.3, \\ \frac{0.7 - x}{0.4} & 0.3 < x < 0.7, \\ 0 & x \ge 0.7. \end{cases} \quad (4)$$

$$\mu_{Moderate\_deadline}(x) = \begin{cases} 0 & x \le 0.3, \\ \frac{x-0.3}{0.4} & 0.3 < x \le 0.7, \\ \frac{1-x}{0.3} & 0.7 < x < 1.0, \\ 0 & x \ge 1.0. \end{cases} \quad (5)$$

$$\mu_{Relaxed\_deadline}(x) = \begin{cases} 0 & x \le 0.7, \\ \frac{x-0.7}{0.3} & 0.7 < x \le 1.0, \\ 1 & x \ge 1.0. \end{cases} \quad (6)$$

Where x is the deadline time for a task in milliseconds.

3.1.3 Network Latency Membership Functions

The Network Latency is classified into three membership functions: low, medium, and high. The membership functions are defined by equations 7, 8, and 9 as follows:

$$\mu_{Low\_Latency}(x) = \begin{cases} 1 & x \le 0.3, \\ \frac{0.7 - x}{0.4} & 0.3 < x < 0.7, \\ 0 & x \ge 0.7. \end{cases} \quad (7)$$

$$\mu_{Medium\_Latency}(x) = \begin{cases} 0 & x \le 0.3, \\ \frac{x-0.3}{0.4} & 0.3 < x \le 0.7, \\ \frac{1-x}{0.3} & 0.7 < x < 1.0, \\ 0 & x \ge 1.0. \end{cases} \quad (8)$$

$$\mu_{High\_Latency}(x) = \begin{cases} 0 & x \le 0.7, \\ \frac{x-0.7}{0.3} & 0.7 < x < 1.0, \\ 1 & x \ge 1.0. \end{cases} \quad (9)$$

Where x is the measured network delay.

3.1.4 Storage Utilization Membership Functions

These membership functions are classified into three fuzzy sets: low, medium, and high. These functions are defined by equations 10, 11, and 12 as follows:

$$\mu_{Low\_StorageUtilization}(x) = \begin{cases} 1 & x \le 30, \\ \frac{70 - x}{40} & 30 < x < 70, \\ 0 & x \ge 70. \end{cases} \quad (10)$$

$$\mu_{Medium\_StorageUtilization}(x) = \begin{cases} 0 & x \le 30, \\ \frac{x-30}{40} & 30 < x \le 70, \\ \frac{100-x}{30} & 70 < x < 100, \\ 0 & x \ge 100. \end{cases} \quad (11)$$

$$\mu_{High\_StorageUtilization}(x) = \begin{cases} 0 & x \le 70, \\ \frac{x-70}{30} & 70 < x < 100, \\ 1 & x \ge 100. \end{cases} \quad (12)$$

Where x is the percentage of storage currently used.

3.1.5 Bandwidth Utilization Membership Functions

The Bandwidth utilization is classified into three membership functions: low, medium, and high. The membership functions are defined by equations 13, 14, and 15 as follows:

$$\mu_{LowBandwidth\_Utilization}(x) = \begin{cases} 1 & x \le 30, \\ \frac{70 - x}{40} & 30 < x < 70, \\ 0 & x \ge 70. \end{cases} \quad (13)$$

$$\mu_{MediumBandwidth\_Utilization}(x) = \begin{cases} 0 & x \le 30, \\ \frac{x-30}{40} & 30 < x \le 70, \\ \frac{100-x}{30} & 70 < x < 100, \\ 0 & x \ge 100. \end{cases} \quad (14)$$

$$\mu_{HighBandwidth\_Utilization}(x) = \begin{cases} 0 & x \le 70, \\ \frac{x-70}{30} & 70 < x < 100, \\ 1 & x \ge 100. \end{cases} \quad (15)$$

Where x is the percentage of available bandwidth being consumed.

3.2 Create Fuzzy Rules

Table 2 shows the selected fuzzy rules, where the qualities of CPU Utilization, Task Deadline, Network Latency, Storage Utilization, and Bandwidth are Low, Medium, and High, respectively. By presenting every potential value for every attribute as a Cartesian product, $3^5 = 243$ combinations are obtained. Every combination depicts a different set of work characteristics. Each rule maps a combination of fuzzified task parameters to an urgency level. Rules that have an aggregate weight score of at least 0.60 are kept. As a result, the final rule base is reduced to approximately 32 highly significant rules, which are then mapped to three urgency levels. This optimized fuzzy rule set forms the basis for task classification and subsequent Markov Decision Process (MDP) modeling.

Table 2: Selected Fuzzy rules

Rule No.	CPU Utility	Deadline	Latency	Storage	Bandwidth	Urgency Level
1	High	Tight	High	High	Low	Very Urgent
2	High	Tight	Medium	Medium	Medium	Very Urgent
3	High	Tight	Low	Low	High	Urgent
4	Medium	Tight	High	High	Low	Very Urgent
5	Medium	Tight	Medium	Medium	Medium	Urgent
6	Medium	Tight	Low	Low	High	Urgent
7	Low	Tight	High	High	Low	Urgent
8	Low	Tight	Medium	Medium	Medium	Urgent
9	Low	Tight	Low	Low	High	Non-Urgent
10	High	Moderate	High	High	Low	Very Urgent
11	High	Moderate	Medium	Medium	Medium	Urgent
12	High	Moderate	Low	Low	High	Non-Urgent
13	Medium	Moderate	High	High	Low	Urgent
14	Medium	Moderate	Medium	Medium	Medium	Urgent
15	Medium	Moderate	Low	Low	High	Non-Urgent
16	Low	Moderate	High	High	Low	Urgent
17	Low	Moderate	Medium	Medium	Medium	Non-Urgent
18	Low	Moderate	Low	Low	High	Non-Urgent
19	High	Relaxed	High	High	Low	Urgent
20	High	Relaxed	Medium	Medium	Medium	Non-Urgent
21	High	Relaxed	Low	Low	High	Non-Urgent
22	Medium	Relaxed	High	High	Low	Urgent
23	Medium	Relaxed	Medium	Medium	Medium	Non-Urgent
24	Medium	Relaxed	Low	Low	High	Non-Urgent
25	Low	Relaxed	High	High	Low	Non-Urgent
26	Low	Relaxed	Medium	Medium	Medium	Non-Urgent
27	Low	Relaxed	Low	Low	High	Non-Urgent
28	High	Tight	Low	Medium	Medium	Urgent
29	Medium	Tight	Low	Low	High	Non-Urgent
30	Low	Tight	Low	High	Low	Non-Urgent
31	High	Moderate	Low	Medium	Medium	Urgent
32	Medium	Moderate	Low	Low	High	Non-Urgent

From the fuzzy rule evaluation, the system computes a Fuzzy Strength Score (0 to 1). According to this score, the system classifies the task into one of three urgency states (S0, S1, and S2). The MDP framework uses these states to track system behavior, as shown in Figure 2. The States are mapped according to the fuzzy strength score as follows.

- S0 (Non-Urgent) if score ≤ 0.40
- S1 (Urgent) if 0.40 < score ≤ 0.60
- S2 (Very Urgent) if score > 0.60

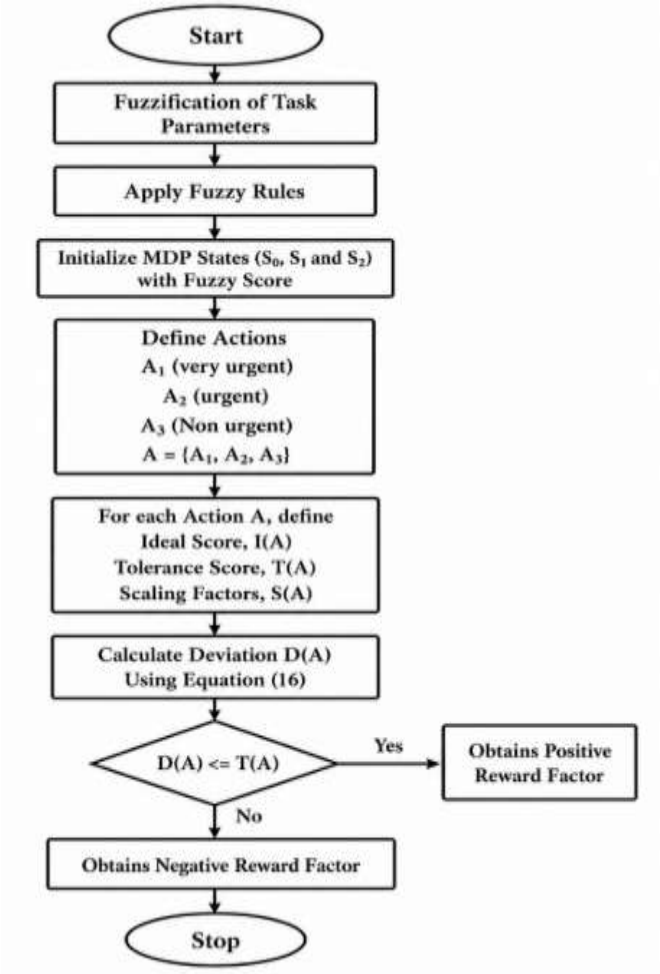


Figure 2. Flow of the MDP model

Based on the state, the system selects a set of possible actions: A1, which offloads the task immediately; A2, which offloads if Possible; and A3, which Processes Locally. These actions reflect different task execution strategies. Each action has a target fuzzy score called the Ideal Score (I(A)). A Tolerance Score (T(A)) defines the range within which deviation is acceptable. To improve decision quality, we use a Scaling Factor (S(A)) for each action A, and the deviation D(A) is given by equation 16 below.

$$D(A) = |fuzzy\ score - I(A)| \times S(A) \quad (16)$$

The deviation D(A) measures the affinity of the fuzzy score to the ideal score. If the D(A) is within the tolerance T(A) then action A obtain a reward of +1 × BaseRewardFactor. Otherwise it obtains a negative reward of -1 × BaseRewardFactor. The system then uses this reward to update future decisions. Finally, the system intelligently selects actions for each task based on urgency and system state.

3.3 Algorithm

Input: Task parameters (Delay, Energy, Size), Fuzzy Rule Base, Ideal Scores, Tolerance, Scaling Factors.
Output: Selected Action (Offload Immediately / Offload If Possible / Process Locally), Reward Score.

Step 1: Input Collection
Receive different input parameters for each incoming task. The parameters we considered are CPU Utilization, Network Latency, Task deadline, Storage Utilization, and Bandwidth.

Step 2: Fuzzification
Convert the crisp input values into fuzzy labels using predefined membership functions.

e.g., CPU Utilization → {Low, Medium, High}, Bandwidth → {Low, High}, etc.

Step 3: Fuzzy Inference and Score Calculation
Use the fuzzy rule base to infer an overall fuzzy score (0 to 1) that represents the urgency of the task.

Step 4: MDP State Initialization
Based on the fuzzy score, MDP states are initialized as follows:

- If score ≤ 0.40 → State S0: Non-Urgent
- If 0.40 < score ≤ 0.60 → State S1: Urgent
- If score > 0.60 → State S2: Very Urgent

Step 5: Define Candidate Actions and Action Parameters
Set possible actions A = {A1, A2, A3}:

- A1 = Offload Immediately
- A2 = Offload If Possible
- A3 = Process Locally

For each action A, we define the parameters as Ideal Score I(A), Tolerance T(A), Scaling Factor S(A), and BaseRewardFactor based on the urgency level.

Step 6: Deviation Computation
For each action A, we calculate the Compute Deviation D(A) by equation 16.

Step 7: Tolerance Check, Action Validation, and Selection
For each action:

If D(A) ≤ T(A): Mark as valid
Else: Mark as invalid or penalized

From valid actions, select the one with the lowest deviation. If multiple, use a tie-breaking rule (e.g., select the action with the lower energy cost).

Step 8: Execute Action and Reward Assignment
Perform the selected action (offload or local execution).
After task execution:

If successful → Reward R(A) = +1 × BaseRewardFactor(A)
If failed → Reward R(A) = -1 × BaseRewardFactor(A)

Step 9: Policy Update
Using the reward information, update the decision policy. The algorithm improves offloading decisions over time by learning to link states with actions that produce the highest predicted rewards.

3.4 MRFO-GA
This section presents a novel combination of the Genetic Algorithm (GA) and Manta Ray Foraging Optimization (MRFO) to achieve effective task offloading. Figure 3 shows the detailed architecture. The GA generates tasks with an initial server mapping. The optimal allocation takes place through the phases of selection, crossover, and mutation. The resulting population is used as input to Manta Ray Foraging Optimization (MRFO), which applies chain, cyclone, and somersault foraging strategies to generate a new population. If the desired fitness score is achieved, then the best solution is considered. Otherwise, the loop continues until convergence is satisfied (no more improvement is possible).

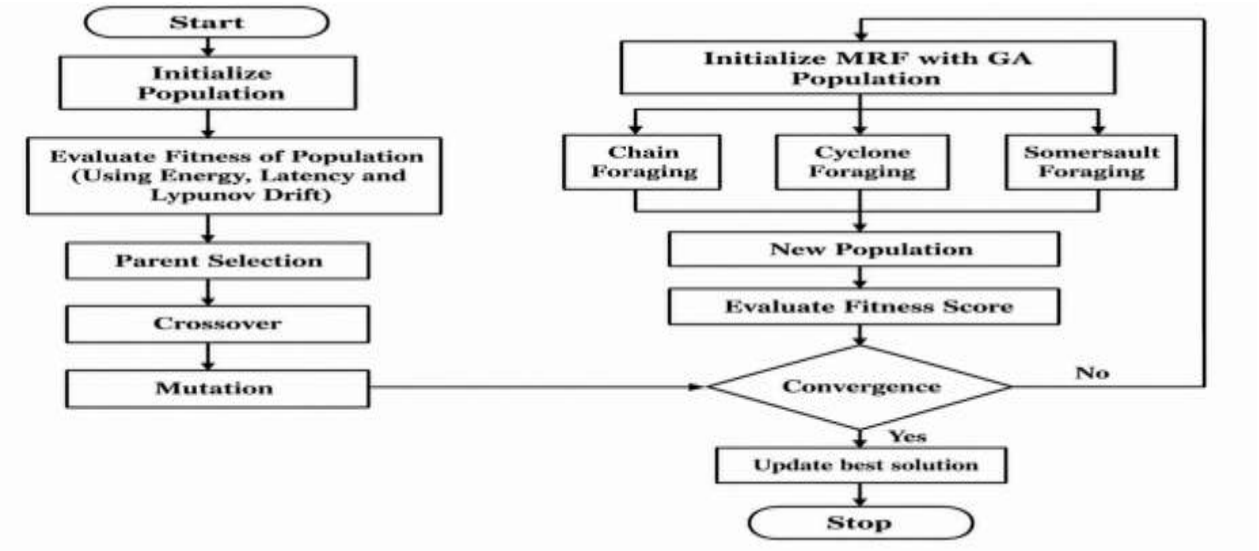


Figure 3: Architecture of the MRFO-GA algorithm

3.4.1 Chain foraging strategy

This strategy allows individuals to follow the best solution found so far. It helps exploit known good areas in the search space, as given by equation 17.

$$x_k^i(t+1)=\begin{cases} x_k^i(t)+r\left(x_k^{best}(t)-x_k^i(t)\right)+\alpha\left(x_k^{best}(t)-x_k^i(t)\right), & i=1 \\ x_k^i(t)+r\left(x_k^{i-1}(t)-x_k^i(t)\right)+\alpha\left(x_k^{best}(t)-x_k^i(t)\right), & i>1 \end{cases} \quad (17)$$

Where,

- $x_k^i(t)$): position of the i^{th} manta ray in the k^{th} dimension at iteration t.
- r: Random number between 0 and 1.
- α: Weight coefficient.
- $x_k^{best}(t)$: Position of the best solution so far.

3.4.2 Cyclone foraging strategy

Cyclone foraging simulates the spiral-like movement of manta rays. It balances exploration and exploitation, gradually reducing the exploration as the search converges, as depicted in equation 18.

$$x_k^i(t+1)=\begin{cases} x_k^{best}(t)+r\left(x_k^{best}(t)-x_k^i(t)\right)+\beta\left(x_k^{best}(t)-x_k^i(t)\right), & i=1 \\ x_k^{best}(t)+r\left(x_k^{i-1}(t)-x_k^i(t)\right)+\beta\left(x_k^{best}(t)-x_k^i(t)\right), & i>1 \end{cases} \quad (18)$$

Where

- T: Total number of iterations.
- β: Spiral weight.
- r: Random number between 0 and 1

3.4.3 Somersault Foraging Strategy

Somersault behavior lets the algorithm perform global jumps in the search space. This is critical for escaping local optima and improving solution diversity, as shown in equation 19.

$$x_k^i(t+1)=x_k^i(t)+S\times\left(r_1\times x_k^{best}(t)-x_k^i(t)\right)-r_2\times\left(x_k^i(t)\right) \quad (19)$$

Where

- S: Somersault factor.
- r1, r2: Random numbers between 0 and 1.

Next, we present the full flow of the algorithmic steps.

Step 1: Randomly initialize the GA Population to allocate the tasks to available servers.

Step 2: Evaluate the fitness score of each individual (i.e., one allocation of all tasks) in the population using a combined metric based on Energy consumption, latency, and Lyapunov drift.

Step 3: Use the fitness score to evolve better solutions via the GA process

- **Parent Selection:**
  Use selection techniques (e.g., tournament selection or roulette-wheel selection) to select parents for reproduction based on their fitness.
- **Crossover:**
  Combine selected parents to produce offspring using a crossover mechanism (e.g., single-point or uniform crossover).
- **Mutation:**
  Apply a mutation operator to introduce variations in offspring solutions and explore diverse solutions.
- **Generate Offspring Population:**

Replace the existing population with the newly generated offspring.

Step 4: Initialize the MRFO model using the improved population generated by GA.

- **Chain Foraging:** Move each individual toward the best neighbor to improve exploitation of good solutions.
- **Cyclone Foraging**: Apply spiral-like movement to explore the global space for better solutions.
- **Somersault Foraging**: Introduce random large jumps in the solution space to escape local optima.
- **New Population Update:** Update the population based on the foraging movements.

Step 5: For each updated MRFO solution, re-evaluate fitness using the same formula from Step 2.

Step 6: Convergence Check
Assess if the algorithm has met the stopping criterion (e.g., minimal change in fitness values, maximum iterations reached, or an optimal solution found).

Step 7: Update Best Solution
If a better solution is found in the current population then update the best solution.

Step 8: Repeat or Stop
If convergence is achieved or the maximum number of iterations is reached then terminate the algorithm. Otherwise, repeat from Step 2.

4. Results and Discussion

The MRFO-GA (Manta Ray Foraging Optimization with Genetic Algorithm) approach represents an adaptive stage of the proposed hierarchical offloading system. The dataset used here is a synthetic, random IoT task dataset designed to simulate realistic edge computing environments. It consists of a varying number of tasks, typically ranging from 50 to 450. Each task is characterized by a set of parameters that influence both energy consumption and execution latency, which are critical for evaluating the offloading strategy. Table 3 shows each hyperparameter used in the proposed approach.

Table 3: Details of hyperparameters

Parameter	Value
Population size	10
Generation	20
MRFA Generations	10
Crossover Rate	0.8
Mutation Rate	0.1
Movement Strength (α)	0.1 (used in cyclone foraging)
Foraging Strength (β)	1.0 (used in chain foraging)
Diffusion Strength (γ)	0.1 (used in somersault foraging)

4.1 Performance Metrics

This section provides a detailed evaluation of the performance metrics used to compare MRFO-GA with existing approaches. Here, we consider five metrics: Energy Consumption, Average Latency, Fitness Score, Packet Loss Ratio, and Reliability.

4.1.1 Energy Consumption (EC)

The total energy consumed by all edge servers for computing offloaded tasks is given by equation 20.

$$EC=\sum_{i=1}^{M}\left(E_{CPU}^{i}+E_{storage}^{i}\right) \quad (20)$$

Where E_{CPU}^{i} is CPU Energy Consumption for task i, and $E_{storage}^{i}$ is Storage Energy Consumption for task i.

4.1.2 Average Latency (AL)

Average latency measures the total time taken to complete all tasks, including transmission, queueing, and processing delays, as depicted in equation 21.

$$AL=\frac{1}{M}\sum_{i=1}^{M}\left(L_i^{network}+L_i^{queue}+L_i^{execution}\right) \quad (21)$$

Where M is the total number of tasks, $L_i^{network}$ is the time to transmit task i, L_i^{queue} is the waiting time in the server queue for task i, and $L_i^{execution}$ is the time to execute task i.

4.1.3 Fitness Score

The fitness score is a metric used to evaluate the quality of a task offloading decision. It considers task priority, energy consumption, latency, deadline violations, available server capacity, and Lyapunov drift. A higher fitness score indicates a better offloading decision. The fitness score is given in equation 22.

$$Fitness_{i,j} = w_1 \frac{P_i}{E_i+L_i+\delta_i} + w_2 C_j^{avail} - w_3 V.\Delta L(t) \quad (22)$$

Where $w_1 + w_2 + w_3 = 1$, $Fitness_{i,j}$ is the fitness score of assigning task i to server j, P_i is the Priority of task i, E_i is the Energy consumption of task i (normalized), L_i is the latency of task i (normalized), δ_i is the penalty for violating the deadline (normalized), C_j^{avail} is the available capacity on server j (normalized), V is the Lyapunov control weight (a tuning parameter), $\Delta L(t)$ is the Lyapunov drift at time t, and is defined by equation 23 as below.

$$\Delta L(t)=L(t+1)-L(t)=\frac{1}{2}\sum_{j=1}^{N}\left(Q_j(t+1)^2-Q_j(t)^2\right) \quad (23)$$

Where L(t)is the Lyapunov function representing total system congestion at time t, and it is represented by equation 24.

$$L(t)=\frac{1}{2}\sum_{j=1}^{N}\left(Q_j(t)\right)^2 \quad (24)$$

Where $Q_j(t)$ is the Queue length or current load on server j at time t, and $Q_j(t+1)$ is the updated load after assigning the new task.

4.1.4 Packet Loss Ratio (PLR)

PLR in equation 25 quantifies the packet loss during the transmission.

$$PLR(\%) = \frac{P_{lost}}{P_{sent}} \times 100 \quad (25)$$

Where, P_{lost} is the number of packets lost during transmission, and P_{sent} is the total number of packets sent from IoT devices.

4.1.5 Reliability

Equation 26 measures system robustness and task handling reliability. A value close to 1 indicates high reliability.

$$R\ (\%) = \left(1-\frac{T_{failed}}{T_{total}}\right) \times 100 \quad (26)$$

Where T_{failed} is the number of failed or rejected tasks and T_{total} is the total number of tasks.

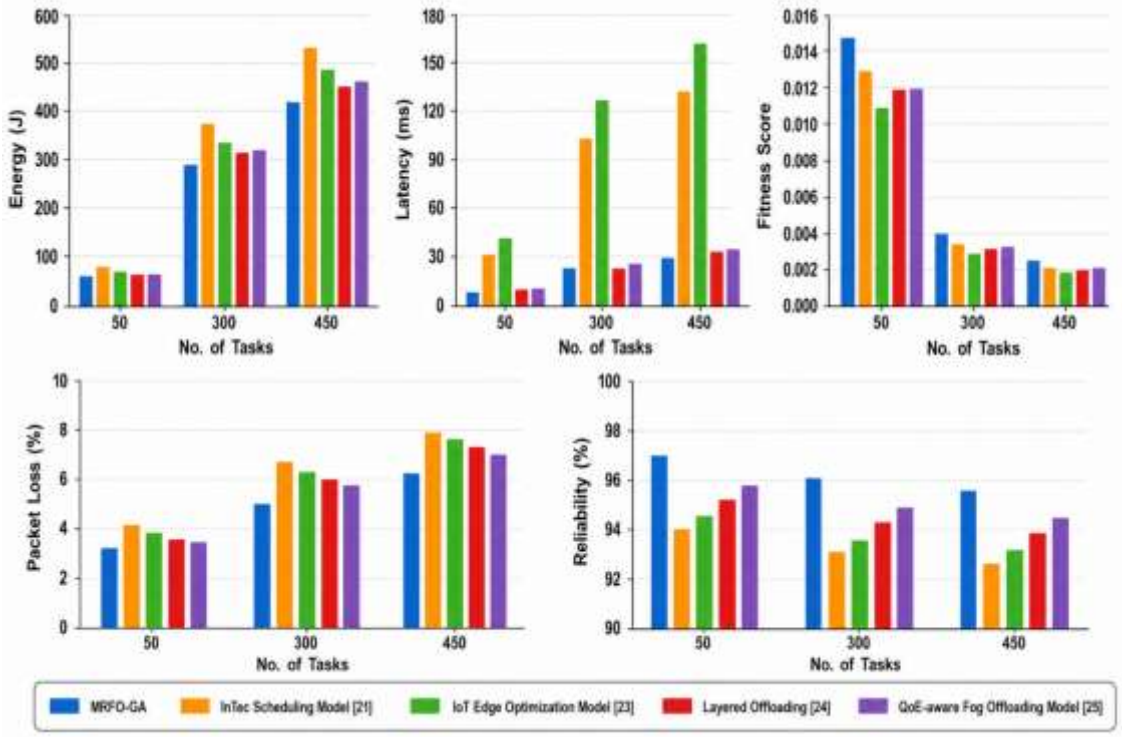
Table 4 compares the proposed MRFO-GA approach with existing approaches using Average Energy Consumption, Average Latency, Fitness Score, Packet Loss Ratio (PLR), and Reliability under task loads of 50,

300, and 450. Figure 4 illustrates the performance trends across workloads ranging from 50 to 450 tasks. The results show that MRFO-GA consistently achieves the lowest energy consumption and latency, demonstrating efficient resource allocation and task scheduling. In contrast, the InTec model records the highest energy usage and delay, while IoT Edge Optimization, Task Offload, and QoE Fog exhibit moderate performance. Additionally, MRFO-GA attains the highest fitness scores, indicating superior task allocation decisions. Packet loss remains below 5%, and reliability exceeds 95% across all workloads, outperforming the compared methods. Overall, the proposed MRFO-GA framework strikes an effective balance among energy efficiency, responsiveness, reliability, and scalability in IoT-enabled edge computing environments.

Table 4: Comparison of approaches for different Tasks

Models	Energy (J)			Latency (ms)			Fitness Score			Packet Loss (%)			Reliability (%)		
	No. of Tasks			No. of Tasks			No. of Tasks			No. of Tasks			No. of Tasks		
	50	300	450	50	300	450	50	300	450	50	300	450	50	300	450
MRFO-GA	60	290	420	6	20	26	0.015	0.004	0.0025	3.2	5.0	6.2	97	96.1	95.6
InTec Scheduling Model [21]	76.8	371.2	537.6	30.6	102	132.6	0.0128	0.0034	0.0021	4.1	6.7	7.9	94	93.1	92.6
IoT Edge Optimization Model [23]	69	333.5	483	37.5	125	162.5	0.0113	0.0030	0.0019	3.8	6.3	7.6	94.5	93.6	93.1
Layered Offloading [24]	64.8	313.2	453.6	7.5	25	32.5	0.0120	0.0032	0.0020	3.6	6.0	7.3	95.2	94.3	93.8
QoE-aware Fog Offloading Model [25]	66	319	462	7.8	26	33.8	0.0123	0.0033	0.0021	3.5	5.8	7.0	95.8	94.9	94.4

Figure 4: Comparison among the algorithms



5. Conclusion and Future Work

This paper presents a hybrid approach combining techniques like fuzzy rule-based systems, Markov Decision Process (MDP), Lyapunov drift, metaheuristics like Manta Ray Foraging Optimization, and Genetic Algorithm to ensure effective task offloading and resource allocation in edge/cloud environments. The proposed MRFO-GA approach is compared with four state-of-the-art approaches using the evaluation parameters Energy (J), Latency

(ms), Fitness Score, Packet Loss (%), and Reliability (%). The proposed MRFO-GA outperforms the four state-of-the-art approaches. Future researchers may investigate alternative methods to improve the proposed framework. Future research can include security-aware task scheduling, privacy-preserving optimization, and critical-task prioritization policies.

Declarations

Conflict of interest The authors have no conflict of interest**.**

Funding No funds have been received for this work.

References

1. Gao J, Chang R, Yang Z, Huang Q, Zhao Y, Wu Y. A task offloading algorithm for cloud-edge collaborative system based on Lyapunov optimization. Cluster Computing. 2023 Feb;26(1):337-48.
2. Jia Y, Zhang C, Huang Y, Zhang W. Lyapunov optimization based mobile edge computing for Internet of Vehicles systems. IEEE Transactions on Communications. 2022 Sep 19;70(11):7418- 33.
3. Yang G, Hou L, He X, He D, Chan S, Guizani M. Offloading time optimization via Markov decision process in mobile-edge computing. IEEE internet of things journal. 2020 Oct 23;8(4):2483-93.
4. Men R, Fan X, Shan A, Yuan G. Fuzzy Logic Based Binary Computation Offloading Scheme in V2X Communication Networks. IEEE Access. 2024 Mar 18.
5. Pilli, Neelima, Debasis Mohapatra, and Shiva Shankar Reddy. "Review on Meta-heuristic Algorithm-Based Priority-Aware Computation Offloading in Edge Computing System." Journal of The Institution of Engineers (India): Series B (2025): 1-26.
6. Zhang, Siqi, Na Yi, and Yi Ma. "A survey of computation offloading with task types." IEEE Transactions on Intelligent Transportation Systems (2024).
7. Abdulazeez DH, Askar SK. A Novel Offloading Mechanism Leveraging Fuzzy Logic and Deep Reinforcement Learning to Improve IoT Application Performance in a Three-Layer Architecture Within the Fog-Cloud Environment. IEEE Access. 2024 Mar 13.
8. Li N, Zhai L, Ma Z, Zhu X, Li Y. Lyapunov-guided Deep Reinforcement Learning for service caching and task offloading in Mobile Edge Computing. Computer Networks. 2024 Jun 10:110593.
9. Shahidinejad A, Ghobaei-Arani M. A metaheuristic-based computation offloading in edge-cloud environment. Journal of Ambient Intelligence and Humanized Computing. 2022 May;13(5):2785- 94.
10. Dulout, Romain, Leo Mendiboure, Yannis Pousset, Virginie Deniau, and Frédéric Launay. "Non- orthogonal multiple access for offloading in multi-access edge computing: A survey." IEEE Access 11 (2023): 118983-119016.
11. Al-Hammadi I, Li M, Islam SM, Al-Mosharea E. Collaborative computation offloading for scheduling emergency tasks in SDN-based mobile edge computing networks. Computer Networks. 2024 Jan 1;238:110101.
12. Wu X, Dong S, Hu J, Huang Z. An efficient many-objective optimization algorithm for computation offloading in heterogeneous vehicular edge computing network. Simulation Modelling Practice and Theory. 2024 Feb 1;131:102870.
13. Bi J, Wang Z, Yuan H, Zhang J, Zhou M. Cost-Minimized computation offloading and user association in hybrid cloud and edge computing. IEEE Internet of Things Journal. 2024 Jan 16.
14. Lin N, Bai L, Hawbani A, Guan Y, Mao C, Liu Z, Zhao L. Deep reinforcement learning-based computation offloading for servicing dynamic demand in multi-UAV-assisted IoT network. IEEE Internet of Things Journal. 2024 Jan 22.
15. Zhou J, Yang Q, Zhao L, Dai H, Xiao F. Mobility-aware computation offloading in satellite edge computing networks. IEEE Transactions on Mobile Computing. 2024 Jan 29.
16. Chen, Ying, et al. "Qos-aware computation offloading in leo satellite edge computing for iot: A game-theoretical approach." Chinese Journal of Electronics 33.4 (2024): 875-885.
17. Guo, Yinzhi, Xiaolong Xu, and Fu Xiao. "MADRLOM: A Computation offloading mechanism for software-defined cloud-edge computing power network." Computer Networks 245 (2024): 110352.
18. Gao, Xiangqiang, et al. "Hierarchical Dynamic Resource Allocation for Computation Offloading in LEO Satellite Networks." IEEE Internet of Things Journal (2024).
19. Zhang, Siqi, Na Yi, and Yi Ma. "A survey of computation offloading with task types." IEEE Transactions on Intelligent Transportation Systems (2024).
20. Lin, Na, et al. "Deep reinforcement learning-based computation offloading for servicing dynamic demand in multi-UAV-assisted IoT network." IEEE Internet of Things Journal (2024).

21. Larian, Habib, and Faramarz Safi-Esfahani. "InTec: integrated things-edge computing: a framework for distributing machine learning pipelines in edge AI systems." Computing 107, no. 1 (2025): 41.
22. Shah, Syed Sarmad, and Anas Ali. "Optimizing Resource Allocation and Energy Efficiency in Federated Fog Computing for IoT." arXiv preprint arXiv:2504.00791 (2025).
23. That, Vitou, Kimchheang Chhea, and Jung-Ryun Lee. "Optimizing Energy Consumption and Latency in IoT through Edge Computing in Air-Ground Integrated Network with Deep Reinforcement Learning." IEEE Open Journal of Vehicular Technology (2024).
24. Birhanie, Habtamu Mohammed, and Mohammed Oumer Adem. "Optimized task offloading strategy in IoT edge computing network." Journal of King Saud University-Computer and Information Sciences 36, no. 2 (2024): 101942.
25. Keat, Low Choon, Ng Yen Phing, and Tan Xuan Ying. "Optimizing QoE and Energy Consumption for IoT Applications in Fog Computing." International Journal of Research and Innovation in Social Science 9, no. 2 (2025): 1997-2059.
26. Gamazo-Real, Jose-Carlos, Raúl Torres Fernández, and Adrián Murillo Armas. "Comparison of edge computing methods in Internet of Things architectures for efficient estimation of indoor environmental parameters with Machine Learning." Engineering Applications of Artificial Intelligence 126 (2023): 107149.
27. Wang, Zhiyu, Mohammad Goudarzi, Mingming Gong, and Rajkumar Buyya. "Deep reinforcement learning-based scheduling for optimizing system load and response time in edge and fog computing environments." Future Generation Computer Systems 152 (2024): 55-69.
28. Chen, Yishan, Xiansong Luo, Peng Liang, Junxiao Han, and Zhonghui Xu. "Priority-based dag task offloading and secondary resource allocation in iot edge computing environments." Computing 106, no. 10 (2024): 3229-3254.
29. Lăcătuşu, Florin, Anca Daniela Ionita, Marian Lăcătuşu, and Adriana Olteanu. "Performance Evaluation of Information Gathering from Edge Devices in a Complex of Smart Buildings." Sensors 22, no. 3 (2022): 1002.
30. Abouaomar, Amine, Soumaya Cherkaoui, Zoubeir Mlika, and Abdellatif Kobbane. "Resource provisioning in edge computing for latency-sensitive applications." IEEE Internet of Things Journal 8, no. 14 (2021): 11088-11099.
31. Zhang, Yanfang, Jian Chen, Yuchen Zhou, Long Yang, Bingtao He, and Yijin Yang. "Dependent task offloading with energy-latency tradeoff in mobile edge computing." IET Communications 16, no. 17 (2022): 1993-2001.
32. Alqahtani, Daghash K., Muhammad Aamir Cheema, and Adel N. Toosi. "Benchmarking deep learning models for object detection on edge computing devices." In International Conference on Service-Oriented Computing, pp. 142-150. Singapore: Springer Nature Singapore, 2024.
33. Trieu, Quoc Lap, Bahman Javadi, Jim Basilakis, and Adel N. Toosi. "Performance evaluation of serverless edge computing for machine learning applications." In 2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC), pp. 139-144. IEEE, 2022.
34. Song, Liangjun, Gang Sun, Hongfang Yu, and Mohsen Guizani. "SD-AETO: Service-deployment- enabled adaptive edge task offloading scheme in MEC." IEEE Internet of Things Journal 10, no. 21 (2023): 19296-19311.
35. Kjorveziroski, Vojdan, and Sonja Filiposka. "Kubernetes distributions for the edge: serverless performance evaluation." The Journal of Supercomputing 78, no. 11 (2022): 13728-13755.