International Journal of Computational and Experimental Science and ENgineering (IJCESEN)

Vol. 11-No.3 (2025) pp. 4429-4442
http://www.ijcesen.com

ISSN: 2149-9144

Research Article

Adaptive Hormesis-Based Optimization (AHBO) for Efficient Task Offloading in Industrial IoT Edge Environments

Amit Malik[1*], Amita Rani[2]

[1] Department of Computer Science and Engineering, SRM University, Delhi-NCR, Sonepat, Haryana, India.
*** Corresponding Author Email:** malik.dcrust@gmail.com- **ORCID**: 0009-0008-0407-7183

[2] Department of Computer Science and Engineering, DCRUST, Murthal, Sonepat, Haryana, India.
Email: amitamalik.cse@dcrustm.org- **ORCID:** 0000-0002-7385-4045

Article Info:

DOI: 10.22399/ijcesen.3067
Received : 22 March 2025
Accepted : 17 June 2025

Keywords

Hormesis,
Edge Computing
Nature Inspired Optimization
Task Offloading
Latency
Bio-inspired.

Abstract:

Adaptive Hormesis-Based Optimization (AHBO) is an adaptation of the Hormesis-Based Optimization (HBO) framework for task offloading in edge computing, targeting Industrial IoT (IIoT) environments. AHBO enhances the original HBO model by introducing an adaptive tuning method. The algorithm operates online, does not require any training and maintains almost a linear time complexity, making it suitable for edge scenarios in IIoT with a frequently varying environment. AHBO is evaluated against Reinforcement Learning Q-learning (RLQ), Harris Hawks Optimization (HHO), and Slime Mould Algorithm (SMA) across 12 different simulation configurations. It consistently outperforms RLQ, SMA and HHO algorithms in most configurations, offering up to 200–300% latency reduction in high-load, low-resource conditions. While SMA shows slight latency advantages in a few low-load cases, its decision time is still significantly higher than AHBO, making AHBO a compelling solution for real-time IIoT task scheduling under variable system stress.

1. Introduction

The landscape of modern computing is changing very rapidly. It is evolving at fast pace through optimized system performance with the help of efficient resource allocation and workload distribution [1]. The demand for optimization has particularly affected sectors including cloud computing, manufacturing, and real-time systems. This need is even more pronounced in the Industrial Internet of Things (IIoT), where the convergence of manufacturing machinery with real-time data processing creates complex and demanding operational environments. The complexity and variability in these environments make the adaptive solutions a necessity to maintain the efficiency and feasibility of the system [2]. In order to implement the optimization effectively, several types of methods are used. These methods have their pros and cons based on the demands of the scenario. For example, the traditional optimization methods, such as linear and integer programming, are effective for simpler, static settings, but their performance degrades when the number of constraints increases the complexity and demands for real time adaptation [3]. Similarly, heuristic methods are computationally efficient but may become trapped in local optima [4]. The machine learning (ML) techniques have emerged very strongly in last decade, including neural networks and deep learning, and offers very powerful solutions for complex, high-dimensional problems [5]. However, these methods require extensive datasets, substantial computational power, and model re-trainings in resource-constrained environments, limiting their applicability [6]. Therefore, the approaches which provide a quality solution with low computational cost and without intensive training-retraining phases can be beneficial [7]. These approaches can enhance decision-making processes in edge computing by effectively balancing resource utilization and performance requirements, ultimately leading to improved system efficiency.

Edge computing brings data processing closer to the source of computation, Figure 1. It significantly reduces task latency and increases the real-time performance and reliability [8]. This capability is crucial for IIoT applications, where delays in processing data from sensors and actuators can

impact production efficiency and safety. However, edge servers are not as capable as cloud servers and often possess limited computing power and storage capacity. It means that increasing workload beyond a certain point reduces the efficiency of the edge servers because their scalability is limited and it may become impractical to offload all tasks to them [9]. Hence, a dynamic and intelligent task offloading strategy to select computing nodes optimally is needed [10]. The advanced methods like reinforcement learning (RL) algorithms have done well in dynamic environments [11]. These architectures use agents which learn from interactions but their deployment in complex mobile edge computing (MEC) environments faces difficulties such as inaccurate reward calculations due to queuing, which leads to suboptimal task scheduling [12]. Deep Reinforcement Learning (DRL) models also incur significant computation and storage overhead due to the use of complex neural network architectures [13].

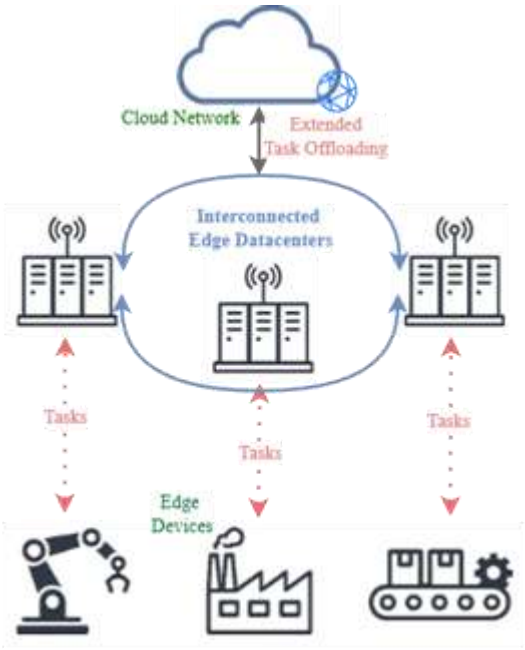

Figure 1. *Overview of the edge computing architecture considered in this study.*

In response to these challenges, the biologically inspired algorithms can be a promising solution. These methods often pack inherent adaptive mechanisms which are simple to manifest and can overcome many limitations of traditional and ML approaches [14]. The Hormesis-Based Optimization (HBO) algorithm is a biologically inspired computational approach which is designed to achieve optimization with low computational cost [15]. HBO is fundamentally inspired by the phenomenon of hormesis, where small doses of stress can strengthen biological systems and enhance their performance in extreme environments [16]. In a biological system, stress can be the excessive workload on muscles during exercise, the exposure to low levels of radiation that trigger cell repair, or the temporary lack of food during fasting which help the body adapt and become stronger [17]. Equivalently, in computational systems, the multiple performance parameters can be translated into stress which helps to trigger certain behavior. Appropriate and controlled application of stress gives beneficial results in natural system and HBO attempts to use this characteristic in the computational systems. The key strength of HBO lies in its ability to tackle a NP-hard problem in multi-constrained systems without exhaustive exploration of all possible solutions.

HBO converts multiple system parameters into a unified "stress" metric, for a predictive-reactive dynamic job shop scheduling problem (DJSS) [18], simplifying a complex and multi-constrained optimization into a single-objective problem. This strategy enables the HBO to achieve reduced time complexity of the order of $O(n\,log\,n)$ which is usually at least $O(n^2)$ for such NP-hard problems. HBO achieves this through rule-based framework which provides straightforward solution, but, with adaptive functionalities.

In this work, we introduce Adaptive HBO (AHBO) algorithm which is an enhanced version and incorporates automated hyperparameter tuning into the core stress-based decision process. In highly dynamic edge computing environments, where workload characteristics change rapidly, static parameters can lead to suboptimal performance [19]. AHBO addresses this by continuously fine-tuning key control parameters based on real-time performance feedback. The fundamental shift from fixed heuristics to self-optimizing control differentiates AHBO from its predecessor, enabling robust and context-aware adaptation to fluctuating system demands. We demonstrate how AHBO addresses the critical need for simple, deterministic, yet adaptive methods that do not require extensive training data or possess rigid strategies. We rigorously compare AHBO's performance against three prominent optimization techniques, such as Harris Hawks Optimization (HHO) [20] and Slime Mould Algorithm (SMA) [21] and ϵ-greedy Q-Learning method (RLQ) [22] which is a tabular reinforcement learning approach, for making offloading decision in edge computing scenario. The key contributions of this study are:

a) We provide a detailed explanation of the AHBO algorithm, highlighting its adaptive parameter tuning mechanism that distinguishes it from the original HBO.
b) We emphasize the crucial advantage of adaptive, biologically inspired algorithms that

operate effectively without large datasets or complex training phases, making them ideal for resource-constrained and dynamic environments.

c) We present a comprehensive comparative analysis of AHBO against HHO, SMA and RLQ in an edge computing scenario, evaluating key performance indicators such as latency, tail latency, maximum task completion time, and decision cost.

d) We demonstrate that AHBO offers a robust and efficient solution for resource optimization, proving its competitive performance against other advanced meta-heuristic algorithms while maintaining a low computational footprint and inherent adaptability which is highly relevant to relevant in IIoT settings.

The remainder of this paper is organized as follows. Section 2 reviews related work on task offloading strategies in edge computing. Section 3 presents the design of the proposed AHBO algorithm. Section 4 describes the simulation setup, including system parameters, task models, and performance metrics. Section 5 reports the experimental results, comparing AHBO with three baseline methods (HHO, SMA, and RLQ) across varying workloads and infrastructure scales, followed by a detailed analysis and discussion. Finally, Section 6 concludes the paper and outlines potential directions for future work.

2. Related Work

Traditional computational techniques such as linear programming, dynamic programming, and heuristic algorithms have long served as the foundation for solving optimization problems in MEC. However, these methods face severe limitations under dynamic environments. For instance, linear programming assumes linear relationships and struggles with scalability, especially when integer or binary decision variables are required [3, 15, 23 and 24]. Dynamic programming methods are capable of breaking down problems into manageable sub-problems, but their efficiency is hampered when used for problems dealing with large number of dimensions, making it computationally infeasible for real-time MEC scenarios [23, 25].

Heuristic algorithms like Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) are widely used due to their simplicity and scalability to large problems. Nonetheless, they also often get trapped in local optima and have difficulty balancing exploration and exploitation, particularly in real-time mobile settings [4, 11, 26 and 27]. Moreover, they lack adaptability to evolving conditions, which is critical in MEC where workloads, network states, and resource availability change rapidly [24, 26]. The computational overhead required for iterative exploration also adds to their inefficiency in latency-sensitive contexts [12, 23].

These limitations have sparked growing interest in nature-inspired and bio-inspired methods. Algorithms like Ant Colony Optimization, Firefly Algorithm, and Differential Evolution emulate natural processes and are shown to be more resilient and effective in high-dimensional search problems [20, 26, 27, 28]. Their inherent stochasticity and feedback mechanisms allow these models to adapt better to dynamic states, outperforming static approaches across various performance indicators such as convergence time, load balancing, and energy efficiency [26, 28].

Among these, algorithms based on biological stress adaptation mechanisms—particularly hormesis—are receiving attention for their capacity to encode environmental variability as a stimulus for system-level resilience. Hormesis, defined as a biphasic response to stress where low doses improve function and high doses inhibit it, has been integrated into computing to support dynamic system stabilization [16, 17, 30 and 31]. HBO operationalizes this concept by translating multiple parameters into a single stress signal that guides optimization—streamlining computation while retaining adaptability [15, 16].

Recent advances suggest that adaptive hyperparameter tuning further enhances such methods. Static parameters, while useful in stable settings, limit responsiveness in unpredictable environments. Studies show that real-time tuning frameworks improve learning and forecasting across fields from machine learning to epidemic modeling [10, 19, 32 and 33]. Bio-inspired designs like Artificial Hormone Systems (AHS) and hormone-guarded agents in organic computing demonstrate robust decision-making and fault-tolerance in distributed systems [28, 34]. These developments support the motivation behind AHBO, which aims to combine biological resilience with dynamic adaptability for edge task offloading.

3. Methodology

In this section, we detail the proposed Adaptive Hormesis-Based Optimization (AHBO) algorithm, which is an enhancement of the Hormesis-Based

Algorithm 1: Adaptive Hormesis-Based Optimization (AHBO) for Edge Computing
Input:
N: Number of Edge Datacenters (EdgeDCs)
T: Number of time steps
k: Adjustment factor for workload allocation
θ: Redistribution factor for workload balancing
δ: Threshold for significant workload adjustments
μ: Performance metrics for EdgeDCs
T_{adj}: Hormetic zone adjustment period
P_{low}, P_{high}: Percentiles for hormetic zone boundaries
a) Calculate Metric Weights
1. $w_j = \lvert corr(\mu_{i,j}, L_i)\rvert$
2. Normalize: $\Sigma_j w_j = 1$
b) Initialize Hormetic Zones for each EdgeDCs
3. Compute stress metric: $\phi_i = \Sigma_j w_j \cdot \mu_{i,j}$
4. Compute hormetic zone: $Z_i = [\alpha_i, \beta_i]$, **where**,
5. $\alpha_i = percentile\ (\phi_i,\ P_{low})$
6. $\beta_i = percentile(\phi_i,\ P_{high})$
c) Real-time Workload Adjustment
7. **for each time step** t:
8. **for each new task** τ:
9. Get $\mu_{i,j}(t)$
10. Compute stress metric: $\phi_i = \Sigma_j w_j \cdot \mu_{i,j}(t)$
11. Choose $EdgeDC_i = \text{argmax}_i[\beta_i - \phi_i(t)]$
12. Assign τ to $EdgeDC_i$
d) Task Re-distribution
13. **for each** $\boldsymbol{EdgeDC_i}$ where $\phi_i(t) > \beta_i + \theta$:
14. compute excess load: $\Delta_i = k \cdot (\phi_i(t) - \beta_i) \cdot \delta$
15. Identify tasks to offload using Δ_i
16. offload to $EdgeDC_u$ with $\phi_u(t) < \alpha_u$
e) Hormetic Zone Readjustment
17. **if** $\boldsymbol{t \bmod T_{adj} = 0}$
18. Recalculate Hormetic Zones
19. Update zone: $Z_i = [\alpha_i, \beta_i]$
f) Adaptive Parameter Tuning
20. Call: "Adaptive Parameter Tuning Subroutine" (Algorithm 2)

Optimization (HBO) algorithm [15] and is designed for dynamic computation offloading decisions in edge computing environments. The algorithm is presented in two parts: Algorithm 1, AHBO, outlines the adaptation of core HBO mechanism for edge computing scenario. It makes real-time offloading decisions for incoming computation tasks to Edge datacenters by using the principles of hormesis, aiming to balance the workload and minimize latency. Algorithm 2 then describes the details of adaptive tuning process subroutine used in AHBO, which continuously adjusts the operational parameters. The proposed AHBO algorithm aims to enhance the performance by making intelligent task scheduling based on real-time performance feedback in rapidly changing edge computing scenarios.

3.1. Adaptive Hormesis-Based Optimization (AHBO) for Edge Computing

In this subsection, we present the details of the proposed AHBO offloading strategy which is described in the Algorithm 1. AHBO is the decision-making component which is responsible for orchestrating the dynamic allocation of computational tasks across Edge Datacenters (EdgeDCs) in edge computing scenario. It continuously monitors key performance indicators of latency, queue length, and CPU utilization from various edge datacenters in the network and these parameters serve as ingredients for the computational stress calculated for each EdgeDC. It uses the biological principle of hormesis which says that the controlled doses of stress induce beneficial adaptation in a natural system. In this case the computational stress is used to produce the positive effects in the computational system comprising of edge datacenters. AHBO aims to maintain system performance within optimal operating ranges, referred to as hormetic zones, under varying computational loads in an edge computing scenario. AHBO inherits its conceptual framework from the original Hormesis-Based Optimization (HBO) algorithm [15], including the definitions of stress, dose, and hormetic boundaries, along with control parameters such as the adjustment factor k, a redistribution coefficient θ, a sensitivity threshold δ for filtering out minor fluctuations from the original HBO algorithm. In addition, AHBO introduces an adaptation interval T_{adj}, which determines how frequently the system re-evaluates its hormetic zones. Percentile boundaries P_{low} and P_{high} are also employed to demarcate the upper and lower bounds of these zones for each EdgeDC.

The algorithm AHBO begins by reading the historical performance data from each EdgeDCs. In step 1.1, it reads three key input metrics: CPU utilization, queue length, and response latency. In this discussion, step 1.1 will refer to line numbered as 1 in Algorithm 1 and step 2.1 will refer to line numbered as 1 in Algorithm 2, and so on. So, the metric $\mu_{i,j}$ in step 1.1 represents the internal state of an EdgeDC and is used to convert this internal state into a unified value referred to as "computational stress" or simply "stress" (ϕ_i) on each EdgeDC in step 1.3. In order to produce a unified stress value, AHBO algorithm first calculates the absolute correlation $|corr(\mu_{i,j}, L_i)|$ between each metric μ_j (j= CPU utilization, Queue Length, Latency) and latency L_i of that edge datacenter and then normalize (step 1.2) these correlation values to produce a set of weights w_j which reflect their relative importance.

The stress value ϕ_i is used to determine that whether an EdgeDC is underutilized, overburdened or is in its optimal zone. As per the hormesis principle, the effect of stress on a natural system is beneficial within a range of stress doses. If the stress is lesser than, or increases this range, then the performance of the system declines. This optimal range, where the effect of the stress is beneficial, is referred to as hormetic zone and is defined by upper and lower bound of the beneficial range in natural systems. In AHBO, the hormetic zone is defined as $Z_i = [\alpha_i, \beta_i]$ for each EdgeDC and is calculated as shown in step 1.4 to 1.6. Any EdgeDC is underutilized if $\phi_i < \alpha_i$ (lower bound) and is overburdened if $\phi_i > \beta_i$ (upper bound). Both underutilized and overburdened EdgeDCs are not beneficial for the system of EdgeDCs. The idea of the algorithm is to keep the EdgeDCs within hormetic zone, as much as possible, in order to minimize the latency among the offloaded tasks in the overall system. According to AHBO algorithm, for each new task τ arriving at time t, the most underutilized machine is located by using calculations, given in step 1.9 to step 1.12, and the new tasks are assigned to it. The excessive load Δ_i is also redistributed to comparatively less burdened EdgeDCs if any aree found to be overburdened. In such a case, the amount of excess load is calculated, and the corresponding number of tasks is migrated from the overburdened to comparatively underutilized EdgeDCs. This process is depicted by steps 1.13 to 1.16, where k, the adjustment factor determines how strongly the algorithm reacts to excess stress, and δ, the redistribution coefficient controls how aggressively workload is moved to other available datacenters. The stress on an EdgeDC is directly comparable to latency and we offload the tasks to underutilized EdgeDCs in proportional to the difference between upper bound β_i and stress $\phi_i$registered on these datacenters.

The procedure described so far aims to keep the EdgeDCs within their hormetic zone. However, when the rate at which tasks are received by the EdgeDCs increases permanently, or when all the EdgeDCs are continuously underutilized, we may need to re-adjust the hormetic zones for EdgeDCs. In AHBO, we achieve it by re-computing the hormetic zone after a fixed interval of time controlled by variable T_{adj}, as shown in steps 1.17 to 1.19. In the end, AHBO concludes each simulation cycle by refining its control parameters through a subroutine call mentioned in step 1.20, which implements adaptive feedback mechanism into AHBO to fine-tune itself across cycles described in Algorithm 2.

Algorithm 2: Adaptive Parameter Tuning Subroutine
Input:
$P_{curr} = \{k, \theta, \delta, T_{adj}\}$: Current parameter values
$f_{curr} = \{\bar{L}, C_{max}\}$: Current performance metrics
$f_{prev} = \{\bar{L}, C_{max}\}$: Previous performance metrics
$D = \{d_k, d_\theta, d_\delta, d_{T_{adj}}\}$: Adjustment directions, where $d \in \{-1, 1\}$
Z_{param} : Hormetic zones for parameters
$S_p = \{s_k, s_\theta, s_\delta, s_{T_{adj}}\}$: Step sizes
f_{best} : Best performance metrics
P_{best} : Parameters corresponding to f_{best}
ϵ : Performance change threshold
a) Calculate Performance Change
1. **if** $f_{prev} = None$, **continue**
2. Compute change: $\Delta f = f_{curr}[\bar{L}] - f_{prev}[\bar{L}]$
b) Determine Adjustment and Update Direction/Step Size
3. **if** $(\Delta f < -\epsilon)$ **then**
4. $adj = D_p \cdot S_p$
5. **else if** $(\Delta f > \epsilon)$:
6. $P_{curr} = P_{best}$
7. $S_p = max(S_p \cdot 0.5, S_{min})$
8. $D_p = -D_p$
9. $adj = D_p \cdot S_p$
10. **else** :
11. $adj = D_p \cdot S_p$
c) Apply Adjustment and Enforce Bounds
12. Update parameter: $P_{next}[p] = P_{curr}[p] + adj$
13. **if** $(P_{next}[p] < Z_{param}[p_{min}])$:
14. $P_{next}[p] = Z_{param}[p_{min}]$
15. $D_p = -D_p$
16. **if** $(P_{next}[p] > Z_{param}[p_{max}])$:
17. $P_{next}[p] = Z_{param}[p_{max}]$
18. $D_p = -D_p$
d) Ensure Integer for T_{adj}
19. **if** $(p = T_{adj})$:
20. $P_{next}[T_{adj}] = \lfloor P_{next}[T_{adj}] + 0.5 \rfloor$
21. $P_{next}[T_{adj}] = max(P_{next}[T_{adj}], 1)$

3.2. Adaptive Parameter Tuning Subroutine for AHBO

The pseudocode for adaptive parameter tuning subroutine for AHBO is given in Algorithm 2. It re-adjustments the key parameters $(k, \theta, \delta, T_{adj})$ used by Algorithm 1 and enables the AHBO framework to continuously self-optimize in ever-changing edge computing environments. This adaptive tuning mechanism improves performance metrics, average latency (L) and maximum completion time C_{max}. The algorithm takes the current, previous, and best-recorded performance metrics as input, along with adjustment directions D, parameter-specific hormetic zones Z_{param} , and step size S_p. The subroutine first calculates the amount of change (Δf) in performance metric, step 2.1 and 2.2. The change will be none at initialization, hence the algorithm will continue without adaptation at the beginning. If the change in performance parameter is in desired direction (step 2.3), then we

continue to adapt by calculating the adjustment amount as per step 2.4 and update the parameters $(k, \theta, \delta, T_{adj})$ using step 2.12. If the performance metric degrades (step 2.5), then we fall back to the parameter values associated with best value recorded for performance metric so far, using step 2.6 and decides the amount of adjustment using the equations in step 2.7 to step 2.9. Consequently, the new parameter values are decided using the step 2.12. Additionally, we can define the boundaries for the lowest and highest values allowed for each of the parameter $(k, \theta, \delta, T_{adj})$ using the predefined hormetic zone Z_{param} as per Step 2.13 to Step 2.18. In Step 2.19 to Step 2.21, we ensure that the value of T_{adj} is integer as it defines a time interval for hormetic adjustments in AHBO.

3.3. Simulation Setup

To evaluate both our proposed and baseline offloading strategies, we built a lightweight, time-stepped simulation environment using Python. This simulator represents a network of edge datacenters, each of which tracks dynamic system metrics like CPU usage, queue depth, and latency. The tasks are introduced into the scenario by using pre-built traces and follows discrete time intervals in scheduling. We have used both online and offline strategies for comparison and the tasks are dispatched in real-time or batch mode depending upon the strategy. The offline strategies (HHO and SMA) use batch mode, whereas the online strategies (AHBO and RLQ) uses real-time task dispatch [8]. The simulator is much simpler than the other simulators used in this domain, but the idea is to captures key system-level behaviors, such as, load balancing, latency impacts, and decision overhead, which are important in this case scenario. The architecture of the simulator, as depicted in Figure 2, is organized into different modular components, such as, the EdgeSimulatorEngine, which manages the simulation timeline and facilitates interactions across multiple modules. The module which handles the task generation and scheduling is TaskArrivalModel module. This module follows predefined patterns and processing demands. The InfrastructureModel module includes the edge datacenter layer (EdgeDCs), while the TaskExecutionEngine module manages task queues and execution based on available capacity. The OffloadingPolicyInterface module controls the task orchestration, offloading and redistribution, and is the place where the core logic of different offloading algorithms is introduced. Further, the simulation events, system states, and task life cycles are tracked by the MonitorAndLogger module, and the summarized performance metrics are exported via the ResultsExporter module.

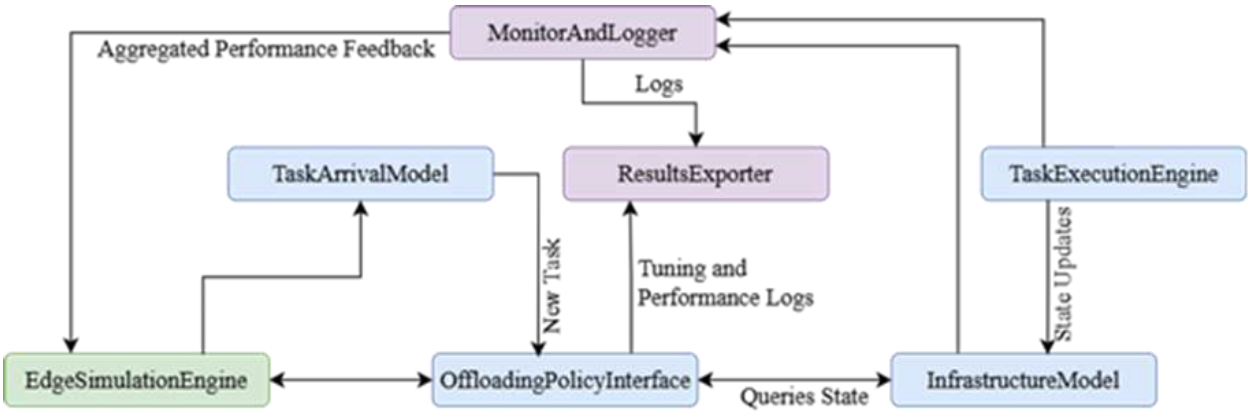

Figure 2. *The modular architecture of the custom simulation environment.*

The simulation experiments model task offloading scenarios across infrastructures comprising 3, 5, 10, and 15 EdgeDCs. For each configuration, simulations are conducted using task sets of 500, 1000, and 5000 tasks, with a dynamically varying task generation rate. In this setup, the task arrival times and processing demands are sourced from pre-generated traces to ensure uniformity in the execution conditions and input data across all algorithms which are being assessed. The simulation parameter and the algorithm specific configurations are listed in the Table 2.

The experimental setup is carefully designed for evaluating the AHBO algorithm for its comparison with leading metaheuristic and reinforcement learning approaches. We explored a wide range of simulation scenarios by varying the number of Edge Datacenters from 3 to 15, and running task sets of 500, 1000, and 5000 tasks for each configuration, as summarized in Table 2.

However, AHBO and RLQ are online, adaptive strategies that make decisions on the fly as tasks arrive and HHO and SMA are offline optimizers that work by computing an entire batch of tasks in advance. This fundamental difference between these two categories of algorithm brings specific challenges in measuring performance. We have addressed these challenges by using the metric for decision overhead and computational cost. As the online methods spread decision-making throughout the simulation, with small, frequent computations, while offline methods require a larger optimization effort before execution even begins, we have provisioned the simulator to track the total decision overhead for each algorithm during the experiments.

We also put significant effort in configuring RLQ and to obtain right set of hyperparameters. We carried out an in-depth tuning phase, testing multiple combinations of learning rate, discount factor, and epsilon decay. The result of the tuning effort is shown in the 3D surface plot, Figure 3, which shows that how the learning rate and discount factor affect average latency. This plot highlights how sensitive performance can be to these choices and confirms that RLQ was fine-tuned to perform at its best.

Table 2: Simulation Configuration Parameters

Parameter	**:**	**Value(s)**
Number of Edge Datacenters (N)	:	$3, 5, 10, 15$
Total Tasks (Jobs)	:	$500, 1000, 5000$
Task Arrival Model	:	$Traced$
Task Processing Time	:	$Uniform$
Metrics Used ($\mu_{i,j}$)	:	$CPU\ Utilization, Queue\ Leng$
AHBO Specific Configuration		
Adjustment factor (k)	:	$Initial$: 1.5, $Range$: [1.0, 2.0]
Redistribution factor (δ)	:	$Initial$: 1.5, $Range$: [1.0, 2.0]
Adjustment Threshold (θ)	:	$Initial$: 0.3, $Range$: [0.1, 0.5]
Hormetic zone adjustment period (T_{adj})	:	$Initial$: 10, $Range$: [5, 20]
HHO/SMA Specific Configuration		
Population Size	:	$30, 50, 100$
Max Iterations	:	$20, 50$
Objective Function	:	$L_{max+0.1} \cdot \bar{L}$
RLQ Specific Configuration		
Learning Rate	:	$Tuned(\ 0.1, 0.2, 0.5)$
Discount Factor	:	$Tuned\ (0.7, 0.9)$
Epsilon Decay Rate	:	$Tuned\ (0.995, 0.999)$
Min Epsilon	:	$0.01\ /\ 0.05$
Training Episodes	:	$200\ (tuning), 1000\ (final\ ru$

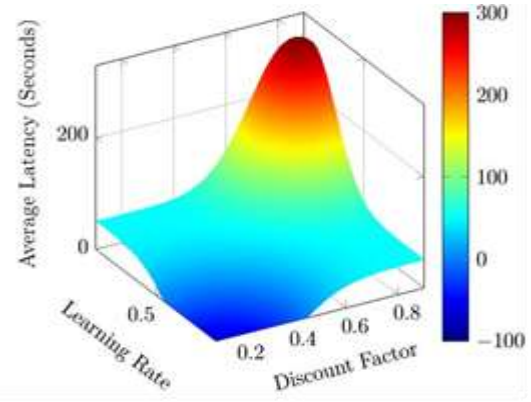

Figure 3. *3D surface plot illustrating the hyper parameter tuning for the RLQ agent; highlighting the optimal parameter space identified*

The performance of each algorithm is evaluated using the following key indicators:

a) *Average Latency (L):* The mean total time from task arrival to its completion.
b) *Maximum Completion Time* (L_{max})*:* The total time elapsed until the last task in a given simulation run is completed.
c) *99th Percentile Latency:* To capture tail latency, indicating the worst-case performance for the majority of tasks.
d) *Total Decision Overhead:* The cumulative computational time spent by an algorithm in making offloading and redistribution decisions. For batch optimizers, this represents their total optimization time.

Finally, all simulations in this study were executed on the Google Colab platform using its default CPU runtime 2-core Xeon processor, with approximately 12 GB RAM and no GPU or TPU acceleration was employed during the execution for any of the algorithm evaluated here.

4. Results and Analysis

This section presents the experimental evaluation of the AHBO algorithm against the metaheuristic algorithms HHO, SMA, and reinforcement learning based RLQ. The algorithms used as baseline here are well performing algorithms used in various edge computing scenarios in recent studies [21, 35 and 36]. We analyze the performance using key indicators, such as, average latency, makespan, 99th percentile latency, and total decision overhead, to provide an extensive understanding of the efficiency of each algorithm.

4.1.Results

The simulation results presented in this sub-section are obtained from the experimental setup which is

detailed in Section 4 and are presented across different scenarios varying in the number of Edge Datacenters (3, 5, 10, 15) and total tasks (500, 1000, 5000).

Table 3 provides the relative percentage difference in average latency between AHBO and the baseline algorithms used in this study. The metric used to generate these values is relative percentage difference computed by using the formula $\left(\left(\frac{L_{Baseline}-L_{AHBO}}{L_{AHBO}}\right)\times 100\right)$. In these readings, which are measured across the scenarios, AHBO demonstrates significant improvements in average latency, often reducing it by over 200–300% compared to HHO and RLQ. The performance advantage is particularly noticeable in resource-constrained, high-load scenarios. For instance, with 3 EdgeDCs managing 5000 tasks, AHBO achieves latency reductions of 306% and 194% relative to HHO and RLQ, respectively.

Table 3. *Relative percentage difference in average latency between AHBO and baseline algorithms across scenarios.*

No. of EdgeDCs	No. of Tasks	Relative Difference in Average Latency with AHBO (%)		
		HHO	SMA	RLQ
3	500	250.04	249.23	96.6
3	1000	286.23	269.41	43.12
3	5000	306.29	301.33	193.8
5	500	263.01	142.81	245.16
5	1000	305.95	198.92	300.33
5	5000	311.45	233.41	413.25

The extent of these improvements was initially unexpected. However, thorough validation of both the implementation and metric calculations confirms that AHBO has performed consistently well because the adaptive stress balancing and dynamic hormetic zone re-adjustment mechanism contributes to its strong performance across different workload conditions. Nevertheless, further analysis is required to fully understand the magnitude of this improvement, particularly when compared to optimization-driven strategies operating under identical constraints.

In contrast to HHO and RLQ, SMA outperforms AHBO in certain large-scale, low-load configurations, e.g. when the number of EdgeDCs (≥ 10) is more and number of tasks (=500) is lesser, reducing the latency up to 53%. This outcome highlights the advantage of batch-optimized methods when workloads are predictable and system resources are sufficiently available. However, this improvement comes with an increased decision overhead, as SMA introduces significantly higher scheduling latency explained in subsequent subsections.

Continually, Table 4 provides a detailed overview of the key performance metrics, including, Average Latency, Maximum Completion Time, 99th Percentile Latency and Total Decision Overhead. It reports the values achieved by AHBO and the comparative algorithms HHO, SMA, RLQ across all tested edge datacenter configurations and task loads. Each value represents the aggregate performance observed over the respective simulation cycles or optimization runs, offering a precise numerical foundation for the visual trends and in-depth analysis discussed in the subsequent subsections. Overall, AHBO presents a well-balanced approach to offloading by combining responsiveness with performance. Its capability to dynamically adapt to varying workload demands while maintaining low latency makes it a suitable option for real-time edge computing environments.

4.1.1 Average Latency ($\bar{L}$)

Figure 4 illustrates the trend reported in Table 3 and Table 4 visually. It shows the comparative performance of AHBO, HHO, SMA, and RLQ in terms of average latency plotted against different

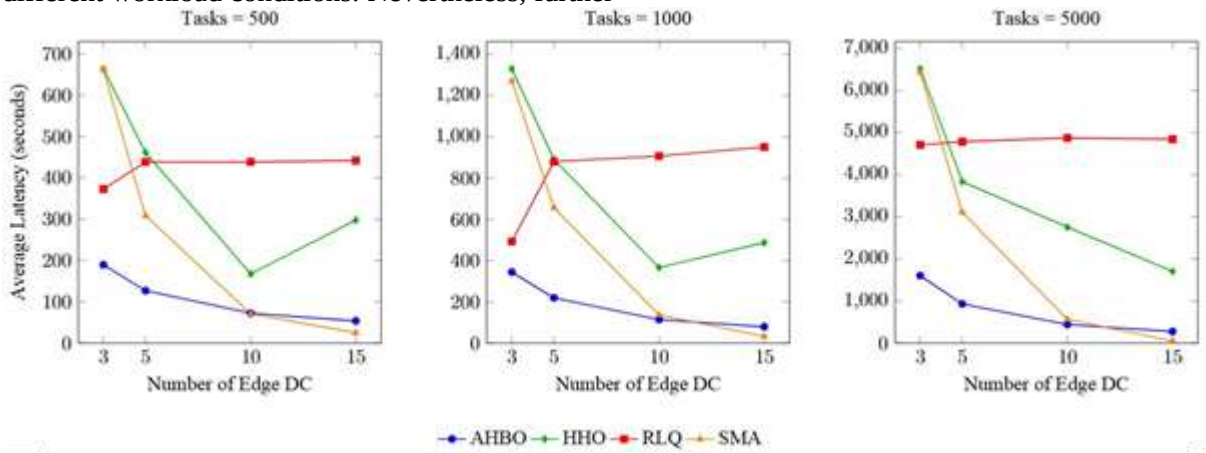

Figure 4. *Comparison of average latency for AHBO, HHO, RLQ and SMA across different scenarios*

Table 4. *Results obtained for Average Latency, Maximum Completion Time and 99th percentile Latency for different simulation scenarios*

No. of EdgeDCs	No. of Tasks	Algorithm	Average Latency	Max Completion Time	99th Percentile Latency
3	500	AHBO	189.54	733.79	342.27
		RLQ	372.63	1219.26	819.68
		SMA	661.93	1683.89	1283.22
		HHO	663.46	1731.67	1317.54
	1000	AHBO	344.28	1440.95	650.87
		RLQ	492.75	1662.79	1044.4
		SMA	1271.8	3333.54	2533.16
		HHO	1329.71	3805.23	2967.03
	5000	AHBO	1602.33	6997.64	3142.33
		RLQ	4707.69	13407.53	9490.36
		SMA	6430.65	16710.16	12723.05
		HHO	6510.12	17800.85	13772.35
5	500	AHBO	127.0	620.55	217.96
		SMA	308.37	1044.03	632.08
		RLQ	438.35	1323.45	916.87
		HHO	461.02	1708.62	1276.32
	1000	AHBO	219.77	1158.39	397.01
		SMA	656.94	2037.35	1274.21
		RLQ	879.81	2604.56	1828.69
		HHO	892.16	3308.1	2516.42
	5000	AHBO	931.92	5579.96	1821.83
		SMA	3107.11	10088.3	6250.34
		HHO	3834.37	14769.52	10752.09
		RLQ	4783.04	13399.48	9559.64
10	500	SMA	70.98	576.68	168.71
		AHBO	72.39	526.2	114.55
		HHO	167.58	1135.89	688.17
		RLQ	438.36	1337.0	920.66
	1000	AHBO	114.43	966.89	196.45
		SMA	137.25	1063.88	295.4
		HHO	366.63	2342.08	1504.54
		RLQ	907.31	2648.63	1859.21
	5000	AHBO	441.09	4782.01	838.27
		SMA	562.85	5172.15	1186.24
		HHO	2751.22	14042.52	9749.48
		RLQ	4870.9	13698.21	9664.95
15	500	SMA	25.12	453.86	62.88
		AHBO	53.79	475.71	82.29
		HHO	297.76	1506.65	1069.39
		RLQ	441.74	1325.72	920.18
	1000	SMA	32.98	851.17	101.82
		AHBO	79.54	919.06	129.13
		HHO	487.5	2728.08	1860.31
		RLQ	951.48	2722.6	1910.67
	5000	SMA	53.43	3784.7	193.15
		AHBO	276.18	4255.65	512.92
		HHO	1704.82	11275.55	7373.6
		RLQ	4841.21	13459.08	9626.67

number of EdgeDCs using three graphs, one for each set of 500, 1000 and 5000 tasks. As shown in these graphs, the AHBO consistently outperforms HHO, SMA, and RLQ in minimizing average tasklatency for the heavy and moderate workload configurations. However, the trend changes for SMA rapidly for the configurations where enough resources are available and the workload is low. In such situations, as explained already, SMA outperforms even the AHBO algorithm by up to 53% whereas in high workload configurations, AHBO outperformed SMA by up to 300% (N=3, Tasks = 5000). The curves indicate that increase in latencies for HHO, SMA and RLQ can be exponential, whereas, AHBO shows almost linear curve as the complexity of the scenario increases.

4.1.2 Maximum Completion Time (C_{max})

Figure 5 depicts the maximum completion times achieved by each algorithm under the same varying conditions. As expected, the maximum completion time metric results mirror the trends observed for

average latency. AHBO's superior capability in reducing the total time required to complete all tasks is particularly evident in high-load scenarios. While SMA can achieve competitive makespan in smaller scenarios, the superiority of the AHBO in intensive environments is again verified by the fact that it has lesser maximum completion time than SMA in the scenario where SMA achieved lower average latency i.e. when N=10 and tasks=500. This reason for this trend is because the tail latency of AHBO is lesser than SMA for this case scenario.

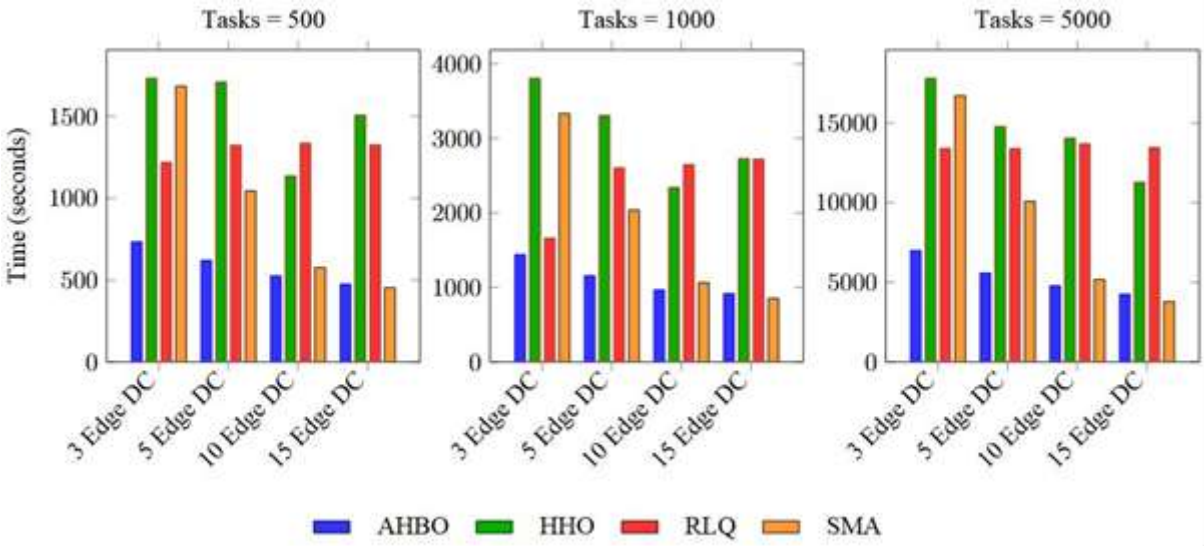

Figure 5. *Maximum completion time for AHBO, HHO, RLQ and SMA across different scenarios*

4.1.3 99th Percentile Latency

Figure 6 provides insights into the tail latency performance, indicating the worst-case experience for 99% of tasks. Crucially, the 99th percentile latency reveals AHBO's robustness in handling peak loads and avoiding significant delays for critical tasks. AHBO consistently maintains lower tail latencies compared to its counterparts across almost all configurations, indicating more predictable and stable system behavior even under stress. Also, as it is already seen in previous subsections, SMA performs well in scenarios where workload is lighter. However, its performance as per this metric is not as distinct as for average latency metric. Summary, AHBO is much more effective at preventing performance bottlenecks that disproportionately affect a small fraction of tasks.

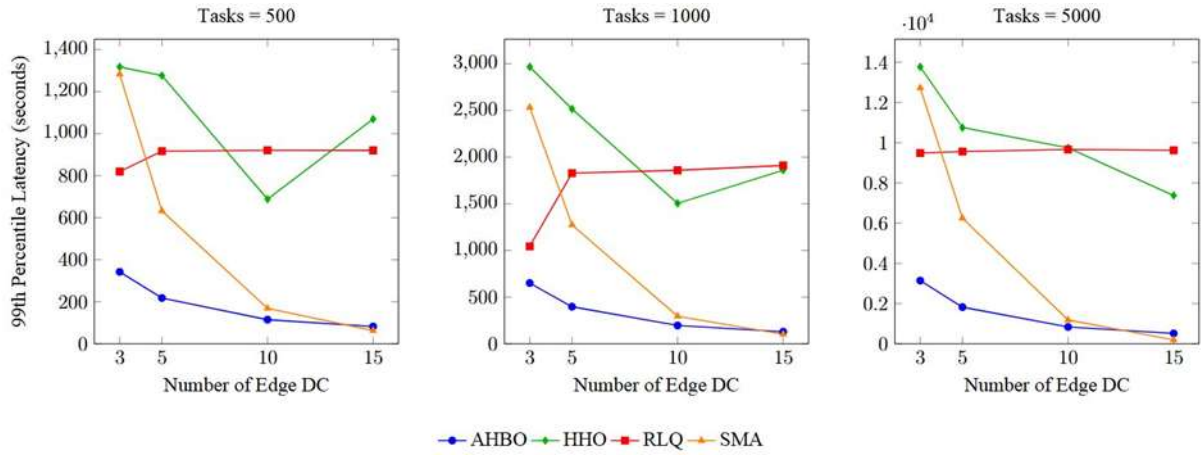

Figure 6. *Comparison of 99th percentile (tail) latency for AHBO, HHO, RLQ and SMA across different scenarios.*

4.1.4 Total Decision Overhead

Figure 7 illustrates the computational overhead associated with the decision-making process of each algorithm. It clearly differentiates between online (AHBO, RLQ) and offline (HHO, SMA) algorithms in terms of how this overhead is accumulated. While HHO and SMA incur a single, upfront optimization cost, AHBO and RLQ demonstrate continuous decision-making overhead

per time step. Despite being an online adaptive algorithm, AHBO maintains a remarkably low decision overhead, often comparable to the continuous decision costs of RLQ and significantly less than the upfront costs of HHO and SMA. Seemingly, the RLQ algorithm is quicker than all other algorithms but the decision overhead in Figure 7 does not include the RLQ model's training time, without which a machine learning or reinforcement learning algorithm may provide very unreliable solution. The training cost for RLQ, in terms of time, is depicted in Figure 8. As it is evident, the time required for training is on a much larger scale than the time required for decision-making. This makes AHBO highly suitable for real-time edge environments where immediate and computationally inexpensive decisions are critical because it requires no training or re-training.

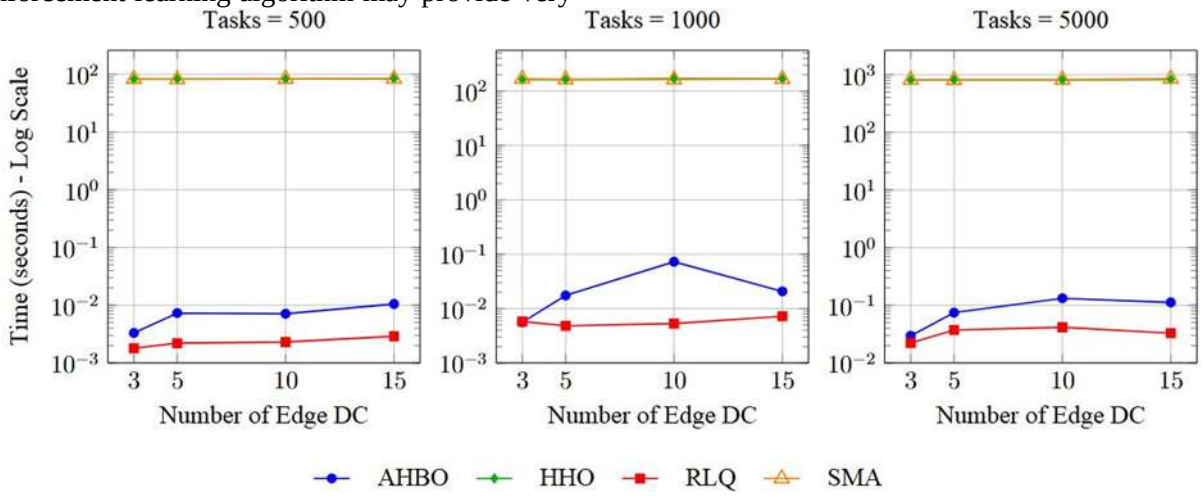

Figure 7. *Total decision overhead (log scale) for AHBO, HHO, RLQ and SMA across different scenarios.*

4.2. Analysis and Discussion

In this subsection we dive deeper to explore the underlying reasons and present more insights for the performances observed in section 5.1. This will help us to understand the position of AHBO algorithm in the broader context of edge computing optimization.

4.2.1 Behavioral Comparison

AHBO has consistently recorded superior performance, up to 200-300% decrease in latency during high workloads, which can be directly linked to its core hormesis principle and adaptive mechanisms. It maintains EdgeDCs within their optimal stress zones dynamically and balances the workload according to the situation taking very low decision time. This helps AHBO to prevent both underutilization and overburdening of edge datacenters. AHBO continuously fine tunes its control parameters $(k, \theta, \delta, T_{adj})$, by using the adaptive parameter tuning subroutine which allows AHBO to dynamically adjust its offloading and redistribution strategies in response to real-time changes in workload characteristics and state of edge datacenters.

In contrast, the metaheuristics algorithms like HHO and SMA are capable of finding high-quality solutions for a given static problem instance but their inability to adapt to sudden changes in conditions limits their effectiveness in edge computing scenarios. These methods find the solution depending upon initial static snapshot, which quickly degrades as the simulation progresses.

On the other hand, RLQ also falls behind due to complexity of state-action space and lack of continuous learning and policy updates. Reinforcement learning methods heavily depends on learning capabilities and can adapt to changing environments through its reward mechanism, but the state-action space it develops during the learning state might not be sufficient in highly varying edge computing scenarios. RLQ shows a competitive performance with batch processing methods, HHO and SMA, but is not able to match the adaptive capabilities of AHBO algorithm.

4.2.2 Trade-offs in Computational Cost and Scalability

This subsection introspect the trade-off between offloading accuracy and time taken in making offloading decisions, introduced as decision overhead in this study, and is the component of computational cost which we intend to minimize. The batch mode algorithms, SMA and HHO, optimize offloading through population-based exploration, but also incur significant overhead in

decision-making time. As shown in Figure 7, the decision time incurred by these algorithms per scheduling cycle remains around 100 to 1000 seconds, regardless of the scenario. These high values reflect the offline nature of their optimization logic, which involves full population evaluation and fitness evaluation for each batch of the tasks.

In contrast, Figure 7 also shows that AHBO and RLQ exhibit decision overheads that are three to four orders lower in magnitude, usually ranging from 0.005 to 0.15 seconds per time step. This places them well within real-time operational thresholds for edge environments. However, while RLQ's decision latency appears comparable to AHBO, it requires a substantial up-front training phase. Figure 8 show that cumulative training time of RLQ algorithm spans over ten thousand seconds in complex scenarios. This cost scales nonlinearly with the size of the system and the number of episodes used during policy convergence.

Together, these results establish that while all methods may appear viable from latency standpoint, their feasibility in real-time deployment depends on decision readiness and computational demand. AHBO achieves the most favorable trade-off, combining online responsiveness with robust average and tail latency performance, while SMA and HHO trade accuracy for significant computational cost, and the utility of RLQ algorithm is limited by its offline training requirements.

4.2.3 Time Complexity Analysis

The AHBO algorithm is designed to make decisions quickly and adaptively and this achievement is evident from decision time results in Figure 7. Unlike the other algorithms, AHBO runs online and makes task offloading decisions at each time step without the requirement to see the entire workload in advance. AHBO combines the important performance parameters like, CPU utilization, queue length, and latency into a single stress value which helps the algorithm understand the state of each datacenter and also keeps the decision making process efficient. Due to this strategy, the time complexity of AHBO stays close to $O(NlogN)$, where N is the number of edge datacenters. Even though AHBO includes an adaptive parameter tuning routine, but the routine does not run every for time step and only adjusts a few parameters, so it does not slow down the algorithm in real-time.

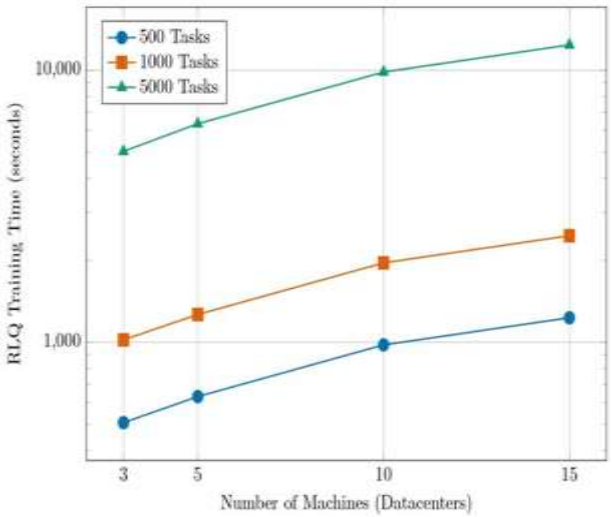

Figure 8. *Training time required for the RLQ agent; a cost that scales with system complexity and is entirely avoided by the training-free AHBO algorithm*

On the other hand, SMA and HHO use global optimization techniques that evaluate large populations of possible solutions. Their time complexity grows much faster and depends on the number of tasks (T), the size of the population (E), and the number of iterations (R). Normally, these algorithms are said to have time complexity $O(E \times T)$ with the assumption that population and number of iterations scale with number of tasks. This leads to high decision-making time as also evident in Figure 7. Similarly, the time complexity for tuning RLQ depends on multiple parameters such as, hyper parameter grid combination(G), number of training episodes (V), number of edge data centers (N), number of tasks(T), but assuming that the grid size and number of data centers remain small finite values, the time complexity of training RLQ can be reduced to $O(E \times T)$. Once the policy is learned, RLQ runs with linear task complexity. Hence, RLQ makes fast decisions once trained, but the training process itself is expensive. Figure 8 shows that RLQ training takes from hundreds to over ten thousand seconds depending on the number of datacenters and tasks which is a serious limitation in dynamic environments where system conditions change quickly, and re-training may be required frequently.

Overall, AHBO avoids both the heavy upfront cost of RLQ and the runtime delays of SMA and HHO. Its ability to make fast decisions using a single combined stress value, without needing full retraining or batch optimization, makes it well suited for real-time, latency-sensitive edge computing.

5. Conclusion

In this paper, we have addressed the problem of computing task offloading in dynamic edge computing environments tailored for Industrial IoT (IIoT) systems. Taking into account the latency sensitivity and the resource constraints of edge infrastructure, we formulated a biologically inspired decision-making mechanism based on hormetic adaptation. Building on this formulation, we proposed an Adaptive Hormesis-Based Optimization (AHBO) strategy that unifies system stress indicators and dynamically adjusts offloading policies using percentile-based hormetic zones. The aim is to optimize latency while maintaining low decision overheads suitable for real-time scenarios. The simulation results confirm that AHBO significantly reduces average latency compared to existing techniques such as HHO, SMA, and Q-learning, particularly under high-load or bottleneck conditions. In future, we plan to extend this methodology to incorporate energy consumption and to reduce the computational cost further. We also intend to explore its application in decentralized and federated edge architectures, where distributed learning and decision making across multiple clusters of edge datacenters may be required.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.

References

[1] Al-Ali, S., & Assalem, A. (2023). Metaheuristics method for computation offloading in mobile edge computing: Survey. *Journal of Advanced Research in Applied Sciences and Engineering Technology*, *36*(1), 43–73.

[2] Alzate, I. C., et al. (2024). Flexibility and adaptability: Dynamic capabilities for building supply chain resilience. *Journal of Infrastructure, Policy and Development*.

[3] Arora, J. S. (2025). *Nature-inspired metaheuristic search methods*.

[4] Casale, G. (2024). *Optimizing Edge AI: Performance engineering in resource-constrained environments* (s. 223). https://doi.org/10.1145/3629526.3649131

[5] Chen, P.-Y., & Jasiuk, I. (2024). Biological and bio-inspired materials: Multi-scale modeling, artificial intelligence approaches, and experiments. *Journal of Materials Research and Technology*. https://doi.org/10.1016/j.jmrt.2024.05.117

[6] Duan, Z. H., Qian, X., & Song, W. (2025). Multi-strategy enhancde slime mould algorithm for optimization problems. *IEEE Access*, *1*. https://doi.org/10.1109/access.2025.3527509

[7] Durairaj, S., Umar, M. M., & Natarajan, B. (2025). Evaluation of bio-inspired algorithm-based machine learning and deep learning models. In *Evaluation of Bio-Inspired Algorithm-Based Machine Learning and Deep Learning Models* (s. 48–69).

[8] Esmaeili, M., Khonsari, A., Sohrabi, V., & Dadlani, A. (2024). Reinforcement learning-based dynamic load balancing in edge computing networks. *Computer Communications*. https://doi.org/10.1016/j.comcom.2024.04.009

[9] Fan, Q., Chen, Z., & Xia, Z. (2020). A novel quasi-reflected Harris hawks optimization algorithm for global optimization problems. *Soft Computing*, *24*(19), 14825–14843.

[10] Giudice, M. D., et al. (2018). What is stress? A systems perspective. *Integrative and Comparative Biology*, *58*(6), 1015–1030.

[11] Gómez Larrakoetxea, N., Sanz, B., Pastor-López, I., García-Barruetabeña, J., & Bringas, P. G. (2024). Enhancing real-time processing in Industry 4.0 through the paradigm of edge computing. *Mathematics*, *13*(1), 29.

[12] Gulić, M., Žuškin, M., & Kvaternik, V. (2023). An overview and comparison of selected state-of-the-art algorithms inspired by nature. *TEM Journal*, *12*(3).

[13] Gupta, A., Sharma, A., Wei, C. L., & Ravi, M. (2024). Integrating evolutionary algorithms and mathematical modeling for efficient neural network optimization. *Advances in Machine Learning & Artificial Intelligence*, *5*(4), 01–06.

[14] Huang, L., & Yu, Q. (2024). Mobility-aware and energy-efficient offloading for mobile edge computing in cellular networks. *Ad Hoc Networks*, *151*, 103472.

[15] Latip, R., et al. (2024). Metaheuristic task offloading approaches for minimization of energy consumption on edge computing: A systematic review. *Discover Internet of Things*.

[16] Li, S., Chen, H., Wang, M., Heidari, A. A., & Mirjalili, S. (2020). Slime mould algorithm: A new method for stochastic optimization. *Future Generation Computer Systems*, *111*, 300–323.

[17] Li, X., Yang, T., & Sun, Z. (2019). Hormesis in health and chronic diseases. *Trends in Endocrinology and Metabolism*, *30*(12), 944–958.
[18] Lindsay, D. G. (2005). Nutrition, hormetic stress and health. *Nutrition Research Reviews*, *18*(2), 249–258.
[19] Liu, M., & Wang, J. (2022). Parameter adaptive SEIRD model for epidemic prediction. In *Chinese Control and Decision Conference* (s. 1277–1282).
[20] Liu, X., Qin, Z., & Gao, Y. (2019). *Resource allocation for edge computing in IoT networks via reinforcement learning*.
[21] Malik, A., & Rani, A. (2025). Hormesis-based optimization (HBO) algorithm: A biologically inspired computational approach. *Journal of Information Systems Engineering and Management*, *10*(49s), 1229–1254.
[22] Masadome, S., & Harada, T. (2025). Reward design using large language models for natural language explanation of reinforcement learning agent actions. *IEEJ Transactions on Electrical and Electronic Engineering*. https://doi.org/10.1002/tee.70005
[23] Nahar, S., Raj, U., Meckel, S., & Obermaisser, R. (2024). Enhancing reliability in organic computing using hormone guard. *IEEE MECO*, 10577946.
[24] Ogunsakin, R., Mehandjiev, N., & Marín, C. A. (2023). Towards adaptive digital twins architecture. *Computers in Industry, August 1, 2023*.
[25] Ouassam, E., Hmina, N., Bouikhalene, B., & Hachimi, H. (2021). *Heuristic methods: Application to complex systems*.
[26] Pérez, C. M. (2017). *Study and benchmarking of modern computing architectures*.
[27] Rayaprolu, R., Randhi, K., & Bandarapu, S. (2024). Intelligent resource management in cloud computing: AI techniques for optimizing DevOps operations. *Journal of Artificial Intelligence and General Studies*, *6*(1).
[28] Shah, M. A., Rajwar, D., Dehury, J. P., & Kumar, D. (2024). VM placement in cloud computing using nature-inspired optimization algorithms. In *Advances in Computer and Electrical Engineering Book Series* (s. 251–282).
[29] Tang, S., Li, S., Tang, B., Wang, X., Xiao, Y., & Cheke, R. (2023). Hormetic and synergistic effects of cancer treatments revealed by modelling combinations of radio- or chemotherapy with immunotherapy. *BMC Cancer*, *23*. https://doi.org/10.1186/s12885-023-11542-6
[30] Wang, H., Peng, T., Brintrup, A., Wuest, T., & Tang, R. (2022). *Dynamic job shop scheduling based on order remaining completion time prediction*.
[31] Wojciuk, M., Swiderska-Chadaj, Z., Siwek, K., & Gertych, A. (2024). Improving classification accuracy of fine-tuned CNN models: Impact of hyperparameter optimization. *Heliyon*, *10*. https://doi.org/10.1016/j.heliyon.2024.e26586
[32] Wu, M., Tao, F., & Cao, Y. (2023). *Value of potential field in reward specification for robotic control via deep reinforcement learning*. AIAA SCITECH 2023 Forum. https://doi.org/10.2514/6.2023-0505
[33] Yan, P., Zhang, J., & Zhang, T. (2024). Nature-inspired approach: A novel rat optimization algorithm for global optimization. *Biomimetics*, *9*(1), 732.
[34] Yang, B., Liu, Y., Li, H., Chen, Y., & Deng, X. (2024). Optimization of edge computing task offloading based on multi-agent game. *Proc. SPIE*.
[35] Zhan, W., et al. (2020). Mobility-aware multi-user offloading optimization for mobile edge computing. *IEEE Transactions on Vehicular Technology*, *69*(6), 6612–6626.
[36] Zhou, S., Sun, J., Xu, K., & Wang, G. (2024). AI-driven data processing and decision optimization in IoT through edge computing and cloud architecture. *Journal of AI-Powered Medical Innovations*, *2*(1), 64–92. https://doi.org/10.60087/vol2iisue1.p006