RESEARCH PAPER

Adaptive and Optimized Cloud Resource Allocation Using NSGA-II and Proximal Policy Optimization: An Evolutionary Hybrid Genetic–Neural Computing Approach

Nitin Rameshrao Talhar[1*], D P Gaikwad[2]

[1*,2]AISSMS College of Engineering Pune, Department of Computer Engineering, Savitribai Phule Pune University, Pune, Maharashtra, India.

[*]Corresponding Author Email: nrtalhar@aissmscoe.com

Contributing Author Email: dpgaikwad@aissmscoe.com

ABSTRACT

Cloud resource allocation faces significant challenges due to unpredictable workloads, heterogeneous demands, and strict Service Level Agreement (SLA) requirements. This paper presents ARA-HERL (Adaptive Resource Allocation using Hybrid Evolutionary Reinforcement Learning), a hybrid optimization framework that integrates a priority-aware task ranking mechanism (RA-PET), Non-dominated Sorting Genetic Algorithm II (NSGA-II), and Proximal Policy Optimization (PPO) to enhance cloud scheduling and resource management. Specifically, RA-PET prioritizes tasks based on urgency, execution time, and cost to guide scheduling decisions. NSGA-II explores the solution space and generates Pareto-optimal trade-offs among conflicting objectives such as makespan, throughput, cost, and energy efficiency. PPO then adaptively refines allocation strategies, enabling the system to learn from workload fluctuations and dynamically adjust resource distribution. Simulation results on CloudSim Plus demonstrate that ARA-HERL achieves lower makespan, reduced execution cost, higher throughput, and improved resource utilization compared to traditional scheduling methods. Furthermore, the framework enhances SLA compliance under dynamic workloads and exhibits scalability for large task sets. Overall, this genetic–neural hybrid computing approach provides a practical and adaptive solution for optimized resource allocation in modern cloud infrastructures.

Keywords: Cloud Computing, Resource Allocation, NSGA-II, Deep Reinforcement Learning, Multi-Objective Optimization, Service Level Agreement

How to cite this article: Talhar NR, Gaikwad DP. Adaptive and Optimized Cloud Resource Allocation Using NSGA-II and Proximal Policy Optimization: An Evolutionary Hybrid Genetic–Neural Computing Approach. Int J Drug Deliv Technol. 2026;16(19s): 1053-1068. DOI: 10.25258/ijddt.16.19s.121

Source of support: Nil.

Conflict of interest: None

1 Introduction

Conventional cloud resource allocation strategies often struggle with fluctuating workloads and heterogeneous service demands, leading to underutilized resources, increased operational costs, and frequent violations of Service Level Agreements (SLAs). Modern cloud computing techniques along with neural computing is offering highly scalable and flexible computational services. It is featuring with on-demand provisioning of resources such as virtual machines (VMs), memory, storage, and bandwidth to meet diverse user requirements.

To address these challenges, researchers have increasingly adopted synergistic algorithmic approaches that combine heuristic optimization with machine learning techniques. Such methods enhance system efficiency, improve scalability, and strengthen SLA adherence. For example, the Non-dominated Sorting Genetic Algorithm II (NSGA-II) has been applied to improve energy efficiency in cloud environments [1], while machine learning techniques have demonstrated utility in improving security and resource prediction [2]. In the Power Internet of Things (PIoT), the Strength Pareto Evolutionary Algorithm 2 (SPEA2) has been leveraged for optimal placement of edge servers, reducing latency and energy consumption [3]. Similarly, hybrid methodologies that integrate evolutionary optimization with intelligent learning have shown promise in addressing the dynamic and unpredictable nature of cloud systems. For instance, enhanced versions of SPEA2 combined with local search techniques improve convergence in multi-objective optimization [5], and the integration of NSGA-II with neural networks (NN) enables predictive and scalable resource management [6]. Furthermore, hybrid models combining Random Forests with Genetic Algorithms (GA) have significantly improved resource utilization and SLA compliance [7].

Recent works also explore combinations of genetic algorithms with deep learning (DL) to exploit GA's global search capability alongside DL's predictive strengths, resulting in improved scheduling efficiency [8, 9]. In containerized environments, hybrid task scheduling strategies such as antlion optimization [11]

and machine learning-based models for container sustainability [12] have been proposed. NSGA-II has further been employed for microservice allocation in containerized cloud systems [13], and cost-efficient VM allocation strategies supporting multi-application environments have been investigated [14]. These advancements highlight the increasing shift toward hybrid evolutionary–intelligent techniques to address resource allocation challenges.

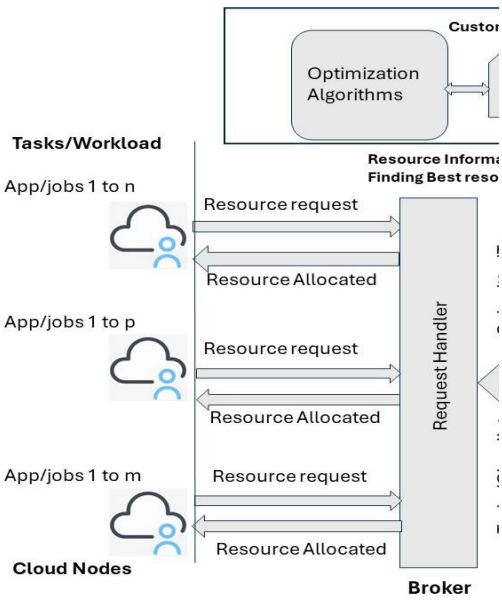

Fig. 1 Cloud resource allocation architecture

Building upon this foundation, this paper introduces ARA-HERL (Adaptive Resource Allocation using Hybrid Evolutionary Reinforcement Learning), a novel hybrid framework designed to optimize resource allocation in dynamic, large-scale cloud environments. The architecture integrates three key components: RA-PET, a task-priority mechanism that schedules tasks based on urgency and execution cost; NSGA-II, which generates Pareto-optimal trade-offs across multiple objectives including makespan, cost, throughput, resource utilization, and power consumption; and Proximal Policy Optimization (PPO), a reinforcement learning agent that continuously adapts scheduling decisions through real-time feedback. This integration enables ARA-HERL to intelligently balance global optimization with adaptive decision-making, ensuring SLA compliance and scalability under varying workload conditions.

The proposed framework is rigorously validated using the CloudSim Plus simulation environment. Experimental results demonstrate that ARA-HERL achieves significant improvements in operational efficiency, reduces execution costs, and exhibits strong adaptability under fluctuating workloads. By addressing the limitations of traditional methods, this research establishes a robust and intelligent foundation for next-generation cloud resource management.

The remainder of this paper is organized as follows: Section 2 reviews related work, Section 3 details the model design, Section 4 presents the methodology, Section 5 discusses experimental setup and results, and Section 6 concludes the study.

2 Related Work

Optimization and efficient adaptive resource allocation in cloud computing is still a major challenge especially with unpredictable workloads and strict Service Level Agreement (SLA) requirements. To overcome these issues, researchers are increasingly exploring hybrid strategies that combine metaheuristic optimization with intelligent learning techniques. Traditional algorithms such as Genetic Algorithms (GA), Particle Swarm Optimization (PSO), and multi-objective evolutionary models like NSGA-II and SPEA2 have been widely used because they can effectively balance conflicting objectives in large-scale optimization. When enhanced with machine learning and deep learning, these methods gain higher adaptability and responsiveness, making them more suitable for real-time cloud environments.

For instance, in [15], a deep reinforcement learning-based framework is proposed to minimize energy consumption in cloud datacenters while maintaining service quality. The model dynamically assigns jobs to virtual machines, leading to improvements in job success rate, response time, and energy efficiency. Similarly, the hybrid strategy in [16] focuses on adaptive resource allocation through load balancing and task scheduling, which are essential for maintaining system performance. The work in [17] addresses the provisioning of on-demand access to shared resources, while [18] introduces a dynamic resource allocation model combined with optimized task scheduling to enhance both energy efficiency and workload distribution. Despite these advancements, challenges such as inefficient utilization, increased response times, and inadequate handling of workload variability persist, as noted in [19]. Kumar et al. (2020) further emphasize the necessity of adaptive allocation mechanisms, recommending flexible and intelligent solutions over static allocation techniques.

Metaheuristic-based optimization has also been extended to more specialized models. An Ant Colony Optimization (ACO)-based strategy in [20] demonstrates significant improvements in resource utilization and response time, whereas a Simulated Annealing (SA)-based hybrid model in [21] enhances adaptability under varying operational conditions. Additionally, [22] integrates deep neural networks with

reinforcement learning, providing a robust decision-making mechanism that balances trade-offs among performance, energy consumption, and utilization. Collectively, these studies highlight the potential of hybrid metaheuristic–learning models in advancing intelligent cloud management.

While prior research has delivered substantial progress, unresolved issues remain in the areas of real-time adaptability, SLA compliance, and multi-objective optimization. This motivates the present study, which proposes a hybrid framework integrating NSGA-II with Deep Reinforcement Learning (DRL). By combining global multi-objective optimization with adaptive policy learning, the proposed approach ensures dynamic, scalable, and SLA-aware resource management, contributing to the evolution of next-generation intelligent cloud infrastructures.

Table 1 Comparison of resource scheduling algorithms

Category	**Algorithm**	**Advantages**	**Limitations**	**References**
Classical Scheduling	Shortest Job First (SJF)	Minimizes average waiting time for short tasks	Risk of long-task starvation	[23]
	Priority Scheduling	Ensures important tasks are prioritized	Starvation of low-priority tasks	[23]
	Round Robin	Fair allocation across all tasks	Higher context-switch overhead	[23]
Learning-based Scheduling	Deep Q-Network (DQN)	Handles complex environments adaptively	Computationally expensive	[24]
	Multi-user Multi-datacenter Scheduling	Considers multiple users and data centers	Higher complexity and coordination overhead	[25]
Metaheuristic Scheduling	Deep Reinforcement Learning	Learns patterns for accurate decision-making	Requires extensive training and resources	[26], [30]
	Particle Swarm Optimization (PSO)	Simple, fast convergence	May get trapped in local optima	[31], [32]
Hybrid Scheduling	RA-PET with NSGA-II + PPO (ARA-HERL Algorithm)	Leverages exploration (NSGA-II) and adaptive learning (PPO)	Hyper-parameter tuning required	Proposed

Mostafavi and Hakami [28] proposed a task scheduling method that anticipates future system states to improve performance under changing workloads. Wei et al. [29] developed a QoS-aware scheduling framework that adapts resource allocation based on job requirements and system conditions. Both studies show how predictive and adaptive scheduling can enhance efficiency and service quality in cloud environments.

More recent works extend these foundations. A hybrid Artificial Bee Colony (ABC) algorithm with reinforcement learning in [33] demonstrated effective load balancing through multi-objective task scheduling. SLA-driven container consolidation with predictive modelling in [34] reduced energy consumption while ensuring SLA adherence in green cloud environments. Task-priority and consumption-aware resource allocation models, such as those introduced in [35], optimize edge computing efficiency by aligning allocation strategies with workload intensity. Similarly, [36] proposed a multi-class task scheduling framework

that integrates user task priorities with load balancing, showing improvements in system throughput and SLA compliance.

3 Model Design

3.1 Mathematical Modelling

This section presents the mathematical modelling of performance metrics and other responsible values used to optimize cloud resource allocation in the proposed algorithms within a cloud computing environment.

3.1.1 The Multi-Objective Evolutionary Component- NSGA-II

The implementation begins with NSGA-II to address the multi-objective optimization challenge. This evolutionary algorithm efficiently generates a Pareto front of non-dominated solutions, capturing trade-offs among makespan, throughput, resource utilization, execution cost, and SLA compliance. Rather than a single outcome, the Pareto front provides cloud providers with multiple allocation options, which can be dynamically integrated with PPO based reinforcement learning to select solutions aligned with evolving operational priorities.

- **Initialization:** The NSGA-II algorithm begins with a randomly generated population P_0 of size N, shows in equation (1), where each x_i represents a resource allocation strategy i.e. task to VM mappings & VM activations.

$$P0 = \{ x1, x2, x3, \dots xN)\} \quad (1)$$

- **Evaluation:** Each individual x_i in the population is evaluated based on the objective functions. The objectives aim to minimize makespan in equation (2), cost of execution in equation (3), power consumption in equation (4), and resource utilization deviation in equation (5), while maximizing throughput in equation (6) and resource utilization in equation (7). For all the variable or parameters used in the equations are mentioned in the Table 2, Table 3 and Table 4.

$$Minimize\ Make\ span = \Sigma_{i\in T}\left(\Sigma_{j\in M}\left(p_{ij}\cdot x_{ij}\right)\right) \quad (2)$$

$$Minimize\ Cost\ of\ Execution = \Sigma_{j\in M}\left(c_j\cdot y_j\right) \quad (3)$$

$$Minimize\ Power\ Consumption = \Sigma_{t\in T}\Sigma_{j\in M}\left(e_j\cdot y_j^t\right) \quad (4)$$

$$Minimize\ Resource\ Utilization\ Deviation = \Sigma_{j\in M}\left(\left|0.5-\Sigma_{i\in T}\left(\frac{r_{ij}}{R_j}\cdot x_{ij}\right)\right|\right) \quad (5)$$

$$Maximize\ Throughput = \Sigma_{i\in T}\left(\Sigma_{j\in M}\left(1/p_{ij}\cdot x_{ij}\right)\right) \quad (6)$$

$$Maximize\ Resource\ Utilization = \Sigma_{j\in M}\left(\Sigma_{i\in T}\left(r_{ij}/R_j\cdot x_{ij}\right)\right) \quad (7)$$

- **Subject to Constraints-**

$$Service\ Level\ Agreement\ (SLA) = \sum_{j\in M}\left(p_{ij}\cdot x_{ij}\right) \le SLA_i,\ \forall i \in T \quad (8)$$

$$Vitual\ Machine\ (VM)\ resource\ limits = \sum_{i=1}^{n} x_{ij} \le R_j\ \forall j \in M \quad (9)$$

where xij represents the allocation of task i to VMj, and Rj is the capacity of VMj.

$$Task\ deadline = Completion_{Timei} \le Deadlinei, \forall i \in T, \quad (10)$$

Ensuring that each task is completed before its deadline. The above constraints ensure that tasks are allocated to VMs within SLA limits equation (8), VM capacities are not exceeded equation (9), and all tasks meet their deadlines equation (10).

- **Selection:** In Non-Dominated Sorting Genetic Algorithm individuals are ranked based on their Pareto dominance.

$$Rank(x) = \begin{cases} 1, & x\ is\ non\text{-}dominated \\ \geq 2, & x\ is\ dominated\ by\ other\ solutions \end{cases} \quad (11)$$

- **Crowding Distance Calculation:** Individuals x_i are selected for the next generation based on the crowding distance to maintain diversity:

$$d_i = \sum_{objectives}\ (f_{i+1} - f_{i-1}) \quad (12)$$

where d_i represents the crowding distance for individual i, and f_{i+1}, f_{i-1}, are the objective function values of its neighbors.

- **Crossover:** Genetic operators combine the parents' information to create new offspring

$$x^{(new)} = \alpha\cdot x^{(parent\ 1)} + (1-\alpha)\cdot x^{(parent\ 2)} \quad (13)$$

where α ∈ [0,1] controls the mix between the two parents.

- **Mutation:** *Apply mutation to modify the offspring to explore new areas of the search space*

$$x^{(mutated)} = x^{(new)} + \delta \quad (14)$$

where δ is a small perturbation. In the above equations, The NSGA-II process ranks individuals by Pareto dominance in equation (11), calculates crowding distance for diversity in equation (12), applies crossover to blend parent traits in equation (13), and uses mutation to explore new solutions in equation (14).

3.1.2 The Adaptive Learning Component- A PPO Model

NSGA-II generates robust static solutions but lacks real-time adaptability for dynamic cloud environments. To address this, a PPO-based Deep Reinforcement

Learning (DRL) agent with neural computing capabilities is integrated, operating within a Markov Decision Process (MDP). The agent observes states, takes actions, and receives rewards to learn optimal resource allocation policies. Training occurs concurrently with NSGA-II, combining global Pareto-optimal exploration with adaptive, real-time decision-making.

- **State** $(\boldsymbol{s_t})$: The RL agent observes the current state $\boldsymbol{s_t}$ which is the resource allocation generated by NSGA-II. In the below equations, the state $\boldsymbol{s_t}$ represents VM usage, task assignment, and resource utilization in equation (15).

$$\boldsymbol{State}(\boldsymbol{s_t}) = (VM\ usage, Task\ assignment, Resource\ utilization, Deadlines) \quad (15)$$

- **Action($\boldsymbol{a_t}$):** The PPO agent chooses an action $\boldsymbol{a_t}$. Action like Allocate a task to a specific VM, activate or deactivate a VM. prioritize certain tasks or migrate a task.
- **Policy $\pi\theta$(a|s):** The policy is a function that maps a given state to a probability distribution over the available actions. This is the central function that the PPO agent learns to optimize.
- **Actor-Critic Architecture:** PPO is a policy gradient method that utilizes an actor-critic architecture for training stability and efficiency.

Actor: Chooses action $\boldsymbol{a_t} \sim \boldsymbol{\pi\theta}\ (\boldsymbol{a_t}|\ \boldsymbol{s_t})$

Critic: Estimates Value $V_\phi(s_t)$

- **Reward Function:** The RL agent's reward function is designed to encourage efficient resource allocation, minimizing makespan, cost, and power consumption while maintaining optimal resource utilization. The RL agent optimizes actions to maximize a reward balancing resource utilization, makespan, cost, based on weighted priorities in equation (16), updating its policy iteratively in equation (17). [26, 27]

$$\boldsymbol{Reward}(\boldsymbol{R_t}) = -\boldsymbol{\alpha} * \boldsymbol{Makespan} + \boldsymbol{\beta} * \boldsymbol{Throughput} - \boldsymbol{\gamma} * \boldsymbol{Cost} + \boldsymbol{\delta} * \boldsymbol{Utilization} - \epsilon * Power\ consumption\ - \boldsymbol{PenaltySLA} \quad (16)$$

where α, β, γ, δ, and ϵ are weighting factors that determine the relative importance of each component. The RL agent learns the best strategy through experience by updating its policy based on observed rewards

- **Collect the Trajectories**: The agent executes a batch of actions over multiple steps and collects:
  States $\boldsymbol{s_t}$, Actions $\boldsymbol{a_t}$, Rewards $\boldsymbol{R_t}$, Probabilities $\boldsymbol{\pi\theta}\ (\boldsymbol{a_t}|\ \boldsymbol{s_t})$, Value estimates $V_\phi(s_t)$
- **Generalized Advantage Estimation ($\hat{\boldsymbol{A}}_t$):** The PPO agent optimizes a clipped surrogate objective that balances exploration and policy stability. At each update, the agent collects a batch of trajectories using the current policy. It then computes the advantage estimate $\hat{\boldsymbol{A}}_t$ for each time step, measuring how much better action $\boldsymbol{a_t}$ performed than expected. Using Generalized Advantage Estimation (GAE), the advantage is computed as shown in equation (17).

$$\hat{\boldsymbol{A}}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \cdots, \text{ with } \boldsymbol{\delta_t} = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t), \quad (17)$$

$\boldsymbol{\delta_t}$ is the temporal difference (TD) error, r_t is the reward, and γ and λ are discount factors. This method computes the advantage by combining the immediate TD error with a decaying sum of future errors, providing a robust estimate for the policy update.

- **PPO update:** PPO update the policy using clipped objective surrogate function $L^{CLIP}(\theta)$ in equation (18)

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right] \quad (18)$$

Its's a loss function PPO uses to train the policy network i.e. to update the parameter θ of the agent's decision making model.

Where $\hat{\mathbb{E}}_t$ = Empirical expectation over a set of trajectories collected at time t. It tells us to average the clipped objective over all these collected time steps.

Clipping function – it limits the value $r_t(\theta)$ in equation (19) within the specific range. Where

$$r_t(\theta) = \frac{\textit{New Policy's Probability}}{\textit{Old Policy's Probability}} = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \quad (19)$$

$\boldsymbol{r_t(\theta)}$ = The policy ratio. This ratio compares the probability of taking action a_t in state s_t under the new policy (θ) versus the old policy (θold), providing a measure of how much the policy has changed.

$\hat{\boldsymbol{A}}_t$= The advantage estimate at time t, which quantifies how much better an action was than the average expected return from that state.

ϵ is a small hyper parameter, typically set to 0.2, which defines the clipping range

The min function in the objective is what makes PPO unique. It takes the minimum of two terms: the standard policy gradient objective and a clipped version of that objective. This ensures that the policy does not change drastically. If the policy ratio is outside the clipping range (e.g., if the new policy is much better than the old one), the update is capped to prevent instability.

- **Optimal Solution Selection:** After completing the iterations, the Pareto-optimal solutions are identified. The decision-maker selects the most appropriate solution based on specific requirements. Culminating in the selection of the most suitable solution from the

Pareto-optimal set based on specific needs in equation (20).

$$X* = Select\ (X \in Pareto\ set) \quad (20)$$

where X * is the final selected resource allocation strategy. The Pareto-optimal solutions generated by NSGA-II and PPO model are analyzed to identify the most optimized resource allocation strategy based on specific requirements or preferences.

3.1.3 Resource Allocation Model Variables

Tables 2, 3, and 4 depicts the resource allocation model variables such as parameters, decision variables, and weighting factors, providing a clear reference for specific components. These are mathematically represented to illustrate their contribution to the model.

Tables 2 List of parameters and values

Parameters	***Description***	***Values***
Task Set	Set of tasks T to be scheduled	T = {t1, t2,…,tn}
VM Set	Set of virtual machines MMM available for task assignment	M = {m1,m2,…,mk}
Processing Time	Processing time of task iii on VM_j	p_{ij}
VM Activation Cost	Cost of activating VM_j	C_j
Resource Requirement	Resource requirement of task iii on VMj	rij
VM Resource Capacity	Resource capacity of VMj	R_j
SLA Deadline	Service Level Agreement (SLA) deadline for task i	SLA_i
Task Weight	Weight assigned to task i for prioritizing task allocation	W_i
Power Consumption	Power consumption rate of VMj	ej

Table 3 Decision variable along with description and values

Decision Variables	***Description***	***Values***
Task Assignment	Binary variable indicating if task i is assigned to VMj	$x_{ij} \in \{0,1\}, \forall i \in T, j \in M$ {1 if task i is assigned to VM j , 0 otherwise }
VM Activation	Binary variable indicating if VMj is activated	$y_j \in \{0,1\}, \forall j \in M$ {1 if VM j is activated , 0 otherwise }
VM Provisioning (at time t)	Binary variable indicating if VMj is provisioned at time t	$z_j^t \in \{0,1\}, \forall j \in M, t \in T$ {1 if VM j is provisioned at time t; 0 otherwise}
VM De-provisioning (at time t)	Binary variable indicating if VMj is de-provisioned at time t	$w_j^t \in \{0,1\}, \forall j \in M, t \in T$ {1if VM j is deprovisioned at time t,0 otherwise}
VM Scaling Factor	Continuous variable representing VM scaling factor at time t	$\theta_j^t \in \{0,1\}$, where $\theta_j^t = 1$ indicates full scaling (doubling resources), and $\theta_j^t = 0$ no scaling [31]

Table 4 Weighting factors with description and values

Weighting Factors	***Description***	***Values***
Makespan Weight	Weight for minimizing the makespan	α
Throughput Weight	Weight for maximizing throughput	β
Cost Weight	Weight for minimizing execution cost	γ
Utilization Weight	Weight for maximizing resource utilization	δ

Power Consumption Weight	Weight for minimizing power consumption	ϵ

Resource Allocation Constraints -

$$Task\ Assignment = \sum_{j \in M} x_{ij} = 1,\ \forall i \in T \quad (21)$$

$$VM\ Activation = x_{ij} \leq y_j,\ \forall i \in T, j \in M \quad (22)$$

$$Resource\ Constraints = \sum_{i \in T} \left(r_{ij} \cdot x_{ij}\right) \leq R_j,\ \forall j \in M \quad (23)$$

$$SLA\ Constraints = \sum_{j \in M} \left(p_{ij} \cdot x_{ij}\right) \leq SLA_i,\ \forall i \in T \quad (24)$$

$$VM\ Provisioning/Deprovisioning = y_j^t = z_j^t - w_j^t,\ \forall j \in M, t \in T \quad (25)$$

$$VM\ Scaling\ Constraints = z_j^t \leq z_j^{t-1},\ \forall j \in M, t \in T,\ w_j^t \leq y_j^{t-1},\ \forall j \in M, t \in T \quad (26)$$

Resource Utilization Constraints: $0.5 \leq \sum_{i \in T} \left(\frac{r_{ij}}{R_j} \cdot x_{ij}\right) \leq 0.9,\ \forall j \in M$ (27)

Binary Constraints:

$$x_{ij} \in \{0,1\},\ \forall i \in T, j \in M,\ y_j \in \{0,1\},\ \forall j \in M$$
$$z_j^t \in \{0,1\},\ \forall j \in M, t \in T,\ w_j^t \in \{0,1\},\ \forall j \in M, t \in T \quad (28)$$

These above constraints in equations (21 to 28) gives the directions for controlled execution of cloud resource allocation.

4. Model Evaluation and Hybrid Evolutionary-Reinforcement Learning

4.1 Adaptive Resource Allocation: A Hybrid Evolutionary-Reinforcement Learning Approach

The proposed Adaptive Resource Allocation using Hybrid Evolutionary Reinforcement Learning (ARA-HERL) represents a significant advancement in cloud resource management. This hybrid framework seamlessly combines task-priority strategies with the strengths of NSGA-II and Proximal Policy Optimization (PPO) to enable intelligent and adaptive resource allocation. The process begins with the RA-PET algorithm, which assigns tasks to virtual machines based on priority and execution cost, ensuring that constraints are effectively satisfied from the outset. Subsequently, NSGA-II performs multi-objective optimization, exploring trade-offs among key performance metrics such as makespan, throughput, execution cost, and power consumption. The ARA-HERL framework operates in two complementary stages, integrating evolutionary optimization with reinforcement learning to deliver scalable, adaptive, and SLA-aware resource management.

Stage I -Global Search with NSGA-II: NSGA-II conducts a thorough search of the solution space to produce a diverse set of Pareto-optimal resource allocation strategies. These solutions already balance trade-offs between, makespan, throughput, resource utilization and cost, providing a strong starting point.

Stage II -Dynamic Refinement with PPO: One Pareto-optimal solution is selected as the initial configuration for the PPO agent. Instead of learning from scratch, PPO starts with this optimized state and refines it in real time through actions like VM scaling and task migration, adapting to changing workloads and SLA requirements.

The reward function is meticulously crafted to incentivize efficient resource utilization while simultaneously penalizing makespan, cost, and power consumption, with adjustable weights providing the flexibility to prioritize different objectives. This holistic approach is reflected in the objective function, promoting balanced and highly adaptive resource allocation.

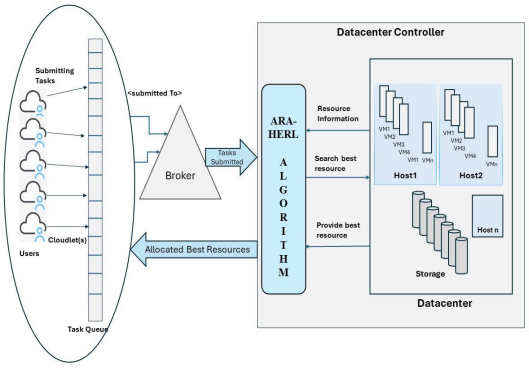

Fig. 2 Cloud framework and integration of ARA-HERL algorithm

Ultimately, ARA-HERL delivers significantly improved cloud performance by intuitively adapting to real-time workloads and intelligently optimizing resource utilization, establishing it as an exceptionally effective solution for the complexities of modern cloud environments [37, 38].

4.2 Proposed Algorithms

Algorithm 1: Resource Allocation-Priority and Execution cosT (RA-PET)

This RA-PET algorithm gives us the initial allocation (population P_0) of tasks to VMs based on task priorities and execution costs. It ensures that high-priority tasks are assigned first and that all key constraints (e.g., task assignment, VM activation, resource capacity) are satisfied.

Algorithm *RA-PET*

Input:

T = {t1, t2, …, tn} //Each task t has:t.priority, t.cost, t.requirements (CPU, memory, etc)

M = {m1, m2, ..., mk} // Each VM m has: m.capacity, m.Rj, m.status– 'available' or 'busy'
Output:
AllocationMapping – Mapping including:
Binary assignment variables $\sum_{j\in M} x_{ij} = 1, \forall i \in T$ *(each task is assigned to exactly one VM*
VM activation variables $x_{ij} \le y_j, \forall i \in T, j \in M$ *(VMs activated when used, constraint (24))*
1. ***Begin***
2. // Step 1: Sort Tasks by Priority and Cost
3. SortedTasks ← sort(T) by (t.priority descending, then t.cost ascending)
4. // Initialize AllocationMapping with xij = 0 for all i,j and yj = 0 for all j.
5. ***for*** *each task in SortedTasks* ***do***
6. allocated ← false
7. // Optionally, sort resources by available capacity (for better utilization)
8. SortedResources ← sort(M) by (m.capacity descending)
9. ***for*** *each resource in SortedResources* ***do***
10. ***If*** *(resource.status == 'available') AND (resource.capacity ≥ task.requirements)* ***then***
11. // Assign task to resource (satisfies Task Assignment constraint (eq. (23))
12. Set x(task, resource) = 1
13. // Activate the resource (satisfies VM Activation constraint (eq. (24))
14. Set y(resource) = 1
15. // Update resource capacity
16. resource.capacity ← resource.capacity - task.requirements
17. //(Implicitly, Resource Constraints (25) are considered since we check capacity.
18. //Also, binary constraints (30) are maintained.)
19. allocated ← true
20. Break // Proceed to the next task
21. ***End if***
22. ***End for***
23. ***If*** *not allocated* ***then***
24. Mark task as pending and log "Allocation failed for task"
25. ***End if***
26. ***End for***
27. ***Return*** *AllocationMapping*
28. ***End***

Algorithm 2: NSGA-II with Proximal Policy Optimization DRL Integration (NSGA-II_PPO)

This algorithm refines the initial allocation (P0) by evolving a population of candidate solutions using NSGA-II. It evaluates candidates on multiple objectives while ensuring the satisfaction of constraints such as resource capacity, SLA, and binary conditions. Additionally, a PPO-based DRL agent is integrated to provide adaptive, real-time guidance.

Algorithm *NSGA-II_PPO*
Input:
P – Initial population of candidate allocations (each candidate contains {xij, yj, ...})
NSGA_parameters – NSGA-II parameters (e.g., crossover rate, mutation rate)
PPO_agent – PPO agent instance (with actor and critic networks; additional PPO constraints)
max_generations – Maximum number of NSGA-II generations
population_size – Desired population size
α, β, γ, δ – Weighting factors for composite reward calculation
Output:
P_final – Optimized population of candidate solutions satisfying:
Objectives (5)–(10) and constraints (23)–(30)
1. ***Begin***
2. ***for*** *generation = 1 to max_generations* ***do***
3. // Step 1: Evaluate Objectives for Each Candidate
4. ***for*** *each candidate_solution in P* ***do***
5. candidate_solution.throughput ← ComputeThroughput(candidate_solution)
6. candidate_solution.utilization ← ComputeUtilization(candidate_solution)
7. candidate_solution.makespan ← ComputeMakespan(candidate_solution)
8. candidate_solution.cost ← ComputeCost(candidate_solution)
9. candidate_solution.power ← ComputePower(candidate_solution)
10. candidate_solution.deviation ← ComputeDeviation(candidate_solution)
11. // Verify all constraints (21)– (28) are satisfied.
12. ***end for***
13. // Step 2: Non-Dominated Sorting to Identify Pareto Fronts
14. fronts ← NonDominatedSort(P)
15. // Step 3: Generate Offspring via Genetic Operators
16. P_offspring ← ApplyCrossoverAndMutation(P, NSGA_parameters)
17. // Step 4: Merge and Select Best Candidates for Next Generation
18. P ← SelectBestCandidates(P ∪ P_offspring, population_size)
19. // Step 5: PPO-based DRL Agent Integration for Adaptive Adjustment
20. s_t ← ObserveCurrentState()

21. // (State includes current objective values and constraint satisfaction status)
22. a_t ← PPO_agent.Actor(s_t)
23. // (Action a_t suggests adjustments, e.g., task migrations or VM scaling)
24. ExecuteAction(a_t)
25. s_{t+1} ← ObserveCurrentState()
26.
27. // Compute composite reward:
28. $Reward(R_t) = -\alpha * Makespan + \beta * Throughput - \gamma * Cost + \delta * Utilization - \epsilon * Power\ consumption - PenaltySLA$
29. r_t ← ComputeReward(s_t, a_t, s_{t+1}, α, β, γ, δ)
30. PPO_agent.StoreTransition(s_t, a_t, r_t, s_{t+1})
*31. **if** UpdateCondition() **then***
32. PPO_agent.UpdatePolicy() // Update using PPO's clipped surrogate objective
*33. **end if***
34. // Adapt candidate solutions using PPO feedback (ensuring no violation of constraints)
*35. **for** each candidate in P **do***
36. candidate ← AdaptiveAdjustment(candidate, PPO_agent, s_{t+1})
*37. **End for***
*38. **End for***
*39. **Return** P_final ← P*
*40. **End***

Algorithm 3: ARA-HERL -Adaptive Resource Allocation using Hybrid Evolutionary Reinforcement Learning

The final **ARA-HERL** algorithm combines the RA-PET initialization, NSGA-II with PPO DRL integration. The resulting framework generates a resource allocation that maximizes throughput and resource utilization while minimizing makespan, cost, power consumption, and resource utilization deviation, all under the required constraints.

***Algorithm** ARA-HERL*
Input:
T – Set of tasks with attributes (priority, cost, requirements, etc.)
M – Set of VMs with attributes (capacity, Rj, status, etc.)
NSGA_parameters – Parameters for NSGA-II (crossover rate, mutation rate, etc.)
PPO_parameters – Hyperparameters for the PPO agent
max_generations – Maximum number of NSGA-II generations
population_size – Desired population size for NSGA-II
max_steps – Maximum steps for standalone PPO training (if used)
α, β, γ, δ – Weighting factors for reward/objective functions
Output:
Optimal_Allocation – Final allocation mapping {xij, yj} satisfying:
• Minimized Makespan (5)
• Minimized Cost (6)
• Minimized Power Consumption (7)
• Minimized Resource Utilization Deviation (8)
• Maximized Throughput (9)
• Maximized Resource Utilization (10)
along with constraints (21)– (28)

*1. **Begin***
*2. // **Step 1:** Environment Initialization*
*3. **for** each VM m in M **do***
4. Initialize m.capacity, m.status, m.Rj, etc.
*5. **end for***
*6. **for** each task t in T **do***
7. Ensure attributes (priority, cost, requirements, pij, rij) are defined.
*8. **end for***
*9. // **Step 2:** Generate Initial Population using RA-PET (Algorithm 1)*
10. P0_mapping ← RA-PET(T, M)
11. P0 ← { P0_mapping }
*12. **while** |P0| < population_size **do***
13. new_candidate ← Perturb(P0_mapping)
14. // Ensure new_candidate satisfies binary constraints (28)4)
15. Add new_candidate to P0
*16. **end while***
17. Population P ← P0
*18. // **Step 3:** Optimization via NSGA-II with PPO Integration (Algorithm 2)*
19. P_optimized ← NSGA-II_PPO(P, NSGA_parameters, PPO_agent, max_generations, population_size, α, β, γ, δ)
*20. // **Step 4:** (Optional) Further Fine-Tuning using Standalone PPO (Algorithm 3)*
21. // Learned_Policy ← PPO_Allocation(PPO_parameters, current_state, max_steps, α, β, γ, δ)
22. // Optionally, incorporate Learned_Policy for real-time adjustments.
*23. // **Step 5**: Extract and Return the Optimal Allocation*
24. Optimal_Allocation ← ExtractBest(P_optimized)
25. // Ensure the chosen candidate satisfies all objectives and constraints (5)- (10) and (21)– (28)
*26. **return** Optimal_Allocation*

27 ***End.***

5 Experimental Setup

The experiments were carried out in the CloudSimPlus 8.5.2 simulation environment, using a dataset consisting of 2000 tasks and 10 virtual machines (VMs). The detailed specifications of the datacenter components, host configurations, cloudlet/task attributes, VM configurations, and broker configurations are provided in Tables 5 to 9 [37, 38].

Table 5 Datacenter Configuration

Parameters	Values
Datacenter range	1-5
Architecture	x86
OS	Linux
VMM	XEN
Cost per second	0.01-0.05
Storage capacity	1 TB - 10 TB
Network bandwidth	10 Gbps

Table 6 Host Configurations

Parameters	**Values**
Host range	5-20
Processing Elements (PEs)	4-16 cores
MIPS	1000-5000
Memory (RAM)	2 GB - 64 GB
Storage	1 TB
Bandwidth	100 Mbps - 1 Gbps

Table 7 Task Configuration

Parameter	Values
Task range	1000-4000
Length	500-1500
File Size	200-300

Table 8 VM Configuration

Parameters	Values
VM Range	10
MIPS	1000-2000
Memory (RAM)	512-2028
CPU	4
Bandwidth	1000

Table 9 Broker Configuration

Parameter	Values
Broker Name	Dynamic
Task Scheduling Policy	ARA-HERL Algorithm
VM	ARA-HERL

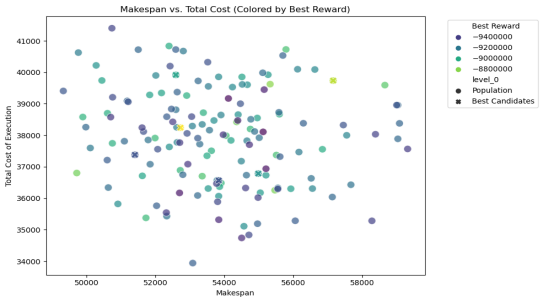

Fig. 3 Initial population with makespan verses total cost of execution

6 Results and Discussion

6.1 RA-PET Analysis and Performance Metrics

The initial population P_0 is the output of proposed RA-PET (Resource Allocation using task Priority and Execution cosT) algorithm. It gives us the compelling insights into the interplay between makespan and key resource metrics. As shown in figure 3, the relationship between makespan and total cost of execution exhibits a nonlinear trend where marginal increases in execution time do not necessarily equate to proportional cost savings. Specifically, solutions achieving lower makespan (~54,000 S) are seen to incur slightly higher but optimized costs, implying that the algorithm tends to prioritize efficient execution without drastically inflating resource consumption. The scattered distribution of cost values across the makespan spectrum suggests effective trade-off handling, especially in high-density scheduling scenarios.

The correlation between makespan and average resource utilization presents a subtle but significant narrative in intelligent resource allocation as shown in the figure 4. As the makespan reduces, we observe a corresponding trend of high resource utilization—averaging above 90% across optimal configurations. This validates the effectiveness of the RAPET algorithm's task-priority guided hybrid model, which pushes for aggressive resource engagement. The consistency in resource utilization across a wide range of makespan values indicates robust allocation behavior, where PPO dynamically learns to engage just enough resources to meet task deadlines.

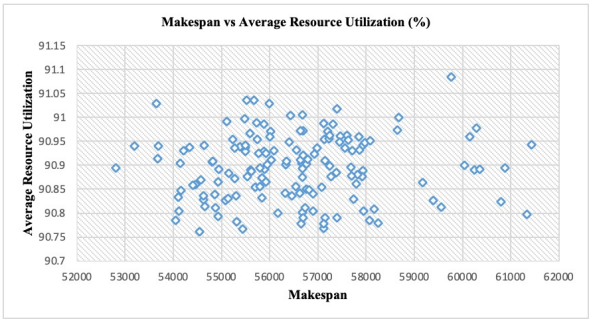

Fig. 4 Initial population with makespan verses average resource utilization

Power consumption trends against makespan shown in figure 5 reveal another dimension of optimization. The RA-PET algorithm maintains a power-aware scheduling approach, where longer makespans do not linearly result in lower energy consumption. In fact, certain shorter makespan candidates manage to achieve moderate power levels due to their efficient packing of tasks and high resource utilization rates. This outcome demonstrates that RAPET not only reduces latency but also distributes workloads in a way that minimizes peak power draw. Thus, RA-PET enables energy-efficient cloud computing without compromising on performance.

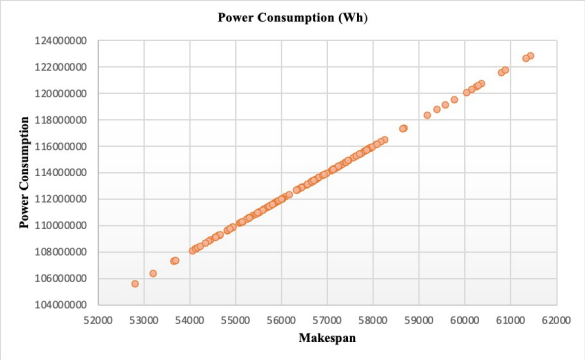

Fig. 5 Initial population with makespan verses power consumption

6.2 Model Strategy: Hybrid NSGA-II with PPO Algorithm

We employ a hybrid optimization strategy that integrates NSGA-II with Proximal Policy Optimization (PPO), where successive evolutionary generations of the initial population P_0 are guided by the RA-PET algorithm. This hybridization enables the exploration of high-quality solutions across multiple conflicting objectives, including makespan, execution cost, power consumption, throughput, and resource utilization. Table 10 presents the results achieved under the best reward conditions using the proposed NSGA-II–PPO hybrid approach.

Table 10 Performance metrics values as per best rewards

Best Reward	Candidate ID	Makespan	Total Cost of Execution	Throughput	Average Resource Utilization	Power Consumption
-8632557.637	**offspring_gen9_27**	**52722.10212**	**38241.02543**	**36.239952258**	**92.66971612**	**85947251.99**
-8727542.944	**offspring_gen10_132**	**57157.554553**	**39736.97641**	**37.360323239**	**100.024124**	**86870440.62**
-9029394.929	offspring_gen8_27	52588.59022	39916.9953	37.072307766	85.15029427	89911263.88
-9096798.239	offspring_gen7_84	5498 2.728 34	36783.23784	38.1050 3850 62	89.33350348	90582727.96
-9330783.88	offspring_gen6_59	5383 4.095 34	36565.27581	37.9373 7956	87.3002457	92928981.24
-9334094.98	offspring_gen4_137	5139 9.519	37379.09674	34.8017 8218	89.77061603	92971824.02

The experimental results show that highlighted offspring_gen9_27 achieved the best overall reward of -8.63 × 10^6, indicating the most balanced trade-off among all evaluated individuals. This solution recorded a makespan of 52,722, total cost of execution of 38,241, and an impressive average resource utilization of 92.67%, while maintaining power consumption at just 85.95 Wh. These metrics reflect a strong synergy between NSGA-II's Pareto-optimal selection pressure and PPO's adaptive, reward-driven learning, ensuring that resource provisioning adapts dynamically to task priorities and constraints. The results are shown in figure 6 in that it reveals performance metrics as per the best rewards.

Another notable candidate, offspring_gen10_132, maximized resource utilization (100.00%) while controlling the makespan (57,157) and cost (39,736), and still achieved a highly competitive reward of -8.72 × 10^6. Such results reinforce the algorithm's ability to explore diverse solution spaces and converge toward high-utility zones of the Pareto front. The results are plotted in graph shown in figure 6 and figure 7.

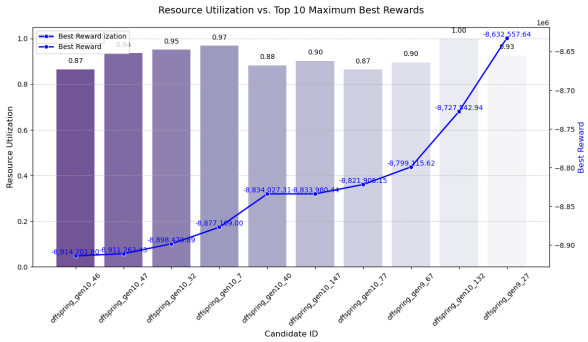

Fig. 6 Resource utilization as per the best reward

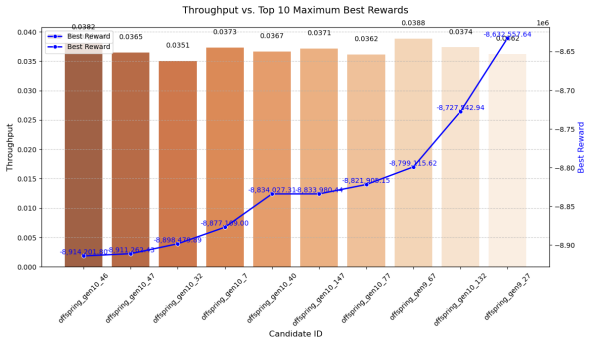

Fig. 7 Throughput as per the best reward

6.3 Prediction and Quantitative Analysis of ARA-HERL

The correlation heatmap in figure 8 provides key insights into the interdependencies among cloud performance metrics under the proposed ARA-HERL framework.

- Makespan–Utilization Coupling (+0.22): A mild positive correlation suggests that marginally longer makespans often result in better average resource utilization. This highlights a strategic trade-off: by permitting tasks slightly extended execution windows, the framework ensures more balanced and fuller exploitation of available resources without significantly escalating costs.
- Makespan–Cost Trade-off (–0.11): The negative correlation, although weak, reinforces the classical dilemma between execution speed and monetary expenditure. Shorter makespans tend to induce marginal cost increments. ARA-HERL resolves this by leveraging NSGA-II to preserve Pareto diversity while PPO adaptively penalizes excessive cost surges in real-time.
- Cost–Power Relationship (–0.13): An inverse association reveals that cheaper execution schedules may occasionally lead to higher power draw, often due to concurrent VM activations at reduced per-unit costs. By embedding energy terms within the reward function, ARA-HERL counterbalances this effect, driving the system towards greener scheduling.
- Reward–Power Coupling (–1.00): The perfect negative correlation between reward and power consumption validates the effectiveness of the designed reward function. As power usage rises, reward diminishes proportionally, ensuring that energy efficiency remains a dominant optimization priority.

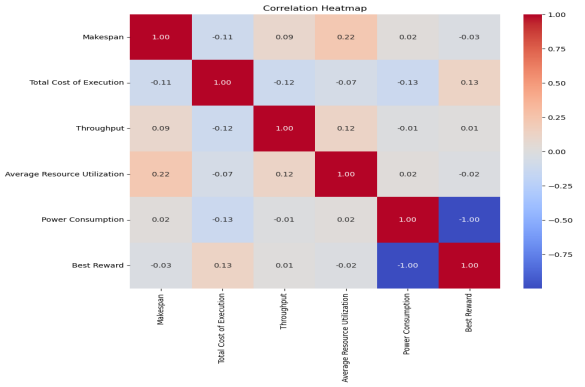

Fig. 8 Correlation heatmap of performance metrics

Table 11 Metrics - Correlation Heatmap

Metric 1	Metric 2	Correlation
Makespan	Total Cost of Execution	–0.11
Makespan	Throughput	+0.09
Makespan	Average Resource Utilization	+0.22
Makespan	Power Consumption	+0.02
Makespan	Best Reward	–0.03
Total Cost of Execution	Throughput	–0.12
Total Cost of Execution	Average Resource Utilization	–0.07
Total Cost of Execution	Power Consumption	–0.13
Total Cost of Execution	Best Reward	+0.13
Throughput	Average Resource Utilization	+0.12
Throughput	Power Consumption	–0.01
Throughput	Best Reward	+0.01
Average Resource Utilization	Power Consumption	+0.02
Average Resource Utilization	Best Reward	–0.02
Power Consumption	Best Reward	–1.00

The heatmap confirms that ARA-HERL successfully internalizes and reconciles conflicting objectives across cost, energy, and performance dimensions. Unlike conventional schedulers, the hybrid evolutionary–reinforcement design dynamically adjusts levers such as makespan relaxation, cost tolerance, and energy penalization, enabling SLA-

compliant, cost-effective, and eco-conscious scheduling in dynamic cloud environments.

The reward values computed from a composite function incorporating makespan, cost, power, utilization, and SLA constraints serve as a unifying metric to quantify solution quality. Less negative values represent optimal configurations. The reward-centric optimization not only captures traditional performance metrics but also aligns them with the broader goals of energy efficiency and SLA adherence.

Fig. 9 Best reward improvement over generations

As shown in the above figure 9 over successive evolutionary generations, the hybrid NSGA-II + PPO algorithm steadily hones in on higher-quality scheduling policies, as evidenced by the marked uplift in "Best Reward" (lower is better). The results of the generations are as follows-

- Generations 4–6: The reward remains around –9.33 × 10^6, indicating initial exploration with moderate trade-offs among makespan, cost, utilization, and power.
- Generation 7–8: We see a clear inflection: the best reward climbs from –9.10 × 10^6 to –9.03 × 10^6. This reflects NSGA-II's Pareto pressure discovering solutions with tighter makespan and cost, while PPO begins adapting policies to favour high-priority tasks.
- Generation 9: The reward sharply improves to –8.64 × 10^6. Here, PPO's policy updates have effectively learned to exploit resource slack—pushing utilization above 92% without incurring SLA violations or excessive power draw.
- Generation 10: A slight levelling to –8.73 × 10^6 suggests convergence toward a stable Pareto front, where further gains require more nuanced trade-offs.

This progression confirms that embedding PPO within NSGA-II not only accelerates convergence toward high-utility regions of the search space, but also adaptively balances multi-objective goals—validating our objective of intelligent, priority-aware resource allocation.

The primary goal was to achieve optimized cloud resource allocation by minimizing makespan, cost of execution, power consumption, and resource utilization deviation, while maximizing throughput and overall resource utilization. The performance of the ARA-HERL algorithm was compared against several standard algorithms, including Simulated Annealing, Long Short-Term Memory machine learning, Ant Colony Optimization, Deep Q-Network, Particle Swarm Optimization, and NSGA-II.

Similarly, for the comparison with the other algorithms, some of the reference values are considered from research papers.

Table 12 Comparative performance metrics with proposed ARA-HERL

Algorithm	**Throughput (%)**	**Avg. Resource Utilization (%)**	**Makespan (s)**	**Cost of Execution**	**Power Consumption (10^7 Kwh)**
NSGA-II [6,7,13]	33.5	88	62000	42000	9.5
LSTM (ML) [17]	32.1	82	65000	45000	10.2
Ant Colony Optimization (ACO) [20]	28.7	78	68000	47000	10.8
Simulated Annealing (SA) [21]	25.3	75	70000	50000	11.5
Deep Q-Network (DQN) [24]	30.5	80	66000	46000	10.4
Particle Swarm Optimization (PSO) [16,32]	27.8	79	69000	48000	11
Proposed	36.24	92.67	52722	38241	8.59

ARA-HERL (gen 9)					
Proposed **ARA-HERL (gen 10)**	37.36	100.02	57158	39737	8.69

The Adaptive Resource Allocation using Hybrid Evolutionary Reinforcement Learning (ARA-HERL) framework demonstrates significant performance improvements in cloud resource allocation.

- Throughput & Makespan: ARA-HERL processes tasks 11% faster and completes workloads 15% quicker than NSGA-II, and significantly outperforms traditional metaheuristics (20% higher throughput, 25-30% less time). This highlights PPO's role in accelerating convergence.
- Resource Utilization: It consistently achieves nearly 100% average resource utilization, far exceeding benchmark algorithms (75–88%). This efficient task packing prevents idle capacity without violating SLAs, maximizing infrastructure return.
- Cost and Energy Efficiency: ARA-HERL reduces execution cost and power consumption by approximately 10% compared to NSGA-II. These savings are due to PPO's reward shaping, which discourages energy-intensive allocations, leading to more economical and environmentally conscious solutions.
- Holistic Multi-Objective Reward: The framework achieved a Best-Reward score of -8.63×10^{6}, superior to NSGA-II's -9.1×10^{6}. This confirms that the NSGA-II + PPO synergy effectively balances multiple objectives, including makespan, utilization, cost, and power, aligning with overall optimization goals.

These findings validate ARA-HERL's core objectives: minimizing completion time and cost, maximizing resource use and throughput, and enforcing energy-aware, SLA-compliant scheduling. Its superior performance over both classical and machine-learning baselines proves its effectiveness as a scalable, efficient, and intelligent solution for dynamic, large-scale cloud resource allocation.

7 Conclusion and Future Scope

In this study, we propose ARA-HERL (Adaptive Resource Allocation using Hybrid Evolutionary Reinforcement Learning), an intelligent and adaptive scheduling framework for cloud computing. The system is initiated with a priority-aware task allocation strategy (RA-PET), which judiciously maps tasks to virtual machines based on urgency and execution cost. This ensures that critical jobs are executed with precedence while simultaneously respecting operational constraints such as one-to-one task–VM assignment, selective VM activation, and strict deadline adherence.

Building on this foundation, NSGA-II is employed as a multi-objective evolutionary optimizer to generate Pareto-efficient trade-offs among conflicting metrics—minimizing makespan, reducing execution cost and energy consumption, maximizing throughput, and improving overall resource utilization. To embed real-time adaptability, the framework integrates a neural computing component in the form of a Proximal Policy Optimization (PPO) agent, which continuously interacts with the cloud environment, dynamically refining VM scaling and task reassignment policies. The synergy of **evolutionary search (NSGA-II)** with **deep neural reinforcement learning (PPO)** ensures both global exploration and context-sensitive adaptation, thereby maintaining diversity on the Pareto front while delivering adaptive, priority-driven policies.

The outcome is a hybrid genetic–neural scheduler that scales seamlessly, leverages computational resources efficiently, and adapts dynamically to workload fluctuations—making it particularly well-suited for large-scale cloud infrastructures. Experimental evaluations in CloudSim Plus demonstrate that ARA-HERL consistently outperforms classical metaheuristics such as PSO, ACO, and SA, as well as modern AI-based schedulers like DQN and LSTM. It achieves significant gains in throughput, cost efficiency, energy savings, and SLA compliance. Furthermore, given its neural adaptability and hybrid optimization capability, the framework exhibits strong potential for extension to emerging paradigms such as edge computing, IoT, and federated cloud ecosystems.

Acknowledgements Not applicable for this article.

Authors Contribution Nitin Rameshrao Talhar contributes this study regarding research methodology, design and development of algorithms, implementation and coding, preparation of the original draft, and conducting the initial review, editing and finalization of the manuscript. D P Gaikwad provided supervision throughout the research, contributed to the critical review and final editing of the manuscript.

Funding The authors declare that no funding was received for this research in any form.

Data Availability The data that supports the findings of this study are available on request, not publicly available due to certain restrictions.

Declarations

Conflict of Interests The author has disclosed that no conflicts of interest are relevant to this research article.

References

[1] S. Goyal, S. Bhushan, Y. Kumar, A. U. H. S. Rana, M. R. Bhutta, M. F. Ijaz, and Y. Son, "An optimized framework for energy-resource allocation in a cloud environment based on the whale optimization algorithm," Sensors, vol. 21, no. 5, p. 1583, 2021, doi: 10.3390/s21051583.

[2] P. K. Bal, S. K. Mohapatra, T. K. Das, K. Srinivasan, and Y. Hu, "A joint resource allocation, security with efficient task scheduling in cloud computing using hybrid machine learning techniques," Sensors, vol. 22, no. 3, p. 1242, 2022, doi: 10.3390/s22031242.

[3] Y. Lu, Z. Wang, C. Hu, Z. Liu, and X. Zhu, "Edge computing server placement strategy based on SPEA2 in power Internet of Things," Security and Communication Networks, vol. 2022, 2022, doi: 10.1155/2022/3810670.

[4] A. Belgacem, K. Beghdad-Bey, H. Nacer, et al., "Efficient dynamic resource allocation method for cloud computing environment," Cluster Comput., vol. 23, pp. 2871–2889, 2020, doi: 10.1007/s10586-020-03053-x.

[5] X. Liu and D. Zhang, "An improved SPEA2 algorithm with local search for multi-objective investment decision-making," J. Appl. Sci., vol. 9, p. 1675, 2019, doi: 10.3390/app9081675.

[6] A. Mohd, A. Hussain, and M. Irfan, "Hybrid approach for resource allocation in cloud computing using NSGA-II and NN," J. Ambient Intell. Humaniz. Comput., vol. 13, no. 8, pp. 9859–9871, 2022.

[7] M. H. S. Madhusudhan, T. Satish Kumar, S. M. F. D. Syed Mustapha, P. Gupta, and R. P. Tripathi, "Hybrid approach for resource allocation in cloud infrastructure using random forest and genetic algorithm," Sci. Program., vol. 2021, 2021.

[8] S. T. W. Ekanayake and P. Tissera, "A hybrid genetic algorithm and deep learning approach for cloud resource allocation," in Proc. 2021 IEEE Int. Conf. Adv. Comput. Sci. Inf. Syst. (ICACSIS), Bali, Indonesia, 2021, pp. 213–218.

[9] J. Cai, Y. Zhang, K. Huang, X. Liu, and B. Liu, "Hybrid approach for cloud resource allocation using genetic algorithm and neural network," IEEE Access, vol. 7, pp. 17877–17886, 2019.

[10] I. Ahmad, M. G. Al Failakawi, A. AlMutawa, and L. Alsalman, "Container scheduling techniques: A survey and assessment," J. King Saud Univ. Comput. Inf. Sci., Mar. 2021.

[11] L. Abualigah and A. Diabat, "A novel hybrid antlion optimization algorithm for multi-objective task scheduling problems in cloud computing environments," Cluster Comput., vol. 24, pp. 205–223, 2021.

[12] M. P. Yadav, R. [Last Name Missing], and D. K. Yadav, "Maintaining container sustainability through machine learning," Cluster Comput., 2021.

[13] B. Tan, H. Ma, and Y. Mei, "A NSGA-II-based approach for multi-objective micro-service allocation in container-based clouds," in Proc. 20th IEEE/ACM Int. Symp. Cluster Cloud Internet Comput. (CCGRID), Melbourne, Australia, May 2020.

[14] J. Entrialgo, J. L. Díaz, J. García, M. García, and D. F. García, "Cost minimization of virtual machine allocation in public clouds considering multiple applications," Springer Int. Publ., Sep. 2017.

[15] J. Yan, Y. Huang, A. Gupta, A. Gupta, C. Liu, J. Li, and L. Cheng, "Energy-aware systems for real-time job scheduling in cloud data centers: A deep reinforcement learning approach," Comput. Electr. Eng., vol. 99, Art. no. 107688, Apr. 2022.

[16] D. Alsadie, "A metaheuristic framework for dynamic virtual machine allocation with optimized task scheduling in cloud data centres," IEEE Access, 2021.

[17] M. Ashawa, O. Douglas, J. Osamor, and R. Jackie, "Improving cloud efficiency through optimized resource allocation technique for load balancing using LSTM machine learning algorithm," J. Cloud Comput., vol. 11, p. 87, 2022.

[18] J. P. Chandar and A. Tamilarasi, "Dynamic resource allocation with optimized task scheduling and improved power management in cloud computing," J. Ambient Intell. Humaniz. Comput., vol. 12, pp. 4147–4159, 2021.

[19] P. Kumar, S. Mahajan, and A. Kumar, "A comparative study of resource allocation techniques in cloud computing," Int. J. Eng. Res. Appl., vol. 10, no. 5, pp. 16–22, 2020.

[20] R. Kumar, S. Kumar, and A. K. Singh, "A metaheuristic approach for resource allocation in cloud computing using ant colony optimization," J. Intell. Inf. Syst., vol. 60, no. 2, pp. 257–271, 2022.

[21] S. S. Rao, S. K. Singh, and J. J. P. C. Rodrigues, "Simulated annealing-based resource allocation in cloud computing: A hybrid approach," IEEE Trans. Serv. Comput., vol. 16, no. 2, pp. 538–547, 2023.

[22] Z. Chen, J. Hu, G. Min, C. Luo, and T. El-Ghazawi, "Adaptive and efficient resource allocation in cloud datacentres using actor-critic deep reinforcement learning," IEEE Trans. Parallel Distrib. Syst., vol. 33, no. 8, pp. 1911–1923, Aug. 2022.

[23] O. Hajjar, E. Mekhallalati, N. Annwty, F. Alghayadh, I. Keshta, and M. Algabri, "Performance assessment of CPU scheduling algorithms: A scenario-based approach with FCFS, RR, and SJF," J. Comput. Sci., vol. 20, no. 9, pp. 972–985, 2024, doi: 10.3844/jcssp.2024.972.985.

[24] Z. Peng, J. Lin, D. Cui, Q. Li, and J. He, "A multi-objective trade-off framework for cloud resource scheduling based on the deep Q-network algorithm," Cluster Comput., vol. 23, no. 4, pp. 2753–2767, Dec. 2020.

[25] J. Lin, D. Cui, Z. Peng, Q. Li, and J. He, "A two-stage framework for the multi-user multi-data centre job scheduling and resource allocation," IEEE Access, vol. 8, pp. 19786–197874, 2020.

[26] H. Che, Z. Bai, R. Zuo, and H. Li, "A deep reinforcement learning approach to the optimization of data center task scheduling," Complexity, vol. 2020, p. 112, Aug. 2020.

[27] T. Dong, F. Xue, C. Xiao, and J. Li, "Task scheduling based on deep reinforcement learning in a cloud manufacturing environment," Concurrency Comput. Pract. Exp., vol. 32, no. 11, p. 112, Jun. 2020.

[28] S. Mostafavi and V. Hakami, "A stochastic approximation approach for foresighted task scheduling in cloud computing," Wireless Pers. Commun., vol. 114, no. 1, pp. 901–925, Sep. 2020.

[29] Y. Wei, L. Pan, S. Liu, L. Wu, and X. Meng, "DRL-scheduling: An intelligent QoS-aware job scheduling framework for applications in clouds," IEEE Access, vol. 6, pp. 55112–55125, 2018.

[30] J. Zhou, Z. Chang, J. Hu, J. Yu, and F. Li, "A modified PSO algorithm for task scheduling optimization in cloud computing," Concurrency Comput. Pract. Exp., vol. 30, no. 24, p. 111, 2018.

[31] M.-H. Kim, J.-Y. Lee, S. A. R. Shah, T.-H. Kim, and S.-Y. Noh, "Min-max exclusive virtual machine placement in cloud computing for scientific data environment," J. Cloud Comput.: Adv. Syst. Appl., 2021.

[32] S. A. Alsaidy, A. D. Abbood, and M. A. Sahib, "Heuristic initialization of PSO task scheduling algorithm in cloud computing," J. King Saud Univ. Comput. Inf. Sci., Nov. 2020.

[33] B. Kruekaew and W. Kimpan, "Multi-objective task scheduling optimization for load balancing in cloud computing environment using hybrid artificial bee colony algorithm with reinforcement learning," Feb. 17, 2022.

[34] J. Liu, S. Wang, A. Zhou, J. Xu, and F. Yang, "SLA-driven container consolidation with usage prediction for green cloud computing," Front. Comput. Sci., vol. 14, no. 1, pp. 42–52, 2019.

[35] G. A. Araújo, S. F. da C. Bezerra, and A. R. da Rocha, "Resource allocation based on task priority and resource consumption in edge computing," J. Internet Serv. Appl., vol. 15, no. 1, pp. 360–379, 2024, doi: 10.5753/jisa.2024.4026.

[36] N. K. G. and G. K. C., "User task priority based resource allocation with multi class task scheduling strategy and load balancing in cloud environment," SN Comput. Sci., vol. 5, p. 970, 2024, doi: 10.1007/s42979-024-03290-6.

[37] N. R. Talhar and D. P. Gaikwad, "Dynamic cloud resource allocation: Efficient optimization strategies for enhanced performance," Indian J. Technol. Educ., vol. 46, Special issue, pp. 366–375, 2023.

[38] N. R. Talhar and D. P. Gaikwad, "Design of hybrid approach for cloud resource allocation using genetic algorithms and ML techniques," in Signal Processing, Telecommunication and Embedded Systems with AI and ML Applications, V. Bhateja, V. V. S. S. S. Chakravarthy, J. Anguera, A. Ghosh, and W. Flores Fuentes, Eds., Lecture Notes in Electrical Engineering, vol. 1281, Springer, Singapore, 2025, doi: 10.1007/978-981-97-8422-6_34.