EAI Endorsed Transactions
on Scalable Information Systems
Research Article EAI.EU

Research on Multi-Objective Task Scheduling Optimization and Reinforcement Learning Decision-Making Methods for Edge-Cloud Collaborative Scalable Information Systems

Lina Guo[1,*], Chengyu Sun[2]

[1] School of General Education, Shandong Huayu University of Technology, Dezhou 253034, Shandong, China
[2] School of Innovation and Entrepreneurship, Shandong Huayu University of Technology, Dezhou 253034, Shandong, China

Abstract

INTRODUCTION: Edge-cloud schedulers must coordinate latency, energy, load balance, and deadline compliance under changing demand while keeping task-arrival and throughput units physically consistent.
OBJECTIVE: This study evaluates MORL-ECSO under an auditable, paired-seed simulation protocol and compares it with tuned heuristic, metaheuristic, value-based, actor-critic, and entropy-regularized baselines.
METHODS: A custom Python discrete-event simulator processes individual tasks in 0.1-s event windows. Offered load is 500-2000 tasks/s, with 4-20 edge nodes and four cloud nodes. DQN, A2C, discrete SAC, and MORL-ECSO receive the same 12,000 training transitions; GA and PSO use a population of 40, 30 iterations, and a common objective. Each reported test point uses 30 independent workload seeds after a 10-s warm-up and a 60-s measurement window. Means, standard deviations, 95% confidence intervals, paired Wilcoxon tests, Holm correction, and rank-biserial effects are reported.
RESULTS: At 2000 tasks/s, MORL-ECSO produced 103.16 ms mean latency, 88.80 kJ energy over 60 s, 94.18% on-time success, and 1947.56 tasks/s throughput. Relative to validation-tuned GA, latency was 18.53% lower, and success was 0.89 percentage points higher.
CONCLUSION: Compared to the optimal baseline, MORL-ECSO improves latency and deadline success rate under high loads without significantly altering energy consumption or throughput. In low-capacity scenarios with four nodes, the genetic algorithm remains superior.

Keywords: edge-cloud collaboration, multi-objective scheduling, discrete-event simulation, double DQN, statistical audit, reproducibility, throughput consistency

Received on 10 January 2026, accepted on 17 April 2026, published on 16 September 2026

doi: 10.4108/eetsis.14518

*Corresponding author. Email: 13050252406@163.com

1. Introduction

Edge computing moves computation and storage closer to data sources, whereas cloud computing provides elastic and high-capacity resources. Their integration forms an end–edge–cloud continuum in which a task may be executed locally, at an edge node, or in the cloud. Recent reviews approach the scheduling problem from distinct angles: delay-sensitive edge task scheduling [1], task-allocation methods and optimization techniques [2], deep-reinforcement-learning-based computation offloading [3], and fog-computing task-scheduling mechanisms [4]. Together, these studies show that scheduling in the continuum increasingly requires joint decisions on task placement, computing and communication resources,

queueing, deadlines, energy consumption, and heterogeneous node capabilities.

Recent studies have addressed these issues from complementary perspectives. Fan et al. jointly studied service placement, task scheduling, and resource allocation under edge–cloud cooperation [5], and later examined time-slotted task offloading and resource allocation in cloud–edge–end cooperative networks [6]. Zhang et al. formulated many-objective edge–cloud resource scheduling [7] and subsequently combined deep reinforcement learning with multiobjective optimization [8]. Shukla and Pandey studied multi-objective offloading and resource scheduling in heterogeneous fog–cloud systems [9]. Hosny et al. optimized dependent-task offloading under latency, energy, and resource-cost objectives [10]. Yin et al. proposed a convergent firefly algorithm for cloud–edge task scheduling [11]. Shen et al. developed reinforcement-learning scheduling for heterogeneous end–edge–cloud computing [12]. Ramezani Shahidani et al. combined reinforcement learning with multi-objective load balancing [13]. Song et al. introduced multi-objective offloading with changing objective preferences [14]. Sellami et al. proposed energy-aware deep-reinforcement-learning scheduling and offloading [15]. Yang et al. targeted low-latency mobile-edge task offloading [16]. Liu et al. studied task-graph offloading [17]. Pang et al. considered multi-agent offloading in ultra-dense heterogeneous mobile-edge networks [18], whereas Xiong et al. applied multi-agent offloading to distributed manufacturing systems [19]. Zhou et al. used two-stage deep-reinforcement-learning scheduling [20]. Ibrahim and Askar developed a multi-objective deep-reinforcement-learning scheduler for fog computing [21]. Wu et al. combined RNN-assisted online task recognition and caching with particle-swarm-based scheduling [22]. Zhang et al. emphasized deadline-aware dynamic scheduling [23], whereas Azizi et al. jointly considered deadlines and energy efficiency [24].

Despite this progress, evidence is often difficult to compare across studies because workload units, queue assumptions, training budgets, and statistical replication are not reported together. A scheduler may appear to improve throughput simply because arrival rate and completion rate are expressed on different scales. This study therefore treats reproducibility and dimensional consistency as part of the scheduling problem rather than as post hoc documentation.

To address these issues, this paper proposes multi-objective reinforcement learning for edge-cloud scheduling optimization (MORL-ECSO). The revised study makes four contributions. It defines a task-resource-link model with individual-task arrival units; uses a dynamically weighted, masked double-DQN policy; compares the method with RR, EDF, GA, PSO, DQN, A2C, and discrete SAC under matched budgets; and supplies seed-level logs, uncertainty intervals, paired significance tests, and executable configuration files.

2. Related Work

2.1. Edge-Cloud Collaborative Task Scheduling

Avan et al. review delay-sensitive task scheduling for edge computing [1]. Patsias et al. classify task-allocation methods and optimization techniques [2]. Peng et al. survey computation offloading from the perspective of deep reinforcement learning [3], while Hosseinzadeh et al. summarize task-scheduling mechanisms in fog computing [4]. Together, these reviews show that a collaborative end–edge–cloud scheduler must select execution tiers and allocate computing and communication resources while accounting for task size, required CPU cycles, deadlines, queue states, and link conditions.

Fan et al. jointly optimized service placement, task scheduling, and resource allocation under edge–cloud cooperation [5]. Their subsequent study incorporated the scheduling delay that occurs before slot-level decisions in cloud–edge–end cooperative networks [6]. Shen et al. studied reinforcement-learning-based task scheduling over heterogeneous end–edge–cloud hardware [12], whereas Wu et al. combined RNN-based online-task recognition and caching with particle-swarm-based scheduling for heterogeneous resources [22]. These studies demonstrate that edge–cloud collaborative scheduling is a cross-layer optimization problem involving execution placement, resource coordination, communication conditions, and temporal system states rather than a simple one-step allocation rule.

2.2. Multi-Objective Scheduling Optimization

Multi-objective optimization is appropriate for edge–cloud scheduling because low latency, low energy consumption, balanced load, low operating cost, high reliability, and deadline satisfaction are frequently conflicting objectives. Zhang et al. formulated edge–cloud resource scheduling as a many-objective optimization problem involving task completion time, monetary cost, load balance, user satisfaction, and resource trust [7]. Their later hybrid method combined deep-reinforcement-learning-based preprocessing with Pareto-oriented optimization of completion time, cost, energy consumption, and reliability [8]. Yin et al. proposed a convergent firefly algorithm for cloud–edge task scheduling [11]. Shukla and Pandey jointly considered task offloading and resource scheduling in heterogeneous fog–cloud environments [9], whereas Hosny et al. optimized latency, energy consumption, and resource cost for dependent-task offloading [10]. These studies establish a strong foundation for multi-objective scheduling. However, population-based and iterative search methods generally require a new optimization process when the task and resource states change. MORL-ECSO instead targets low-overhead online policy inference after the learning stage.

2.3. Reinforcement Learning-based Scheduling Decision

Peng et al. show that deep-reinforcement-learning-based offloading can formulate repeated resource-allocation decisions under changing system states [3]. This property is valuable in edge-cloud environments because an allocation decision affects not only the currently scheduled task but also later queue states, node loads, and available communication resources. Ramezani Shahidani et al. combined reinforcement learning with multi-objective load balancing in an edge-fog-cloud architecture [13]. Song et al. formulated dependent-task offloading as a multi-objective reinforcement-learning problem and incorporated changing objective preferences into policy learning [14]. Sellami et al. used deep reinforcement learning for energy-aware task scheduling and offloading [15]. More recent studies have considered low-latency task offloading [16], task-graph offloading [17], multi-agent decisions in ultra-dense heterogeneous mobile edge computing [18], distributed manufacturing task offloading [19], and two-stage end-edge-cloud scheduling [20]. Ibrahim and Askar further proposed a multi-objective deep-reinforcement-learning scheduling strategy for fog computing [21]. Collectively, these studies show that learning-based scheduling has evolved from single-task and single-objective decisions toward multi-resource, dependency-aware, multi-agent, and multi-objective formulations.

2.4. Summary of Existing Research

The existing literature establishes three important foundations. First, dynamic edge–cloud task scheduling can be represented as a sequential decision problem. Second, deep reinforcement learning provides a practical way to make online decisions in high-dimensional and time-varying state spaces. Third, multi-objective formulations can coordinate latency, energy consumption, load balance, monetary cost, reliability, and deadline-related service requirements.

The contribution of MORL-ECSO is therefore not the first use of deep reinforcement learning or dynamic weighting. Its contribution is the combined use of load- and priority-conditioned objectives, a queue-aware dispatch projection, matched-capacity baselines, and an auditable evaluation protocol in which input rate, completion rate, confidence intervals, and statistical tests share one measurement definition.

3. System Model and Problem Formulation

3.1. Edge–Cloud Collaborative Scalable Information System Architecture

The edge-cloud collaborative scalable information system studied in this paper consists of a user request layer, an edge resource layer, a cloud resource layer, and a global scheduling and control layer. The user request layer is mainly responsible for continuously generating heterogeneous task streams. These tasks have significant differences in business attributes, including different data scales, different computational complexities, different priorities, and different deadline constraints. Figure 1 shows a diagram of the system architecture. The edge resource layer consists of several distributed edge nodes that are close to the data source, have fast response speeds, and have low network round-trip overhead. They are suitable for handling latency-sensitive tasks and lightweight real-time analysis tasks. The cloud resource layer is deployed in a centralized data centre and has higher computing power, storage capacity, and unified resource scheduling capabilities. It is more suitable for handling computationally intensive, delayable, and large-scale batch processing tasks. The global scheduling and control layer is responsible for collecting real-time status information of edge and cloud nodes, including CPU utilization, storage occupancy, bandwidth utilization, node load rate, link latency, and task queue length, and outputs control decisions such as task allocation, task migration, or task delay execution according to the scheduling strategy.

In this architecture, tasks are not always fixed to the edge or cloud but rather dynamically flow between the edge and cloud based on system load, network status, and task characteristics. Scalability is mainly reflected in two aspects: first, the system allows the number of edge nodes to expand elastically according to the business scale, thereby coping with scenarios with a surge of high-concurrency tasks; second, the scheduling and control layer can dynamically adjust the execution ratio of tasks between the edge and cloud based on resource pressure, avoiding the formation of local hotspot nodes and improving the overall system throughput. As edge nodes typically exhibit significant heterogeneity in terms of hardware capabilities, network quality, and business location, whereas cloud resources have obvious centralization and stability, edge-cloud collaborative scheduling is essentially a cross-layer heterogeneous resource optimization problem.

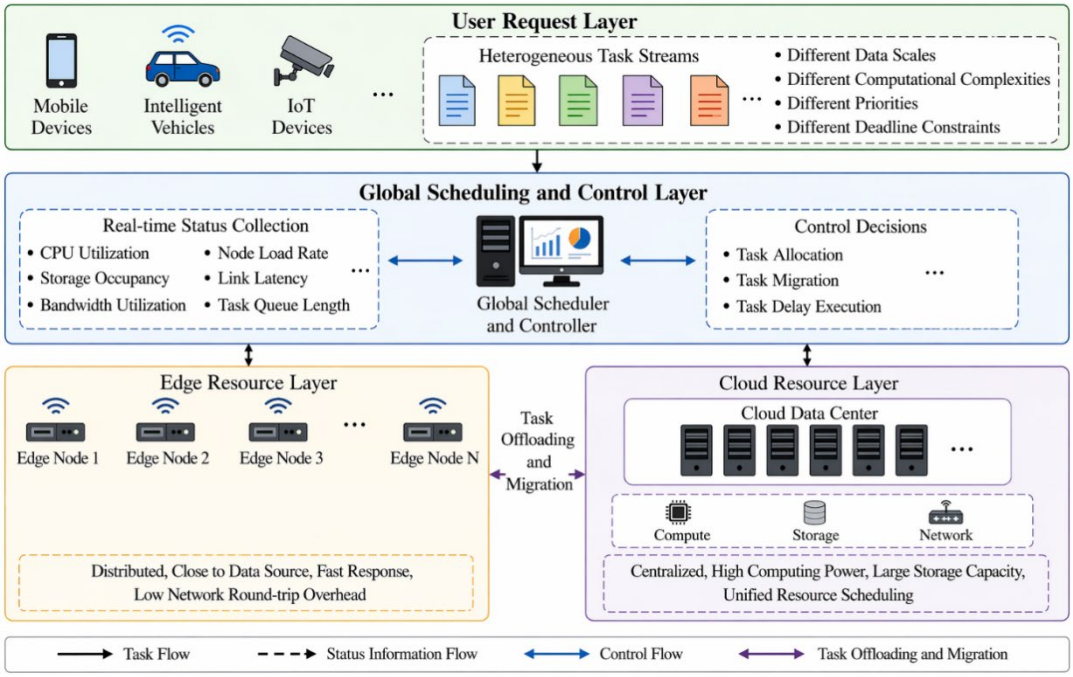

Figure 1. Overall architecture diagram of an edge-cloud collaborative scalable information system

3.2. Task Model

To accurately describe the task characteristics in the system, this paper represents a single task T_i as a quintuple:

$$T_i = (d_i, c_i, p_i, \tau_i, \delta_i) \tag{1}$$

where d_i is the input data size, c_i is the required computation in CPU cycles, p_i is the task priority, τ_i is the arrival time, and δ_i is the absolute deadline. Larger d_i usually increases transmission delay, whereas larger c_i increases execution time on a given node. Tasks are grouped as latency-sensitive, computation-intensive, or mixed-service requests. The task model follows established formulations from complementary perspectives. Fan et al. represent time-slotted cloud–edge–end offloading and resource allocation [6]. Song et al. explicitly formulate dependent-task offloading [14]. Liu et al. encode dependencies through task graphs [17]. Zhang et al. emphasize deadline constraints [23]. Azizi et al. jointly consider deadlines and energy efficiency [24].

Task arrivals are counted as individual requests. In each event interval of length $\Delta t = 0.1$ s, the number of new tasks is sampled from a Poisson distribution with mean $\lambda \Delta t$. The simulator may aggregate these requests for vectorized queue updates, but the horizontal axes, throughput values, and success denominators remain in individual tasks rather than task batches.

3.3. Resource Model

Let the set of edge nodes be $\mathcal{E} = \{E_1, E_2, \ldots, E_M\}$, and the set of cloud nodes be $\mathcal{C} = \{C_1, C_2, \ldots, C_K\}$. For any edge node E_m, this paper uses a resource state vector.

$$E_m = (f_m, b_m, s_m, e_m) \tag{2}$$

where f_j is the available computing capacity, b_j is the available bandwidth, s_j is the available storage capacity, and e_j is the unit computation-energy coefficient. These four dimensions form the resource vector used in this paper. Fan et al. jointly model edge–cloud service placement, task scheduling, and resource allocation [5]. Zhang et al. formulate many-objective edge–cloud resource scheduling [7] and later combine deep reinforcement learning with multiobjective resource optimization [8]. Shen et al. model heterogeneous end–edge–cloud resources for reinforcement-learning scheduling [12]. Wu et al. coordinate heterogeneous end–edge–cloud resources using RNN and particle swarm optimization [22]. Cloud node C_k uses the same type of state vector but generally has larger f_k and s_k, together with a longer network path.

$$\rho_j = \frac{u_j}{U_j} \tag{3}$$

where u_j is the node's current normalized resource use and U_j is its effective capacity. CPU, storage, and bandwidth utilization must be normalized separately before they are combined with the implementation weights; otherwise, quantities with different units are not comparable. A load rate close to one indicates a higher risk of queueing delay and additional energy use.

Node scaling is represented through aggregate edge-tier and cloud-tier capacity, utilization, and queue pressure. Each run samples the active edge-node count from the stated scale or fixes it at the scenario value. The state

therefore keeps a constant input dimension while preserving the capacity and load effects of 4-20 active edge nodes.

3.4. Communication and Delay Model

The total task execution delay consists of three parts: transmission delay, queuing delay, and execution delay. If task T_i is assigned to edge node E_m, its total delay can be expressed as follows:

$$D_{i,m}^{edge} = D_{i,m}^{tr} + D_{i,m}^{q} + D_{i,m}^{ex} \quad (4)$$

where $D_{i,m}^{tr}$ represents the latency of transmitting the task from the terminal to the edge node, $D_{i,m}^{q}$ represents the waiting latency of the task in the queue of the target edge node, and $D_{i,m}^{ex}$ represents the execution latency of the task on the target edge node. The transmission latency is related to the amount of input data and the link bandwidth, and can be written as:

$$D_{i,m}^{tr} = \frac{d_i}{r_{i,m}} \quad (5)$$

The effective transmission time includes a task-size term, a class-dependent base round-trip time, and correlated link jitter. The normal scenario uses effective rates of 150 Mb/s for terminal-edge transfer and 350 Mb/s for the cloud path; the jitter process follows an AR(1) model with coefficient 0.75 and 1.5-ms Gaussian innovations. The disturbance case raises the innovation standard deviation to 7 ms for 10 s.

If the task is offloaded to the cloud for execution after being relayed through the edge layer, the total latency is expressed as:

$$D_{i,k}^{\text{cloud}} = D_{i,m}^{tr} + D_{m,k}^{bh} + D_{i,k}^{q} + D_{i,k}^{ex} \quad (6)$$

where $D_{m,k}^{bh}$ is the backhaul latency between the edge node and the cloud node, $D_{i,k}^{q}$ is the cloud queuing latency, and $D_{i,k}^{ex}$ is the cloud execution latency. The task execution latency is determined by the task's computational load and the available computing power of the node.

$$D_{i,m}^{ex} = \frac{c_i}{f_m}, D_{i,k}^{ex} = \frac{c_i}{f_k} \quad (7)$$

Equations (4)-(7) use a common task-arrival timestamp. Queue waiting, transmission, and execution components are therefore additive and expressed in seconds before conversion to milliseconds. A task can be counted as successful only if it is completed within the measurement horizon and its predicted completion time does not exceed its relative deadline.

It can be observed that edge nodes typically have an advantage in transmission latency but weaker computing power, whereas the cloud has an advantage in execution latency but often comes with higher backhaul costs. Therefore, reasonable scheduling decisions should be based on task type and node status to establish a dynamic balance between edge execution costs and cloud migration benefits.

3.5. Energy Consumption Model

In addition to latency, total system energy consumption is also a crucial optimization objective for edge cloud scheduling. In this study, the energy consumption of a task on a node is expressed as the sum of its computational and transmission energy consumptions:

$$E_i = E_i^{\text{comp}} + E_i^{\text{tr}} \quad (8)$$

where E_i^{comp} represents the energy consumption for task execution computation, and E_i^{tr} represents the energy consumption for task transmission. If the task is executed on node j, the computation energy consumption can be written as follows:

$$E_i^{\text{comp}} = P_j^{\text{comp}} \cdot D_{i,j}^{ex} \quad (9)$$

where P_j^{comp} represents the computational power of node j. Transmission energy consumption is...

$$E_i^{tr} = P_{i,j}^{tr} \cdot D_{i,j}^{tr} \quad (10$$

where $P_{i,j}^{tr}$ represents the average power of the transmission task on the corresponding link. The total system energy consumption is the sum of the energy consumption of all tasks, i.e.

$$E_{\text{total}} = \sum_{i=1}^{N} E_i \quad (11)$$

The implemented power coefficients are 6 W idle and 26 W dynamic power per active edge node, and 45 W idle and 120 W dynamic power per cloud node. Transfer energy uses 35 nJ/bit for the edge path and 55 nJ/bit for the cloud path. These coefficients are simulation inputs rather than measurements from a named commercial platform; their exact values are included in Appendix Table A1 and the audit package.

Finite FCFS workload queues are used. The edge queue is capped at 1.2 s of service demand and the cloud queue at 2.0 s. Work that exceeds the corresponding capacity is rejected and contributes to the violation term, preventing unlimited backlog from being misreported as future throughput.

3.6. Multi-Objective Scheduling Optimization Model

Based on the above modelling, in this study, the task scheduling problem is formulated as a multi-objective joint optimization problem. First, the objective is to minimize the average task completion latency:

$$\min F_1 = \frac{1}{N}\sum_{i=1}^{N} D_i \quad (12)$$

where D_i is the actual completion delay of task T_i. Secondly, the objective is to minimize the total system energy consumption:

$$\min F_2 = \sum_{i=1}^{N} E_i \tag{13}$$

Furthermore, to avoid local resource overload, a node load deviation minimization objective is introduced as follows:

$$\min F_3 = \sqrt{\frac{1}{M+K}\sum_{j=1}^{M+K}\left(\rho_j - \bar{\rho}\right)^2} \tag{14}$$

where $\bar{\rho}$ represents the average load rate of all nodes. In addition, it is also necessary to ensure task completion rate and system throughput; therefore, the following definition is used:

$$\max F_4 = \frac{N_{\text{succ}}}{N}, \max F_5 = \frac{N_{\text{comp}}}{\Delta t} \tag{15}$$

where N_{succ} represents the number of tasks completed before the deadline, and N_{comp} represents the number of tasks completed within the time interval Δt.

Define the decision variable $x_{i,j}$ as a binary variable. If task T_i is assigned to node j, then $x_{i,j} = 1$; otherwise, it is 0. The scheduling process must satisfy the following constraints:

$$\sum_{j=1}^{M+K} x_{i,j} = 1, \quad \forall i \tag{16}$$

This constraint means that each task can only select one execution target, that is:

$$\sum_{i=1}^{N} x_{i,j}\, r_i \le U_j, \quad \forall j \tag{17}$$

Equation (17) requires the normalized demand assigned to a node not to exceed its capacity. If CPU, storage, and bandwidth are constrained separately, the inequality must hold componentwise for every resource dimension.

$$D_i \le \delta_i, \forall i \in \mathcal{T}_{\text{succ}} \tag{18}$$

This constraint means that a task that is successfully completed must meet the deadline requirement. In summary, this problem involves discrete decision-making, resource competition, and multi-objective conflict; therefore, it is difficult to solve in real time in a dynamic environment using traditional exact algorithms.

3.7. Problem Complexity Analysis

With N tasks and $M+K$ candidate nodes, one static allocation round contains $(M+K)^N$ possible assignments before link variation, priorities, scale changes, and feedback are considered. This combinatorial growth motivates an online learned policy but is not presented here as a formal proof of NP-hardness. Compared with repeated iterative search, a trained policy produces each decision by a forward pass, subject to the implementation-level cost discussed in Section 4.8.

4. Proposed MORL-ECSO Method

4.1. Overall Design of MORL-ECSO

To address the above problem, this study proposes multi-objective reinforcement learning for edge-cloud scheduling optimization (MORL-ECSO). The method formulates edge-cloud task scheduling as a multi-objective Markov decision process. In each scheduling cycle, the scheduler selects an allocation action from the current system state and updates its policy through environmental feedback. Compared with conventional DQN scheduling, MORL-ECSO combines a multi-objective reward function, dynamic weight adjustment, priority experience replay, and state enhancement, so that the scheduling policy can respond more steadily to workload fluctuation and resource heterogeneity.

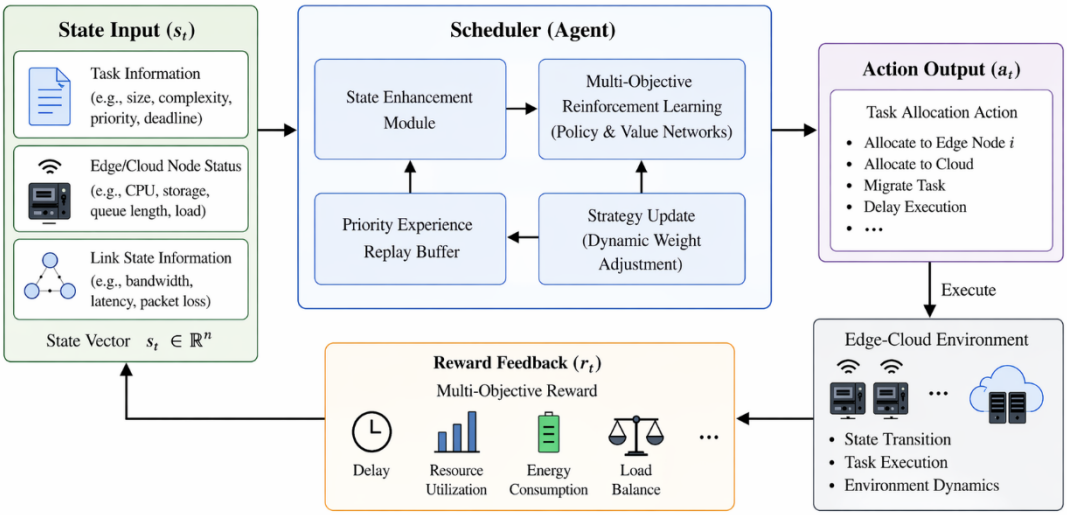

Figure 2. Framework Diagram of Multi-Objective Reinforcement Learning Task Scheduling

Figure 2 shows the relationship among state input, action output, reward feedback, and policy updating. The method has three main design points. First, task attributes, node load, and link status are encoded together so that the scheduler has cross-layer system awareness. Second, a dynamic multi-objective reward adjusts the optimization focus under different load and service pressures. Third, reinforcement learning is used to optimize long-term cumulative benefits instead of only the local gain of a single scheduling round.

4.2. Markov Decision Process Formulation

To convert the scheduling problem into a learnable decision process, this paper defines a quadruple (S, A, P, R). Here, S represents the state space, A represents the action space, P represents the state transition probability, and R represents the reward function. Given the current state s_t, after the scheduler selects action a_t, the system transitions to the next state s_{t+1} and returns an immediate reward r_t. The goal of reinforcement learning is to learn a policy $\pi(a \mid s)$ through long-term interaction and maximize the expected discounted return:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{19}$$

where γ is a discount factor used to control the balance between current and future rewards.

This modelling approach is suitable for edge cloud scheduling because scheduling decisions affect not only the current task completion status but also the node load distribution, queue status, and link pressure at subsequent time steps. Therefore, treating the problem as a sequential decision-making process is more consistent with the actual operating rules of the system than simple static optimization.

4.3. State Space Design

The policy uses a fixed-length composite state consisting of tier utilization, effective link conditions, normalized queue pressure, and current workload statistics:

$$S_t = [\mathrm{R}_t, \mathrm{B}_t, \mathrm{Q}_t, \mathrm{T}_t] \tag{20}$$

Here $\mathbf{R}_t$ summarizes mean edge and cloud utilization, utilization dispersion, and the active-edge-node ratio. This representation keeps the network input fixed while allowing physical capacity to change with node count.
The resource component is defined as:

$$\mathbf{R}_t = \left[\bar{\rho}_t^e, \bar{\rho}_t^c, \sigma_{\rho,t}, M_t/M_{\max}\right] \tag{21}$$

The link component records effective tier rates, base latency, and the current jitter state:

$$\mathbf{B}_t = \left[\bar{b}_t^e, \bar{b}_t^c, \bar{\ell}_t^e, \bar{\ell}_t^c, j_t\right] \tag{22}$$

The superscripts e and c denote edge and cloud tiers. The jitter state is updated at every event interval and is shared by all policies under the same test seed.

Normalized queue pressure and its short-term change are represented by:

$$\mathbf{Q}_t = [q_t^e/F_t^e, q_t^c/F_t^c, \Delta q_t] \tag{23}$$

In Equation (23), q is queued CPU demand, and F is effective tier capacity after any node failure. The workload component is:

$$\mathbf{T}_t = \left[\lambda_t, \boldsymbol{\pi}_t, \bar{c}_t, \bar{d}_t, \bar{\delta}_t, \rho_t^H\right] \tag{24}$$

The base state contains 14 scalars: offered load, three task-class proportions, mean computation, mean data size, mean deadline, high-priority ratio, two normalized queues, two utilization values, jitter magnitude, and active-edge ratio. MORL-ECSO adds four derived features - load trend, utilization imbalance, failed-edge fraction, and mean queue pressure - for an 18-dimensional input. DQN, A2C, and SAC retain the 14-dimensional state; hidden widths are adjusted so that parameter counts remain comparable.

4.4. Action Space Design

The network does not emit an unconstrained node index. It chooses one of six dispatch templates, after which a feasibility projection allocates individual tasks across online edge and cloud capacity:

$$\mathcal{A} = \{a_{\text{edge}}, a_{\text{cloud}}, a_{\text{bal}}, a_{\text{ddl}}, a_{\text{energy}}, a_{\text{load}}\} \tag{25}$$

The templates respectively emphasize edge locality, cloud computation, balanced placement, deadlines, energy, and load-aware placement. The load-aware template preserves edge locality for latency-sensitive work, shifts compute-intensive work toward the tier with the shorter normalized queue, and adjusts mixed traffic between the two tiers.

Internal masks exclude failed or zero-capacity resources before proportional allocation. If an action index is invalid, the implementation falls back to the balanced template and applies a reward penalty of 0.2. Thus, an unavailable node or tier cannot receive work merely because its Q value is large.

4.5. Multi-Objective Reward Function Design

The reward function is the core of MORL-ECSO. If the reward design is flawed, the strategy will be biased towards a single metric, thereby failing to truly reflect the optimization of all objectives. Therefore, in this study, we construct the following joint reward function:

$$r_t = -\alpha_t \hat{D}_t - \beta_t \hat{E}_t - \gamma_t \hat{L}_t + \lambda_t \hat{S}_t - \mu_t \hat{V}_t \tag{26}$$

In Equation (26), latency is divided by 0.45 s and clipped at 3, slot energy is divided by 100 J and clipped at 3, load skew is the absolute tier-utilization difference, and success is the on-time completion fraction. Queue rejection and incomplete service form the violation term. The resulting raw reward is standardized with training-only

running moments and clipped to [-5,5] before a replay update.

Let the reward-weight vector follow the order: latency, energy, balance, success, and violation. Before L1 normalization, it is adjusted by mean utilization and the high-priority proportion as follows:

$$\tilde{w}_t = [0.28,0.20,0.13,0.25,0.14] + \bar{u}_t[0.18,-0.08,0.08,0,0] + \rho_t^H[0.12,0,0,0.15,0]$$
$$w_t = \tilde{w}_t / \| \tilde{w}_t \|_1 \quad (27)$$

Higher utilization therefore raises latency and balance pressure while reducing the relative energy weight; a larger high-priority share increases latency and success weights. The same normalization is applied in every training seed, and all coefficients are fixed before test evaluation.

4.6. Network Structure and Policy Learning

MORL-ECSO uses an 18-120-64-32-6 fully connected double-DQN with ReLU activations. The online network selects the feasible action, and the target network evaluates it. The model contains 12,302 trainable parameters; DQN, A2C, and discrete SAC contain 12,454, 12,487, and 9,714 parameters, respectively.

$$a_t^\star = \arg\max_{a'} Q(s_{t+1}, a')$$
$$y_t = r_t + \gamma Q^-(s_{t+1}, a_t^\star)$$
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \zeta[y_t - Q(s_t, a_t)] \quad (28)$$

Adam optimization uses a learning rate of 3×10^{-4}, discount factor $\gamma = 0.97$, batch size 128, replay capacity 100,000, and gradient clipping at 5. The target network is synchronized every 100 gradient updates. All off-policy agents update once per 64 environment transitions. A2C uses 32-step rollouts, an entropy coefficient of 0.01, a value coefficient of 0.5, and gradient clipping at 0.5.

MORL-ECSO uses prioritized replay with $\alpha = 0.6$ and β increasing linearly from 0.4 to 1.0. DQN and SAC use uniform replay. Discrete SAC uses two Q networks, soft target updates with $\tau = 0.005$, and automatic entropy tuning toward 0.9 ln(6).

$$\epsilon_t = \epsilon_{\min} + (\epsilon_{\max} - \epsilon_{\min})\exp(-t/\tau_\epsilon) \quad (29)$$

$$\hat{r}_t = \mathrm{clip}\big((r_t - \mu_r)/(\sigma_r + 10^{-8}), -5, 5\big) \quad (30)$$

For DQN and MORL-ECSO, ε begins at 1.0, decays toward 0.05, and uses τ = 12,000 transitions in Equation (29). Running reward moments are updated only on the training stream and are frozen during validation and test.

4.7. Online Scheduling Procedure

At every 0.1-s event, the simulator samples individual task counts, updates the AR(1) jitter state, calculates queue-admissible dispatch fractions, processes available CPU capacity, and records arrivals, completions, on-time completions, rejections, energy, and policy decision time. This order is fixed across all methods. Figure 3 is a flowchart of the MORL-ECSO algorithm.

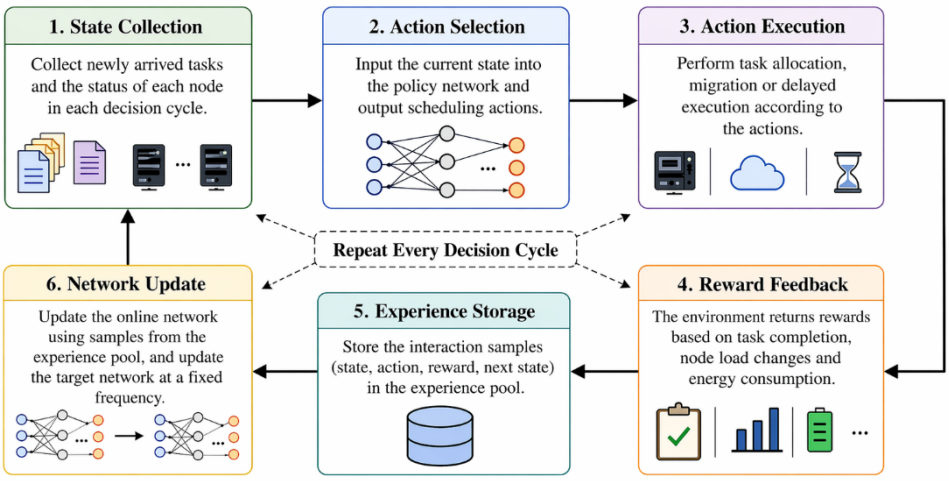

Figure 3. Flowchart of MORL-ECSO Algorithm.

After training, each scheduling decision requires one forward pass. Periodic background updates are possible, but a deployed system should freeze validated policy versions and provide rollback and offline checks before a new policy is activated.

4.8. Complexity Analysis of MORL-ECSO

The final policy is selected by validation return rather than test performance. All principal methods use the same number of training transitions and the same workload generator. Appendix Table A2 reports architecture, parameter count, optimizer, search budget, and stopping rule so that improvements cannot be attributed to a weak or under-trained comparator.

5. Experimental Design

5.1. Experimental Environment and Parameter Settings

The simulator is a custom Python 3.13.5 implementation using NumPy 2.3.5, PyTorch 2.10.0, pandas 2.2.3, and SciPy 1.17.0. It was executed on five virtual Intel Xeon Platinum 8370C CPU cores with 5.9 GB RAM. The event interval is 0.1 s. Each test run uses a 10-s warm-up followed by a 60-s measurement window; warm-up completions are excluded from reported throughput.

Individual task arrivals follow $N_t \sim \text{Poisson}(\lambda \Delta t)$, with $\lambda \in \{500, 750, 1000, 1250, 1500, 1750, 2000\}$ tasks/s. Latency-sensitive, computation-intensive, and mixed-service tasks occur with probabilities 0.30, 0.40, and 0.30. Their median CPU demands are 50, 300, and 140 million cycles; median input sizes are 0.20, 1.20, and 0.55 MB; and median relative deadlines are 0.18, 1.20, and 0.55 s. CPU-cycle multipliers are lognormal with class-specific $\sigma = (0.18, 0.22, 0.20)$ and clipping to [0.55, 1.70]; data-size multipliers use $\sigma = (0.20, 0.25, 0.22)$ and clipping to [0.50, 1.80]; deadlines are multiplied by U(0.80, 1.20); and high-priority ratios receive $N(0, 0.015^2)$ perturbations and are clipped to [0.05, 0.70].

Each run samples edge capacity independently as the sum of per-node U(8,20) GHz capacities and cloud capacity as the sum of four U(40,80) GHz nodes. The edge and cloud finite queues correspond to 1.2 and 2.0 s of service demand. Test seeds are 3000-3029. Training uses initialization seeds 1000-1002; validation streams are generated from a disjoint 2,000,000-series seed block. Complete parameter definitions are listed in Appendix Table A1.

5.2. Baseline Methods

Eight principal methods are evaluated: RR, EDF, GA, PSO, DQN, A2C, discrete SAC, and MORL-ECSO. RR and EDF provide non-learning controls; GA and PSO provide online metaheuristic controls; DQN, A2C, and SAC are the main deep-reinforcement-learning baselines.

GA and PSO both use 40 candidates, at most 30 iterations, and early termination after eight iterations without improvement. Re-optimization occurs every 5 s using the same normalized objective as MORL-ECSO. GA uses elitist selection, arithmetic crossover, mutation probability 0.35, and mutation standard deviation 0.07. PSO uses $c_1 = c_2 = 1.7$ and linearly decreasing inertia from 0.9 to 0.4. All six dispatch templates are inserted into the initial population or swarm. Neural baselines receive 120 episodes × 100 decisions = 12,000 training transitions and use the matched configurations in Appendix Table A2.

5.3. Evaluation Metrics

All metrics are calculated from the same 60-s measurement window. Throughput is the number of individual completed tasks divided by 60 s and is capped by the number of tasks that actually arrived during that window. Thus, it cannot exceed offered load because of a batch/task unit mismatch.

1) Mean task-completion latency, measured from individual-task arrival to completion.

2) Total system energy, defined as computation plus transmission energy accumulated over the fixed 60-s measurement window.

3) Energy per completed task, calculated as total measurement-window energy divided by completed individual tasks.

4) Load-balance factor, represented by the dispersion of normalized utilization across active edge and cloud resources.

5) On-time task-success ratio, defined as deadline-compliant completions divided by all individual arrivals in the measurement window.

6) Throughput, defined as completed individual tasks divided by 60 s and capped by arrivals observed in the same window.

7) Decision overhead, measured as wall-clock policy or optimizer selection time; mean and P95 values are retained in the raw logs.

5.4. Experimental Scenarios

1) Offered-load scenario: λ = 500–2000 tasks/s in 250-tasks/s increments, with 12 edge nodes and four cloud nodes.

2) Computation-intensity scenario: the class-specific CPU-cycle medians are multiplied by 0.50-1.75 at 1500 tasks/s.

3) Workload-composition scenario: latency-heavy, computation-heavy, mixed-service, and balanced class proportions are evaluated at 1800 tasks/s.

4) Scale and overhead scenario: active edge nodes are {4, 8, 12, 16, 20} at 1500 tasks/s, and both performance and absolute decision time are recorded.

5) Dynamic-disturbance scenario: link-jitter variance is increased for 10 s; 25% of edge capacity is removed for 10 s and then restored; or arrival rate is multiplied by 1.8 for 5 s. The disturbance begins 15 s into the measurement window.

6) Ablation scenario: dynamic reward adjustment, prioritized replay, or enhanced state features are removed separately. Three initialization seeds and ten paired workload seeds per initialization produce 30 observations per ablation.

5.5. Experimental Procedure

Each neural method is trained for 120 episodes of 100 decisions. Validation is performed every 20 episodes on two disjoint streams, and the checkpoint with the highest mean validation return is retained. Training stops at 120 episodes; the reported convergence indicator is the validation-selected episode, not a test-selected stopping

point. No replay sample or normalization statistic is shared across training, validation, and test partitions. Three independent initialization seeds (1000–1002) were used for each neural method, with 12,000 training transitions per initialization. For each of the 30 paired workload test seeds, performance was evaluated using the three independently trained validation-selected checkpoints, and the three checkpoint-level results were averaged before calculating the reported workload-level means, confidence intervals, and paired statistical tests.

5.6. Data Recording and Statistical Analysis

For each metric and scenario, the manuscript reports the sample mean, sample standard deviation, and two-sided 95% t confidence interval. High-load comparisons use paired Wilcoxon signed-rank tests because latency and success distributions are skewed. p values are Holm-adjusted separately within each metric family. Paired rank-biserial correlation is reported as the effect size, with positive values favoring MORL-ECSO.
The supplementary audit package contains the executable simulator, exact configuration, trained checkpoints, all 30 seed-level summaries, disturbance time series, table-generation code, and SHA-256 manifest. Appendix Tables A3-A5 reproduce the principal summary, inferential tests, and raw MORL-ECSO versus GA observations.

6. Results and Analysis

6.1. Training Stability and Checkpoint Selection

The validation-selected checkpoint occurred at episode 33.3 ± 23.1 for MORL-ECSO, compared with 46.7 ± 30.6 for A2C, 53.3 ± 30.6 for SAC, and 73.3 ± 30.6 for DQN (Figure 4). The large between-seed variation prevents a claim of deterministic convergence speed, but all selected checkpoints occurred before the 120-episode cap and remained separate from the test set.

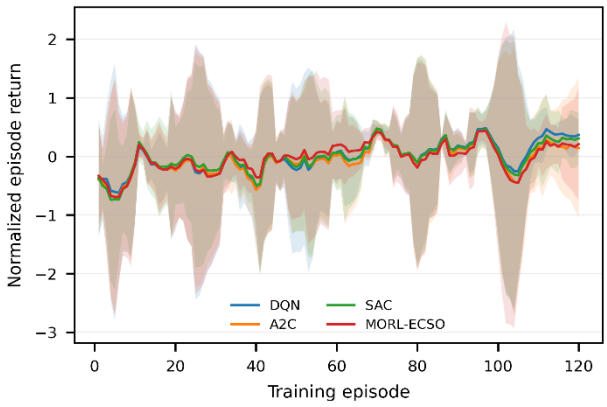

Figure 4. Mean normalized training return across three initialization seeds; shaded bands show 95% confidence intervals

Figure 5 complements the return curves with robustly scaled loss trajectories. The broad early intervals and occasional spikes show that optimizer transients differ across seeds; no method is therefore claimed to have a uniformly smoother training path.

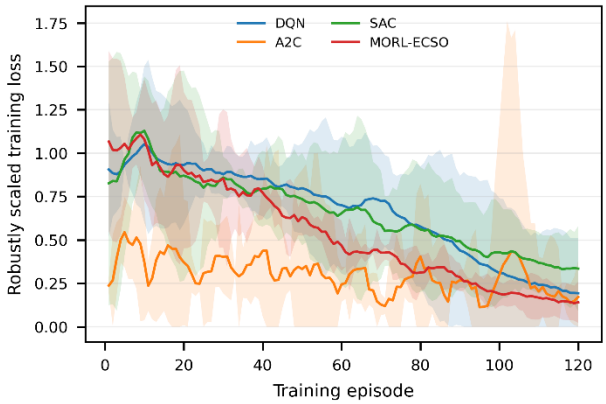

Figure 5. Robustly scaled training loss across three initialization seeds; shaded bands show 95% confidence intervals

6.2. Delay Performance under Different Task Loads

At 1500 tasks/s, MORL-ECSO, GA, PSO, and SAC all remain near 52 ms mean latency and above 99% on-time success. Separation appears near saturation. At 2000 tasks/s, MORL-ECSO reaches 103.16 ± 200.77 ms, whereas validation-tuned GA reaches 126.63 ± 269.88 ms. The paired reduction is 18.53% (Holm-adjusted $p < 0.001$; rank-biserial effect 0.935) (Figure 6).

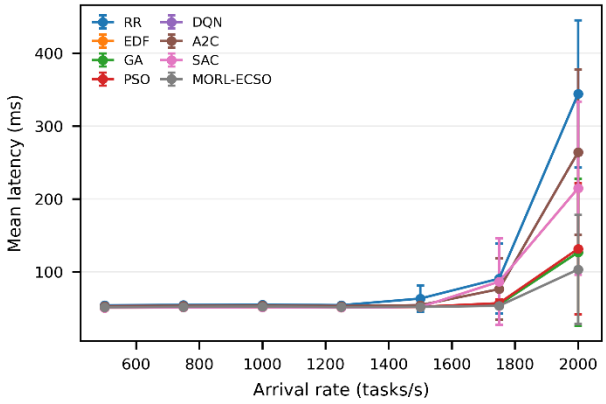

Figure 6. Mean task-completion latency versus individual-task arrival rate; error bars show 95% confidence intervals over 30 test seeds

6.3. Energy Consumption and Energy Efficiency Analysis

Energy is reported over the same 60-s window rather than as an untraceable aggregate (Figure 7). Across computation

scales 0.50-1.75, MORL-ECSO and GA differ by less than 1% in total energy. At the default scale and 2000 tasks/s, MORL-ECSO uses 88.80 kJ and GA 88.73 kJ; the 0.08% difference is not significant after Holm correction. Under the 1.75 computation multiplier, MORL-ECSO reduces latency from 1462.26 to 1161.45 ms while energy changes from 86.82 to 86.23 kJ, indicating that its principal gain is queue placement rather than a large energy reduction.

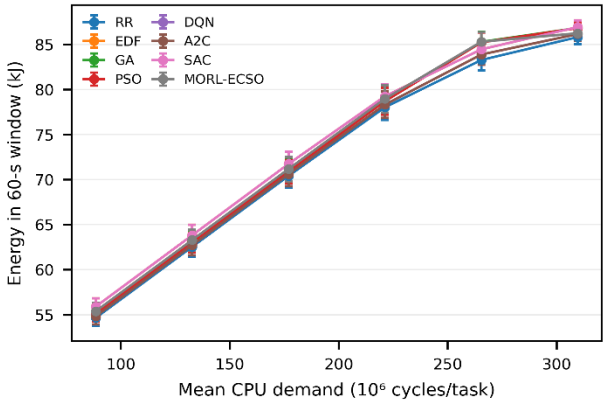

Figure 7. Total 60-s system energy versus weighted mean CPU demand; error bars show 95% confidence intervals over 30 test seeds

6.4. Load Balancing and Resource Utilization Analysis

At 2000 tasks/s, MORL-ECSO maintains mean edge and cloud utilizations of 88.18% and 86.61%, respectively. GA records 87.62% and 86.65% (Figure 8). The close tier values explain why both methods sustain nearly identical throughput, while MORL-ECSO obtains lower latency through class-specific placement rather than by leaving capacity idle.

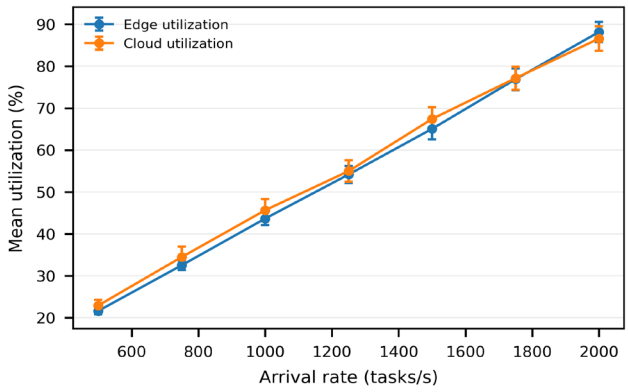

Figure 8. MORL-ECSO edge- and cloud-tier utilization versus arrival rate; error bars show 95% confidence intervals over 30 test seeds

6.5. Task Success Ratio and Throughput Analysis

MORL-ECSO achieves 94.18% on-time success at 2000 tasks/s, compared with 93.29% for GA and 92.74% for PSO (Figure 9). The paired gain over GA is 0.89 percentage points (Holm-adjusted p = 0.002; rank-biserial effect 0.639). The smaller absolute gain than the latency reduction reflects that many runs remain below their deadlines despite shorter queues.

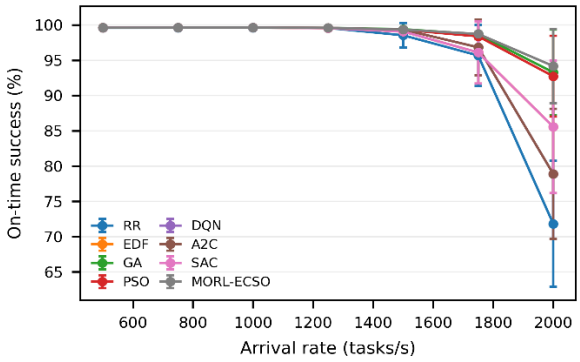

Figure 9. On-time task success versus individual-task arrival rate; error bars show 95% confidence intervals over 30 test seeds

Workload composition changes the ranking. In the latency-heavy case, all main methods process about 1799 tasks/s with roughly 99.3% success (Figure 10). In the computation-heavy case, throughput falls to 1581.65 tasks/s for MORL-ECSO, 1605.62 for GA, and 1610.53 for DQN; MORL-ECSO nevertheless raises on-time success to 27.73%, compared with 23.41% for GA. This is a deliberate latency-deadline trade-off rather than universal throughput dominance.

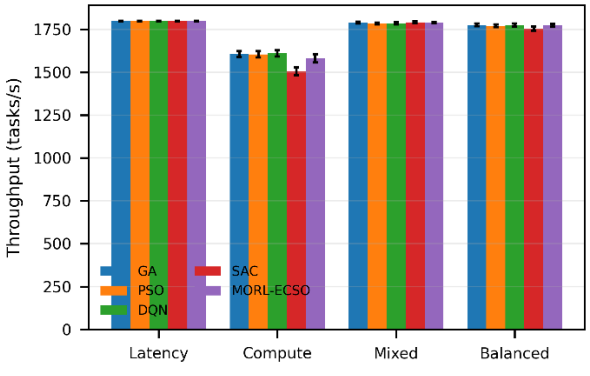

Figure 10. Throughput under four workload compositions; error bars show 95% confidence intervals over 30 test seeds

6.6. Scalability Analysis in Large-Scale Edge-Cloud Environments

Node scaling reveals a capacity boundary (Figure 11). With only four edge nodes, GA obtains 349.88 ms mean latency, while MORL-ECSO and SAC reach 631.65 and 649.08 ms. From eight edge nodes onward, MORL-ECSO remains near 52-58 ms and is competitive with or lower than the learning and metaheuristic baselines.

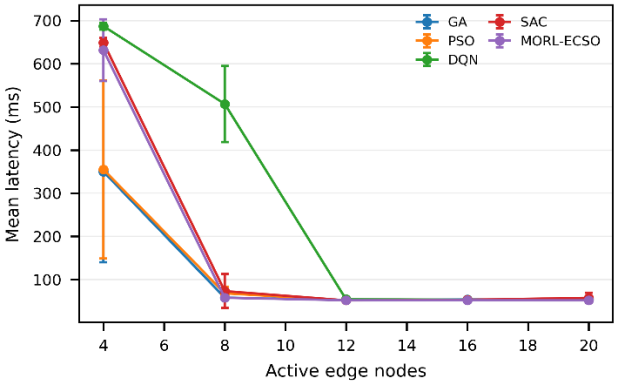

Figure 11. Mean latency under edge-node scaling; error bars show 95% confidence intervals over 30 test seeds

Throughput remains dimensionally consistent with offered load (Figure 12). At 500, 1000, 1500, and 2000 tasks/s, MORL-ECSO completes approximately 500, 998, 1495, and 1947.56 tasks/s. At the highest load, GA completes 1947.71 tasks/s. Their difference is -0.008% and is not significant after Holm correction ($p = 0.105$).

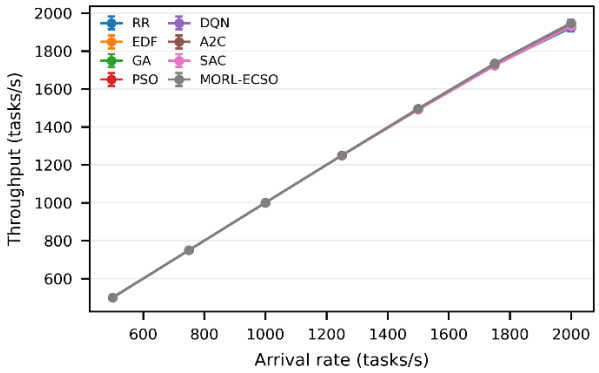

Figure 12. Completed individual-task throughput versus offered arrival rate; error bars show 95% confidence intervals over 30 test seeds

Decision overhead is reported as absolute policy-selection time (Figure 13). Across 4-20 edge nodes, MORL-ECSO requires about 0.10-0.13 ms per decision, DQN/A2C/SAC about 0.08-0.13 ms, GA about 1.1 ms, and PSO about 0.9-1.2 ms after amortizing their 5-s re-optimization intervals. RR and EDF remain below 0.002 ms. The learned policy adds measurable cost but remains an order of magnitude faster than the online metaheuristics.

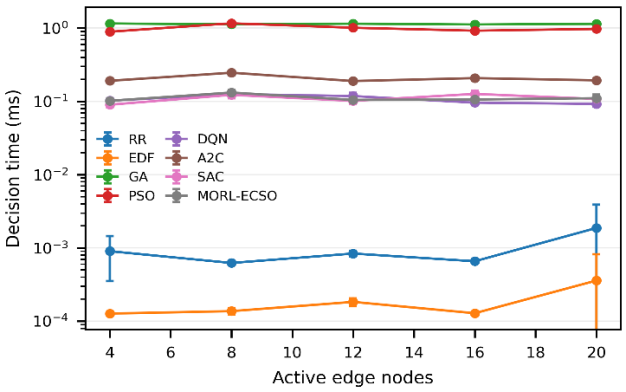

Figure 13. Absolute policy-selection time under edge-node scaling; the vertical axis is logarithmic, and error bars show 95% confidence intervals

6.7. Robustness Analysis under Dynamic Disturbances

Under increased link jitter, mean latency is 55.55 ms for MORL-ECSO and 60.74 ms for GA; success rates are 98.45% and 98.22% (Figure 14). The 95% bands widen during and shortly after the 10-s disturbance because the AR(1) jitter state decays gradually rather than resetting instantly.

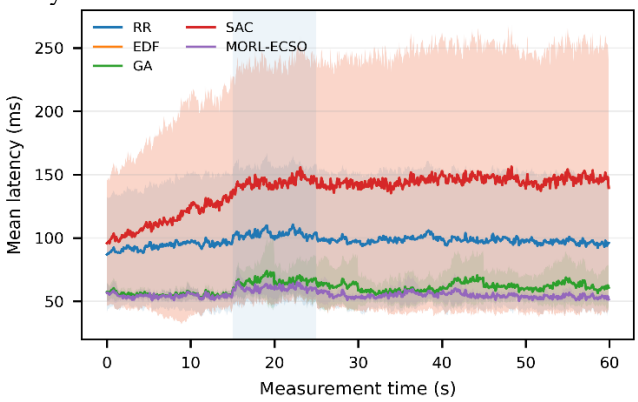

Figure 14. Mean latency during a 10-s correlated link-jitter disturbance; shaded bands show 95% confidence intervals over 30 test seeds

When 25% of edge capacity is removed, MORL-ECSO records 66.43 ms mean latency, 97.43% success, and 1772.46 tasks/s throughput over the full measurement window (Figure 15). GA records 80.42 ms, 96.56%, and 1769.70 tasks/s. Queued work is retained while capacity is reduced, so recovery after restoration reflects backlog drainage rather than deletion of failed-node tasks.

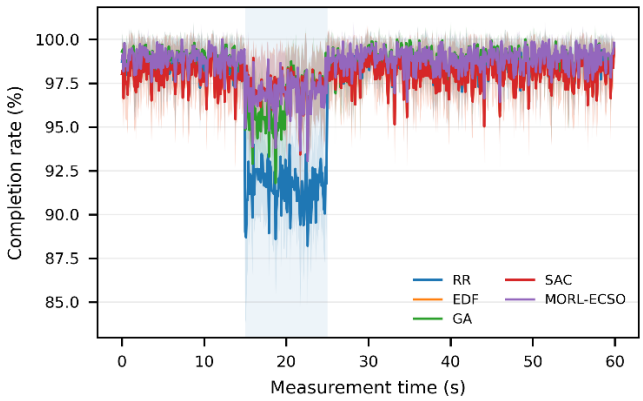

Figure 15. Completion-rate response when 25% of edge capacity is unavailable for 10 s; shaded bands show 95% confidence intervals

The 1.8x burst raises the instantaneous offered rate above 3000 tasks/s for 5 s (Figure 16). MORL-ECSO reaches 242.35 ms mean latency over the run, compared with 280.59 ms for GA and 349.38 ms for SAC/EDF. Throughput temporarily exceeds the nominal 1800-tasks/s baseline while stored work is drained, which is physically compatible with the burst input and differs from the earlier unit mismatch.

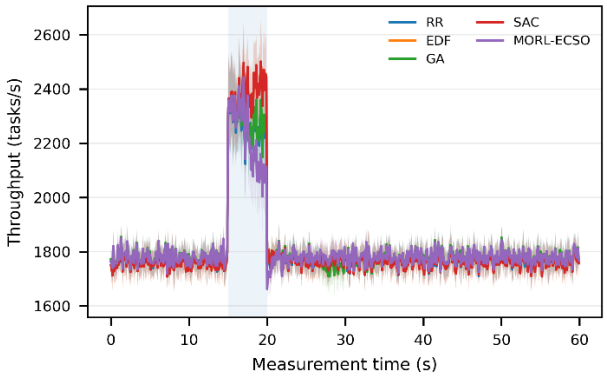

Figure 16. Throughput response to a 1.8x arrival-rate burst; shaded bands show 95% confidence intervals over 30 test seeds

6.8. Ablation Study and Mechanism Verification

Removing dynamic weighting increases mean high-load latency from 103.16 to 177.19 ms (Figure 17). Removing prioritized replay increases it to 252.96 ms and reduces success from 94.18% to 80.99%, showing that rare overload transitions matter in this training budget. Removing the enhanced state yields 190.44 ms and 86.26% success. Validation-return differences are smaller, which confirms that deployment stress tests reveal information not captured by a single aggregate return.

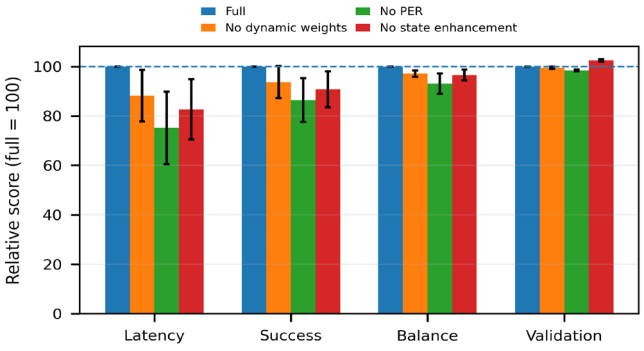

Figure 17. Ablation scores relative to the full method; deployment bars use 30 paired runs, and validation bars use three training seeds

7. Conclusion

This study develops and audits MORL-ECSO for multi-objective edge-cloud scheduling. The revised implementation uses individual-task arrival units, finite queues, correlated jitter, explicit node-failure and burst models, matched neural capacities, tuned GA/PSO budgets, and disjoint training, validation, and test streams. At 2000 tasks/s, MORL-ECSO lowers mean latency by 18.53% and raises on-time success by 0.89 percentage points relative to GA, while energy and throughput remain statistically indistinguishable. The result is therefore a latency-deadline coordination gain rather than a universal improvement in every objective.

The remaining limitations are substantive. The simulator uses aggregate tier queues rather than packet- and container-level execution traces; physical power coefficients are configured inputs; only three training initializations were used; and the four-edge-node and computation-heavy cases expose capacity limits. Future work should validate the policy on an operational testbed, model migration and cold-start costs explicitly, and test whether the paired advantages persist across hardware and workload distributions not used in validation.

References

[1] Avan A, Azim A, Mahmoud QH. A State-of-the-Art Review of Task Scheduling for Edge Computing: A Delay-Sensitive Application Perspective. Electronics. 2023;12(12):2599. https://doi.org/10.3390/electronics12122599

[2] Patsias V, Amanatidis P, Karampatzakis D, Lagkas T, Michalakopoulou K, Nikitas A. Task Allocation Methods and Optimization Techniques in Edge Computing: A Systematic Review of the Literature. Future Internet. 2023;15(8):254. https://doi.org/10.3390/fi15080254

[3] Peng P, Lin W, Wu W, Zhang H, Peng S, Wu Q, Li K. A survey on computation offloading in edge systems: From the perspective of deep reinforcement learning approaches. Computer Science Review. 2024;53:100656. https://doi.org/10.1016/j.cosrev.2024.100656

[4] Hosseinzadeh M, Azhir E, Lansky J, Mildeova S, Ahmed OH, Malik MH, Khan F. Task Scheduling Mechanisms for Fog Computing: A Systematic Survey. IEEE Access. 2023;11:50994-51017. https://doi.org/10.1109/ACCESS.2023.3277826
[5] Fan W, Zhao L, Liu X, Su Y, Li S, Wu F, Liu Y. Collaborative Service Placement, Task Scheduling, and Resource Allocation for Task Offloading With Edge-Cloud Cooperation. IEEE Transactions on Mobile Computing. 2024;23(1):238-256. https://doi.org/10.1109/TMC.2022.3219261
[6] Fan W, Liu X, Yuan H, Li N, Liu Y. Time-Slotted Task Offloading and Resource Allocation for Cloud-Edge-End Cooperative Computing Networks. IEEE Transactions on Mobile Computing. 2024;23(8):8225-8241. https://doi.org/10.1109/TMC.2024.3349551
[7] Zhang J, Ning Z, Ali RH, Waqas M, Tu S, Ahmad I. A Many-Objective Ensemble Optimization Algorithm for the Edge Cloud Resource Scheduling Problem. IEEE Transactions on Mobile Computing. 2024;23(2):1330-1346. https://doi.org/10.1109/TMC.2023.3235064
[8] Zhang J, Ning Z, Waqas M, Alasmary H, Tu S, Chen S. Hybrid Edge-Cloud Collaborator Resource Scheduling Approach Based on Deep Reinforcement Learning and Multiobjective Optimization. IEEE Transactions on Computers. 2024;73(1):192-205. https://doi.org/10.1109/TC.2023.3326977
[9] Shukla P, Pandey S. MOTORS: multi-objective task offloading and resource scheduling algorithm for heterogeneous fog-cloud computing scenario. The Journal of Supercomputing. 2024;80(15):22315-22361. https://doi.org/10.1007/s11227-024-06315-2
[10] Hosny KM, Awad AI, Khashaba MM, Mohamed ER. New Improved Multi-Objective Gorilla Troops Algorithm for Dependent Tasks Offloading problem in Multi-Access Edge Computing. Journal of Grid Computing. 2023;21(2):21. https://doi.org/10.1007/s10723-023-09656-z
[11] Yin L, Sun J, Zhou J, Gu Z, Li K. ECFA: An Efficient Convergent Firefly Algorithm for Solving Task Scheduling Problems in Cloud-Edge Computing. IEEE Transactions on Services Computing. 2023;16(5):3280-3293. https://doi.org/10.1109/TSC.2023.3293048
[12] Shen W, Lin W, Wu W, Wu H, Li K. Reinforcement learning-based task scheduling for heterogeneous computing in end-edge-cloud environment. Cluster Computing. 2025;28(3):179. https://doi.org/10.1007/s10586-024-04828-2
[13] Ramezani Shahidani F, Ghasemi A, Toroghi Haghighat A, Keshavarzi A. Task scheduling in edge-fog-cloud architecture: a multi-objective load balancing approach using reinforcement learning algorithm. Computing. 2023;105(6):1337-1359. https://doi.org/10.1007/s00607-022-01147-5
[14] Song F, Xing H, Wang X, Luo S, Dai P, Li K. Offloading dependent tasks in multi-access edge computing: A multi-objective reinforcement learning approach. Future Generation Computer Systems. 2022;128:333-348. https://doi.org/10.1016/j.future.2021.10.013
[15] Sellami B, Hakiri A, Ben Yahia S, Berthou P. Energy-aware task scheduling and offloading using deep reinforcement learning in SDN-enabled IoT network. Computer Networks. 2022;210:108957. https://doi.org/10.1016/j.comnet.2022.108957
[16] Yang W, Liu Z, Liu X, Ma Y. Deep reinforcement learning-based low-latency task offloading for mobile-edge computing networks. Applied Soft Computing. 2024;166:112164. https://doi.org/10.1016/j.asoc.2024.112164
[17] Liu J, Mi Y, Zhang X, Li X. Task graph offloading via deep reinforcement learning in mobile edge computing. Future Generation Computer Systems. 2024;158:545-555. https://doi.org/10.1016/j.future.2024.04.034
[18] Pang S, Wang T, Gui H, He X, Hou L. An intelligent task offloading method based on multi-agent deep reinforcement learning in ultra-dense heterogeneous network with mobile edge computing. Computer Networks. 2024;250:110555. https://doi.org/10.1016/j.comnet.2024.110555
[19] Xiong J, Guo P, Wang Y, Meng X, Zhang J, Qian L, Yu Z. Multi-agent deep reinforcement learning for task offloading in group distributed manufacturing systems. Engineering Applications of Artificial Intelligence. 2023;118:105710. https://doi.org/10.1016/j.engappai.2022.105710
[20] Zhou X, Liang W, Yan K, Li W, Wang KIK, Ma J, Jin Q. Edge-Enabled Two-Stage Scheduling Based on Deep Reinforcement Learning for Internet of Everything. IEEE Internet of Things Journal. 2023;10(4):3295-3304. https://doi.org/10.1109/JIOT.2022.3179231
[21] Ibrahim MA, Askar S. An Intelligent Scheduling Strategy in Fog Computing System Based on Multi-Objective Deep Reinforcement Learning Algorithm. IEEE Access. 2023;11:133607-133622. https://doi.org/10.1109/ACCESS.2023.3337034
[22] Wu H, Shen W, Lin W, Li W, Li K. End-Edge-Cloud Heterogeneous Resources Scheduling Method Based on RNN and Particle Swarm Optimization. IEEE Transactions on Network and Service Management. 2025;22(2):1664-1676. https://doi.org/10.1109/TNSM.2024.3507017
[23] Zhang Y, Tang B, Luo J, Zhang J. Deadline-Aware Dynamic Task Scheduling in Edge-Cloud Collaborative Computing. Electronics. 2022;11(15):2464. https://doi.org/10.3390/electronics11152464
[24] Azizi S, Shojafar M, Abawajy J, Buyya R. Deadline-aware and energy-efficient IoT task scheduling in fog computing systems: A semi-greedy approach. Journal of Network and Computer Applications. 2022;201:103333. https://doi.org/10.1016/j.jnca.2022.103333

Appendix A. Reproducibility and Statistical Audit

All appendix content is new in the revision. Yellow shading marks added material. The complete machine-readable versions are included in Supplementary Audit Package S1.

Appendix Table A1. Auditable simulator and workload configuration.

Parameter	Exact setting
Platform	Custom Python 3.13.5 simulator; NumPy 2.3.5; PyTorch 2.10.0; pandas 2.2.3; SciPy 1.17.0.
Hardware	Intel Xeon Platinum 8370C, 5 virtual CPU cores, 5.9 GB RAM; CPU-only execution.
Event and measurement window	$\Delta t = 0.1$ s; 10-s warm-up; 60-s measured interval.
Arrival process	Individual tasks; $N_t \sim \text{Poisson}(\lambda \Delta t)$; $\lambda \in \{500, 750, 1000, 1250, 1500, 1750, 2000\}$ tasks/s.
Task-class proportions	Latency-sensitive 0.30; computation-intensive 0.40; mixed-service 0.30.
Median CPU cycles	50, 300, and 140 million cycles by class; lognormal multipliers with σ = 0.18/0.22/0.20, clipped to [0.55, 1.70].
Median input size	0.20, 1.20, and 0.55 MB; lognormal multipliers with σ = 0.20/0.25/0.22, clipped to [0.50, 1.80].
Relative deadlines	0.18, 1.20, and 0.55 s; multiplied by U(0.80, 1.20).
High-priority probability	0.45, 0.15, and 0.25 by class; slot ratio receives $N(0, 0.015^2)$ perturbation and clipping to [0.05, 0.70].
Node capacity	Active edge nodes 4-20, each U(8,20) GHz; four cloud nodes, each U(40,80) GHz.
Finite queues	FCFS workload queues; edge capacity 1.2 s and cloud capacity 2.0 s of service demand.
Link model	Effective rates 150 Mb/s edge and 350 Mb/s cloud; class base RTTs 7/11/9 ms edge and 26/33/29 ms cloud.
Jitter model	AR(1), $\rho = 0.75$; normal innovation SD 1.5 ms, raised to 7 ms for the disturbance.
Power/transfer model	Edge 6 W idle + 26 W dynamic per node; cloud 45 W idle + 120 W dynamic per node; 35/55 nJ per bit.
Failure and burst	25% edge capacity unavailable for 10 s; burst multiplier 1.8 for 5 s; disturbance starts at measurement time 15 s.
Random seeds	Training 1000-1002; validation 2,000,000-series block; paired test 3000-3029.

Appendix Table A2. Matched algorithm architecture, training, and search settings.

Method	Architecture / capacity	Training or search budget
RR	No trainable parameters	Capacity-proportional fixed fractions; same finite queues and task stream.
EDF	No trainable parameters	Validation-tuned deadline template [0.94, 0.19, 0.60] edge fractions by class.
GA	No neural parameters	Population 40; 30 iterations; elite 20%; arithmetic crossover; mutation p = 0.35, SD = 0.07; patience 8; update every 5 s.
PSO	No neural parameters	Swarm 40; 30 iterations; $c_1 = c_2 = 1.7$; inertia 0.9→0.4; patience 8; update every 5 s.

Method	Architecture / capacity	Training or search budget
DQN	14-128-64-32-6; 12,454 parameters	Adam 3×10^{-4}; $\gamma = 0.97$; batch 128; replay 100,000; target every 100 gradient updates; ε decay in Eq. (29), τ = 12,000; 12,000 transitions.
A2C	14-128-64-32 with actor/critic heads; 12,487 parameters	Adam 3×10^{-4}; $\gamma = 0.97$; entropy 0.01; value coefficient 0.5; gradient clip 0.5; 32-step rollouts; 12,000 transitions.
Discrete SAC	Actor and twin Q: 14-64-32-6; 9,714 parameters total	Adam 3×10^{-4}; $\gamma = 0.97$; $\tau = 0.005$; automatic entropy target 0.9 ln(6); replay 100,000; 12,000 transitions.
MORL-ECSO	18-120-64-32-6; 12,302 parameters	Double DQN; Adam 3×10^{-4}; $\gamma = 0.97$; PER $\alpha = 0.6$, β: 0.4→1.0; batch 128; replay 100,000; target every 100 gradient updates; ε decay in Eq. (29), τ = 12,000; 12,000 transitions.

Appendix Table A3. Thirty-run summary at 2000 tasks/s: mean ± SD [95% confidence interval].

Algorithm	Latency (ms)	Energy (kJ/60 s)	On-time success (%)	Throughput (tasks/s)
EDF	214.65 ± 318.18 [95.84, 333.46]	89.11 ± 3.79 [87.70, 90.53]	85.55 ± 25.15 [76.16, 94.94]	1930.35 ± 45.30 [1913.43, 1947.27]
DQN	216.10 ± 234.19 [128.66, 303.55]	88.22 ± 4.12 [86.68, 89.76]	82.99 ± 19.43 [75.74, 90.25]	1936.10 ± 44.70 [1919.41, 1952.79]
A2C	234.83 ± 252.49 [140.55, 329.11]	88.19 ± 4.12 [86.65, 89.73]	82.06 ± 20.09 [74.56, 89.56]	1938.03 ± 47.53 [1920.29, 1955.78]
SAC	208.32 ± 255.75 [112.82, 303.82]	88.81 ± 3.92 [87.35, 90.28]	84.76 ± 20.66 [77.04, 92.47]	1933.76 ± 41.98 [1918.09, 1949.44]
Algorithm	Latency (ms)	Energy (kJ/60 s)	On-time success (%)	Throughput (tasks/s)
EDF	214.65 ± 318.18 [95.84, 333.46]	89.11 ± 3.79 [87.70, 90.53]	85.55 ± 25.15 [76.16, 94.94]	1930.35 ± 45.30 [1913.43, 1947.27]
DQN	216.10 ± 234.19 [128.66, 303.55]	88.22 ± 4.12 [86.68, 89.76]	82.99 ± 19.43 [75.74, 90.25]	1936.10 ± 44.70 [1919.41, 1952.79]
A2C	234.83 ± 252.49 [140.55, 329.11]	88.19 ± 4.12 [86.65, 89.73]	82.06 ± 20.09 [74.56, 89.56]	1938.03 ± 47.53 [1920.29, 1955.78]

Appendix Table A4. Paired Wilcoxon tests at 2000 tasks/s. Δ is oriented so that positive values favor MORL-ECSO; p values are Holm-adjusted within metric families.

Comparator	Latency	Energy	Success	Throughput
RR	Δ=240.975; p<0.001; r_rb=0.923	Δ=-1.422; p<0.001; r_rb=-1.000	Δ=22.358; p<0.001; r_rb=0.819	Δ=23.821; p=0.001; r_rb=0.716
EDF	Δ=111.487; p<0.001; r_rb=0.665	Δ=0.307; p=0.111; r_rb=0.458	Δ=8.626; p<0.001; r_rb=0.948	Δ=17.207; p<0.001; r_rb=0.832
GA	Δ=23.471; p<0.001; r_rb=0.935	Δ=-0.071; p=1.000; r_rb=-0.067	Δ=0.890; p=0.002; r_rb=0.639	Δ=-0.148; p=0.105; r_rb=0.342
PSO	Δ=28.342; p<0.001; r_rb=0.987	Δ=-0.118; p=1.000; r_rb=-0.101	Δ=1.445; p<0.001; r_rb=0.901	Δ=4.458; p=0.002; r_rb=0.682
DQN	Δ=112.941; p<0.001; r_rb=1.000	Δ=-0.584; p<0.001; r_rb=-0.996	Δ=11.186; p<0.001; r_rb=0.901	Δ=11.460; p<0.001; r_rb=0.751
A2C	Δ=131.670; p<0.001; r_rb=1.000	Δ=-0.615; p<0.001; r_rb=-0.970	Δ=12.119; p<0.001; r_rb=0.858	Δ=9.526; p=0.013; r_rb=0.557
SAC	Δ=105.162; p<0.001; r_rb=0.927	Δ=0.007; p=0.142; r_rb=0.415	Δ=9.423; p<0.001; r_rb=1.000	Δ=13.796; p<0.001; r_rb=0.871

Appendix Table A5. Seed-level raw values for MORL-ECSO and the strongest overall comparator, GA, at 2000 tasks/s.

Seed	L-M	L-G	E-M	E-G	S-M	S-G	T-M	T-G
3000	59.21	65.78	89.49	89.56	96.89	96.66	1938.96	1934.99
3001	53.99	53.50	85.89	86.22	97.66	97.70	1968.56	1969.36
3002	54.12	56.36	88.31	88.18	98.00	97.93	1962.69	1961.89
3003	53.92	55.29	87.50	87.04	97.23	97.49	1957.30	1963.35
3004	50.96	51.56	82.88	83.17	98.95	98.88	1991.90	1991.08
3005	61.71	128.65	91.50	91.34	95.98	92.36	1932.07	1923.83
3006	53.26	57.84	87.75	87.61	97.52	97.29	1943.31	1939.95
3007	55.01	68.64	87.20	87.59	97.70	96.74	1959.93	1949.12
3008	57.06	58.84	89.67	89.82	97.56	97.37	1960.66	1957.53
3009	58.77	57.33	87.53	86.04	96.11	97.06	1938.55	1959.98
3010	48.97	50.66	81.92	81.14	98.45	98.80	1967.08	1974.73
3011	54.72	57.29	89.20	89.32	98.70	98.31	1980.05	1972.59
3012	52.39	52.26	84.77	84.95	98.51	98.46	1987.99	1987.42
3013	59.65	65.49	91.30	91.19	96.87	96.74	1938.59	1938.14
3014	58.60	60.60	91.31	91.08	96.90	97.00	1942.90	1945.72
3015	50.74	50.78	83.18	83.62	98.85	98.62	1988.51	1984.41
3016	48.06	50.45	82.57	83.01	98.69	98.18	1983.60	1973.11
3017	58.82	62.28	91.59	91.42	97.04	97.01	1946.72	1947.16
3018	133.91	172.99	97.74	97.64	91.96	89.78	1883.67	1885.66
3019	49.93	50.62	82.92	83.23	99.15	98.87	1984.52	1978.94
3020	52.32	52.86	86.39	85.78	97.56	98.00	1951.80	1961.25
3021	58.22	67.30	91.40	91.60	97.40	96.65	1953.82	1941.23
3022	51.91	53.02	86.03	85.87	98.04	97.99	1969.81	1968.99
3023	77.39	138.30	92.99	92.94	94.84	91.95	1897.30	1897.25
3024	53.56	55.68	87.27	87.07	98.64	98.35	1981.66	1976.15
3025	61.28	71.57	89.92	90.06	96.72	95.72	1945.29	1935.90
3026	58.03	63.04	92.18	92.14	97.50	97.21	1962.23	1958.80
3027	320.49	403.58	96.01	95.34	74.36	72.50	1874.47	1870.83
3028	1132.50	1510.08	98.25	98.61	23.59	11.17	1772.64	1825.66
3029	55.36	56.34	89.49	89.45	98.09	97.89	1960.15	1956.17

Table A5 abbreviations: L = latency (ms), E = energy (kJ per 60 s), S = on-time success (%), T = throughput (tasks/s), M = MORL-ECSO, and G = GA.