Journal of Engineering, Project, and Production Management
2026, 16(6), 2026-0054

Optimizing Resource Allocation Strategies of Multi-Tenant Cloud Platforms using Reinforce Algorithms

Yue Cao
Senior Experimentalist, School of Information Engineering, Nanjing Polytechnic Institute, 188 Xinle Road, Jiangbei New District, Nanjing City, Jiangsu Province, China, E-mail: caoyue198447@126.com

Production Management
Received May 9, 2026; revised Jun 14, 2026; accepted June 18, 2026
Available online July 2, 2026

Abstract: The resource allocation strategy of multi-tenant cloud platforms lacks adaptability and is difficult to cope with dynamic load changes, resulting in a decrease in system operating efficiency and resource utilization. To improve the overall operating efficiency of the platform. This paper proposes an adaptive resource allocation optimization method based on Reinforce with a baseline, modeling resource scheduling as a sequential decision-making process and directly optimizing the allocation policy via policy gradient learning. Unlike value-function-based schedulers, the proposed method learns continuous resource allocation policies that jointly optimize throughput, response latency, and resource utilization, aligning policy optimization directly with operational efficiency objectives. Simulation-based evaluation results show that during the heavy load phase, the method achieves a resource utilization rate of 92.8%, reduces the average system response time to 1.38s, and increases the peak throughput to 483 tasks per second compared with traditional strategies.

Keywords: Cloud resource allocation; Reinforce algorithm; policy gradient optimization; multi-tenant architecture; adaptive scheduling strategy.

DOI 10.32738/JEPPM-2026-0054

1. Introduction

As the core form of modern cloud computing systems, multi-tenant cloud platforms support multi-user sharing of computing infrastructure through resource virtualization and isolation mechanisms, significantly reducing operating costs and improving resource utilization. However, the expansion of cloud computing service scale and the dynamic growth of task load have led to the cloud platform performance and Quality of Service (QoS) being seriously affected by resource allocation efficiency (Saidi and Bardou, 2023). Research attention has increasingly focused on achieving dynamic resource optimization in complex operating environments through intelligent methods, which holds both theoretical and practical value for building efficient and scalable cloud service systems (Moazeni et al., 2023; Huang et al., 2023).

Resource allocation in multi-tenant cloud platforms faces several challenges. On the one hand, the workload in cloud environments is highly dynamic and unpredictable, but traditional static or threshold-based allocation strategies are prone to resource waste and performance bottlenecks under load fluctuations (Wang et al., 2023; Mangalampalli et al., 2024). On the other hand, tenants differ in business priority, computing demand, and latency sensitivity, making performance isolation difficult to guarantee under resource contention and interference (Baburao et al., 2023). Furthermore, cloud platform resources vary across dimensions, and their multidimensional coupling relationships increase the complexity of scheduling decisions. According to existing research, traditional heuristic or rule-based strategies struggle to capture the nonlinear correlations between dynamic system states and task characteristics, resulting in reduced resource utilization and response performance (Yakubu and Murali, 2023; Mittal et al., 2024).

Researchers have tried various methods to optimize the resource scheduling performance of cloud platforms. Some studies have explored the use of linear programming and integer programming models to achieve optimal resource allocation, but these methods suffer from high computational complexity and lack real-time performance in high-dimensional state spaces (Zhou et al., 2023). Some studies have employed intelligent optimization algorithms such as genetic algorithms and particle swarm optimization to improve scheduling accuracy. However, these algorithms rely on global parameter tuning, resulting in slow convergence and poor adaptability to dynamic environments (Fu et al., 2023; Pirozmand et al., 2023; Mikram et al., 2024). Other studies have introduced deep reinforcement learning methods to abstract and learn environmental features through neural networks. However, some value function-based algorithms, such as Deep Q-Network (DQN), are unstable in continuous action spaces and difficult to apply to complex scenarios (Zhou et al., 2024; Wang et al., 2024; Hu et al., 2023). These methods have improved resource management on cloud platforms to some extent, but they generally suffer from issues such as lagging policy updates, insufficient environmental adaptability, and poor convergence stability. This

study explores three research questions to address the aforementioned limitations. First, whether the proposed Reinforce with baseline, enhanced via domain-specific state representation, reward function and action parameterization for multi-tenant scheduling, outperforms static allocation, heuristic scheduling, Q-learning and DDPG in resource utilization under dynamic loads. Second, whether this method achieves lower average latency and higher system throughput across light, medium and heavy load scenarios against the above baselines. Third, whether baseline correction, reward normalization and dynamic learning rate decay individually improve scheduling performance and whether their combined use is essential to boost resource utilization, reduce latency and raise throughput under heavy loads.

To address the lack of adaptability in resource allocation and the difficulty in coping with dynamic load changes in traditional multi-tenant cloud platforms, this paper proposes an adaptive resource allocation optimization method based on the Reinforce algorithm. The Reinforce algorithm directly optimizes the expected return using the policy gradient. It has the advantages of a simple structure, stable convergence, and ease of correlation with system performance indicators, making it suitable for dynamic and continuous decision-making scenarios such as cloud platforms. This method models the resource scheduling process as a sequential decision-making task in a reinforcement learning framework, constructs a state space based on tenant request characteristics, real-time system load, and historical allocation status, and the policy network outputs resource allocation actions based on the state. The system constructs a reward function using operational efficiency metrics such as throughput, response time, and resource utilization, enabling the strategy update process to directly target optimal efficiency. While Reinforce, as a foundational policy gradient algorithm, and the individual techniques of baseline correction, reward normalization, and learning rate decay are well-established in the reinforcement learning literature, the original contribution of this work lies in their domain-specific integration and adaptation for multi-tenant cloud resource scheduling. Specifically, the state space is constructed by jointly encoding tenant-level request heterogeneity, real-time node-level load, and historical allocation deviation across multi-scale time windows, thereby capturing the non-stationary and multi-dimensional coupling characteristics of cloud workloads. The reward function aggregates throughput, response latency, and resource utilization through a generalized mean with a Lagrangian constraint penalty, directly aligning the policy optimization objective with cloud operational efficiency rather than generic task completion. The action space is parameterized as a Dirichlet-distributed continuous allocation ratio vector, which enforces the sum-to-one constraint of resource quota shares without requiring discrete approximation. The combination of these three domain-specific design choices, together with the joint application of baseline correction, reward normalization, and exponential learning rate decay within a unified online closed-loop framework, constitutes the system-level contribution of this work and distinguishes it from prior applications of reinforcement learning to cloud scheduling.

Compared with recent advanced deep reinforcement learning scheduling methods, the proposed framework offers the following distinctions. DDPG achieves continuous action scheduling through a deterministic actor-critic structure but requires a separate critic network and experience replay buffer, which introduce parameter overhead and policy update lag in online scheduling loops where the system state changes continuously between cycles (Zhang et al., 2024). PPO and A2C rely on clipped surrogate objectives or advantage bootstrapping, which require transition storage and off-policy correction, and are structurally misaligned with the stateful online scheduling requirement (Sundar, 2025; Lahza et al., 2024). SAC maximizes entropy-based exploration, which is well-suited to stable-dynamics environments, whereas the multi-tenant scheduling environment is driven by externally imposed non-stationary load patterns, where variance reduction is more effective than entropy maximization (Bao et al., 2025). The proposed Reinforce with baseline framework avoids replay buffers, critic networks, and surrogate clipping entirely, and the three domain-specific components-multi-scale tenant-aware state representation, constrained generalized-mean reward function, and Dirichlet-parameterized action space-collectively address the structural challenges of multi-tenant scheduling that these prior methods do not explicitly target.

2. Adaptive Scheduling Mechanism Design

2.1. Overall Architecture and Innovative Modules

This paper models the multi-tenant resource allocation problem as a sequential decision-making task and solves it using a policy gradient method, forming an efficiency-oriented online closed-loop learning and execution system. Raw monitoring signals are aggregated by the state acquisition subsystem over multi-scale time windows and normalized before being encoded by the feature encoder and fed to the policy network. The policy network outputs the quota-ratio vector for tenants in each resource dimension, parameterized by a probability distribution. The action sampling module generates an allocation vector according to the specified probability distribution and submits it to the execution environment. After the environment performs the allocation, it returns to the next state and the corresponding performance index. The training process obtains discounted rewards via Monte Carlo trajectory sampling and employs Reinforce with a baseline for policy updates, where the baseline is a parameterized value function used solely for variance reduction, not for bootstrapping the policy gradient. The method, therefore, belongs to the Reinforce with baseline category rather than the actor-critic category: the policy update relies entirely on Monte Carlo returns, and the value function does not propagate temporal-difference targets into the gradient. Entropy regularization is added to the objective function to maintain exploratory behavior. (Sumiea et al., 2024; Khani et al., 2024). To reduce policy gradient variance and improve convergence stability, the system performs reward normalization and uses a parameterized value function as a baseline estimator. On the training side, norm clipping is applied to the policy gradient, and parameter updates are performed using an exponentially decaying learning rate and an adaptive first-order optimizer (Petrovska and Kuchuk, 2023; Liu et al., 2023). The expected cumulative return of a strategy is defined in Eq. (1).

$$J(\theta)=E_{\tau\sim\pi\theta}[\sum_{t=0}^{T}\gamma^{t}r_{t}] \tag{1}$$

In Eq. (1), θ represents the policy parameter vector, πθ represents the policy probability distribution parameterized by θ, τ represents the trajectory composed of the state-action sequence, γ is the discount factor, and r_t is the instantaneous reward at time step t.

The policy gradient estimation uses a Monte Carlo expression with a baseline, and an entropy regularization term is added to the target gradient to maintain policy exploration. The gradient is approximated in Eq. (2).

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T_i} \nabla_\theta \log \pi_\theta (a_t^i \mid s_t^i) A_t^i + \beta \mathbb{E}_{s\sim\mathcal{D}}[\nabla\theta \mathcal{H}(\pi_\theta(\cdot \mid s))] \tag{2}$$

In Eq. (2), N is the number of parallel sampling trajectories, a_t^i and s_t^i are the action and state of the i-th trajectory at time t, $A_t^i$is the advantage estimate, βis the entropy regularization coefficient, H(·)represents the policy entropy to encourage exploration, and Drepresents the empirical trajectory distribution.

The advantage estimation is in the form of Monte Carlo reward minus value function estimation, where the reward is defined in Eq. (3).

$$G_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k} \tag{3}$$

In Eq. (3), G_t represents the cumulative discount return starting from time step t.

Before updating the parameters, the original gradient is pruned according to its norm to ensure numerical stability. For the action space where resource quotas are proportional vectors, the policy network uses a parameterized multidimensional proportional distribution to represent the action distribution. The logarithmic form of the policy density is shown in Eq.(4).

$$\log \pi_\theta (a|s) = \log \mathrm{Dirichlet}(a;\alpha_\theta(s)) \tag{4}$$

In Eq. (4), $\alpha_\theta(s)$ represents the policy network's mapping output to the Dirichlet concentration parameter; to facilitate backpropagation, $\alpha_\theta(s)$ is obtained by shifting the network output through a translation activation function to ensure a positive value.

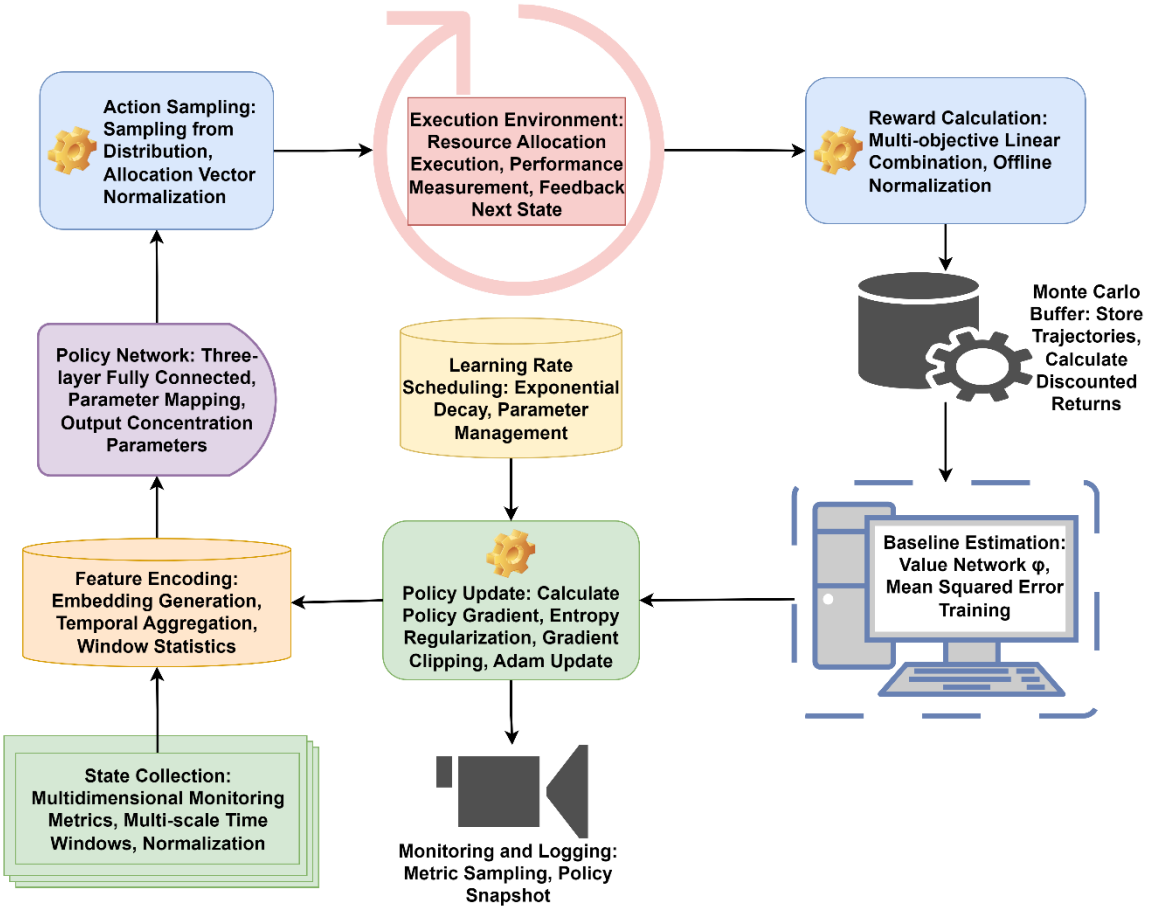

Fig. 1. Overall framework diagram of the adaptive resource allocation method

Fig. 1 shows the overall framework of this method. This fig. illustrates the data flow from state acquisition to feature encoding, and finally to policy network output and action sampling. After the action produces an effect in the execution environment, it enters the reward calculation and the Monte Carlo buffer. The baseline estimator estimates the reward from the buffer to reduce variance. The policy update node integrates policy gradient, entropy regularization, and gradient clipping to perform parameter updates. The learning rate scheduling node manages the optimization step size globally, while the monitoring module records runtime metrics and policy snapshots.

2.2. State-Action Mapping and Reward Design

After acquisition, the observations undergo pipelined preprocessing consisting of preliminary cleaning, time window aggregation, and normalization, and the resulting encoded vector serves as the input to the policy network. Action vectors are generated by the policy using parameterized probability distributions and, after mapping, satisfy the structural constraints imposed by resource quotas. The design of the reward function is the key to driving the agent to optimize its operating efficiency. It integrates the core efficiency indicators-throughput, response latency and resource utilization. By imposing a penalty term on the cost of out-of-bounds allocation, the resource allocation problem is transformed into a sequential decision optimization problem with inequality constraints (Li et al., 2023; Hou et al., 2023).

The state vector s_t adopts a hybrid representation of node-level and tenant-level. Node-level quantities include CPU (Central Processing Unit) utilization, memory usage, and node failures, which are normalized by capacity. Tenant-level quantities include task queue length, average waiting time, and historical allocation deviation. All continuous observations are calculated using a sliding window to determine the moving mean and moving variance over time, and then Z-score standardized to suppress the impact of short-term mutations on policy learning; for sparse binary events, an exponentially decaying counter is used to obtain smooth event frequencies. Action space $\mathcal{A}$ is defined as a resource quota ratio vector with tenant as the granularity, where the vector elements represent each tenant's quota share for a given resource dimension. The state vector dimension is determined by the number of node-level and tenant-level features jointly. With 50 nodes each contributing 3 node-level features and 100 tenants each contributing 3 tenant-level features, the raw observation dimension is 450. After sliding-window aggregation with a window length of 5 and Z-score normalization, the final input vector to the policy network has a dimension of 450. The action vector dimension equals the number of tenants (100), representing the resource quota share assigned to each tenant in a given resource dimension per scheduling cycle.

Table 1 lists the definitions of the main state variables used in this paper and their typical observable ranges on public datasets. The observation ranges are derived from public cluster-tracking and academic analysis reports and serve as a reference for training data generation and simulation load configuration.

To integrate multiple metrics into a single optimizable objective, the reward function adopts a generalized mean aggregation with a penalty term that weights throughput, latency, and utilization and imposes a second-order penalty on out-of-bounds assignments (Raghavendar et al., 2023). The policy network outputs a positive concentration vector for each resource dimension through a final Softplus activation layer, which maps pre-activation values to strictly positive real numbers. This concentration vector parameterizes a Dirichlet distribution over the tenant share space, from which the allocation vector is sampled during training. At evaluation time, the concentration vector is normalized by its sum to obtain the deterministic allocation vector, which corresponds to the mean of the Dirichlet distribution and ensures that the final resource-share vector sums to one with each element strictly between 0 and 1. The Dirichlet parameterization is adopted over a direct Softmax output because it defines a proper probability density over the allocation simplex, enabling the policy gradient to be computed through the log-probability of the Dirichlet distribution rather than through a deterministic mapping.

In the mapping from policy output to actual allocation, the policy network outputs a positive vector $\alpha_\theta(s_t)$ for each resource dimension, which is normalized to obtain the allocation share vector as shown in Eq. (5).

$$a_t = \frac{\alpha_\theta(s_t)}{1^T \alpha_\theta(s_t)} \tag{5}$$

In Eq. (5), a_t represents the tenant quota share vector under a certain resource dimension, $\alpha_\theta(s_t)$ represents the positive concentration vector obtained by mapping the state s_t to the policy parameter θ, and 1 represents a column vector with all elements being 1.

The construction of multi-objective returns employs a weighted aggregation of generalized (p-fold) means and introduces a constraint penalty term to explicitly characterize out-of-bounds costs. The return is defined in Eq. (6).

$$r_t = \left(\sum_{k=1}^{K} w_k \, m_{k,t}^{p} \right)^{1/p} - \mu C_t \tag{6}$$

In Eq. (6), $m_{k,t}$ represents the normalized value of the k-th type of performance, the weight w_k is a preset non-negative weight satisfying $\sum_k w_k = 1$, The parameter p controls the convexity and concavity of the aggregation to reflect the sensitivity to extreme values, μ is the constraint penalty coefficient, and C_t is the constraint penalty function at the current time. In the experiments, the three performance metrics are throughput, response latency, and resource utilization, with weights set to 0.4, 0.3, and 0.3, respectively. The aggregation parameter p is set to 2, which emphasizes the contribution of lower-performing metrics relative to the arithmetic mean and penalizes unbalanced allocations more strongly. The constraint

penalty coefficient is initialized to 0.01 and has linearly increased to 0.1 over the first 200 training epochs. The Lagrange multiplier update step size is set to 0.001.

Table 1. Definitions of state variables and action dimensions

Variable	Symbol	Definition	Publicly Observable Range (Typical Value and Source)
Node CPU Utilization (normalized to capacity)	$U_{cpu}^{(i)}$	Average CPU utilization of machine ii within the sampling time window (0−1)	The parameter ranges from 0.10 to 0.30, as observed in Alibaba cluster-trace-v2018 statistics; Google trace-2019 data shows a range of [0.1, 1] normalized to capacity
Node Memory Utilization	$U_{mem}^{(i)}$	Average memory utilization of machine ii within the sampling time window (0−1)	Most machines maintain memory usage above 0.5; analysis from Alibaba traces shows frequent occurrences
Proportion of Low-Activity Containers	—	Ratio of containers with average CPU utilization below 0.01 in the cluster	Approximately 60% of containers have CPU utilization less than 1%; observed in Alibaba trace analysis
Average Task Waiting Time	$T_{wait}^{(j)}$	Average queuing time (in seconds) from submission to start of execution for tenant j's tasks	Broad distribution; constrained by resource allocation policies; may experience delays on the order of minutes, as seen in Google trace analysis with job scheduling constraints
Historical Allocation Deviation	$\Delta_{alloc}^{(j)}$	Cumulative allocation error ratio for tenant j (historical average allocation vs. actual usage)	Observed in some cases where actual usage is significantly lower than allocated shares; Google's 2019 analysis indicates that failure-prone jobs often result in idle resources
Action (Resource Share Vector)	(a_t)	Output action vector after mapping, representing the share allocation for each tenant in a single dimension	Design constraint: elements in [0,1], $\sum_j a_{t,j} = 1$ (engineering convention).

The constraint penalty employs Lagrange multiplier evolution combined with a second-order out-of-bounds cost to ensure that online allocation respects the physical capacity limit. The penalty term is incorporated into the scalar reward signal at each scheduling step, thereby contributing to the Monte Carlo return used in the policy gradient update. The policy gradient is computed through the log-probability of the policy rather than by backpropagating through the reward function or the environment dynamics: the constraint cost enters the gradient estimate solely through its scaling effect on the return, which is the standard mechanism in Reinforce-style updates. The second-order form of the penalty ensures smoothness near the constraint boundary, producing a nonzero return signal when the allocation approaches or exceeds the capacity limit, thereby guiding the policy away from infeasible allocations through the standard policy gradient mechanism. The penalty and multiplier updates are given in Eq. (7).

$$C_t = \sum_j \left(\max\left(0, a_{t,j} \cdot R_{tot} - c_j\right)\right)^2, \lambda_{t+1} = \max\left(0, \lambda_t + \rho g(a_t)\right) \tag{7}$$

In Eq. (7), R_{tot} represents the total availability of the current resource dimension (scaling constant), c_j represents the available capacity threshold of the j-th physical unit in that dimension, $a_{t,j}$ is the nominal share allocated to this physical unit or tenant, C_t is the second-order penalty for over-allocation, λ_t is the Lagrange multiplier vector, $g(a_t)$ is the metric function for the violation of constraints, the parameter ρ is the multiplier update step size, and $\max(0, \cdot)$ guarantees the non-negativity of the multipliers.

The numerical processing of the reward and constraint terms employs online normalization and moving-window scaling to maintain the numerical stability of the gradient. The reward term $\mathbf{m_{k,t}}$ is normalized over time using the exponentially weighted moving average and standard deviation. The penalty coefficient $\boldsymbol{\mu}$ and the multiplier step size $\boldsymbol{\rho}$ are set to small values in the early stage of training and then gradually increased according to the scheduling strategy. This guides the strategy to explore first and then gradually converge to the compliant solution domain, realizing adaptive resource optimization learning with operational efficiency as the core.

2.3. Policy Network Structure and Parameter Update Strategy

The policy network is implemented as a deep neural network trained via the policy gradient method to optimize resource allocation decisions on the multi-tenant cloud platform.

The policy network adopts a three-layer fully connected neural network structure, with each layer containing several neurons to process input information extracted from the cloud platform's state space. ReLU9 activation function is used between each layer, and the softmax output function is used in the last layer to ensure the probability distribution of action selection. The input layer receives a state vector, which includes indicators such as system load, resource requests of each tenant, historical resource allocation and load status; the output layer represents the proportion allocated to each tenant in different resource dimensions (Gurusamy and Selvaraj, 2024; Zhang et al., 2023; Badri et al., 2023).

Policy network parameters are updated using Monte Carlo returns computed from the reward function, which combines throughput, response latency, and resource utilization. To reduce gradient variance and improve learning efficiency, a baseline correction technique is applied. The formula for baseline correction is shown in Eq. (8).

$$\widehat{A_t}=R_t-b(s_t) \tag{8}$$

In Eq. (8), $\widehat{A_t}$ represents the advantage function at time step t, R_t is the Monte Carlo reward, and $b(s_t)$ is the baseline in the state s_t. The baseline $b(s_t)$ is a parameterized value function fitted from the same Monte Carlo trajectory buffer. Because the policy gradient is computed from full Monte Carlo returns R_t rather than bootstrapped TD targets, the update rule remains within the Reinforce with baseline framework and does not constitute an actor-critic update. The value function baseline is implemented as a two-layer fully connected network with hidden layer sizes of 64 and 32, ReLU activation, and a scalar output. It shares no parameters with the policy network and is trained by minimizing the mean squared error between its output and the Monte Carlo returns from the same trajectory batch. The baseline network is updated once per policy update step using the same Adam optimizer with a fixed learning rate of 0.001, independent of the policy learning rate schedule.

Furthermore, to further improve the stability and convergence of training, this paper introduces a reward normalization mechanism. The reward signal is often significantly affected by system load fluctuations, leading to unstable gradient changes during training. Therefore, this paper standardizes the reward signal to maintain a relatively stable reward distribution. The reward normalization formula is shown in Eq. (9).

$$R_t'=\frac{R_t-\mu}{\sigma} \tag{9}$$

In Eq. (9), R_t' represents the normalized reward, μ and σ are the mean and standard deviation of the reward signal, respectively.

During parameter updates, the Adam optimizer is used to iteratively update the policy network. In each training iteration, the network updates its weight parameters by calculating the policy gradient at each time step and combining it with the aforementioned advantage function and reward normalization mechanism. The optimization process is shown in Eq. (10).

$$\theta_{t+1}=\theta_t+\alpha\cdot\nabla_\theta J(\theta_t) \tag{10}$$

In Eq. (10), θ_t represents the parameters of the policy network, α is the learning rate, and $\nabla_\theta J(\theta_t)$ is the gradient of the current policy. The Adam optimizer adaptively adjusts the learning rate α to ensure that gradient explosion or vanishing caused by over-updating is avoided during training.

Fig. 2 shows the evolution of policy network parameters across different training rounds and compares the performance of the policy with baseline correction and reward normalization against the traditional method. After introducing these mechanisms, the training process of the policy network becomes more stable, the parameter updates are smoother, the convergence curve fluctuates less, and the volatility of the policy output is significantly reduced. This demonstrates the effectiveness of baseline correction and reward normalization mechanisms in resource allocation tasks, thereby further improving the system's overall scheduling performance.

2.4. Learning Rate Scheduling, Convergence Guarantee, and Theoretical Analysis

In dynamic load environments, changes in the learning rate can greatly affect the effectiveness and speed of policy optimization. To ensure rapid convergence and stability of the model, this paper proposes a dynamic learning-rate decay strategy to adapt to load fluctuations (Anand and Karthikeyan, 2025; Singh, 2025).

The learning rate in this paper uses an exponential decay schedule and is adjusted based on the training epochs (Godhrawala and Sridaran, 2023). An initial learning rate is set at α_0 and after each training epoch, the learning rate decays according to Eq. (11).

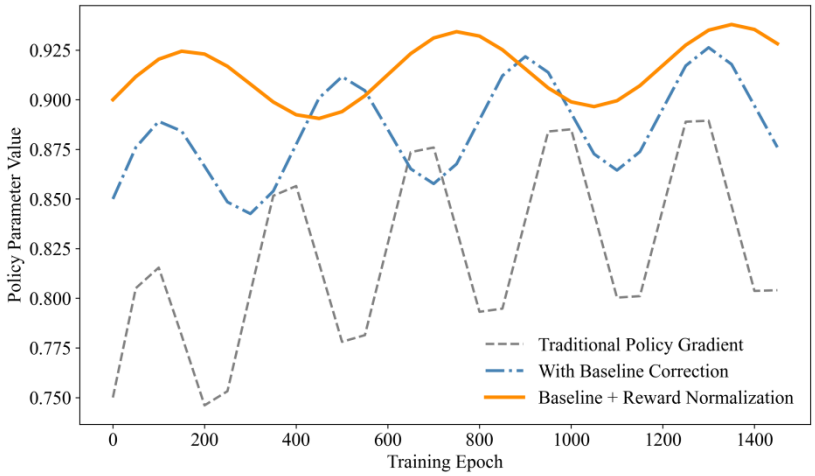

Fig. 2. Comparison of the evolution of strategy network parameters

$$\alpha_{t+1}=\alpha_0 \cdot \beta^t \tag{11}$$

In Eq. (11), α_{t+1} represents the learning rate after training round t, α_0 is the initial learning rate, β is the decay factor, and t is the current training round.

Baseline correction, reward normalization, and learning rate decay are applied jointly to ensure convergence and training stability.

The convergence analysis of the proposed method rests on three assumptions: the policy network parameterization is differentiable with respect to the policy parameters, the reward signal is bounded, and the learning rate sequence satisfies the Robbins-Monro conditions, meaning that the sum of learning rates diverges and the sum of squared learning rates converges. Under these assumptions, the Reinforce with a baseline estimator produces an unbiased estimate of the policy gradient, and the baseline reduces gradient variance without introducing bias because it depends only on the current state, not on the action taken. The exponential learning rate decay schedule satisfies the Robbins-Monro conditions when the decay factor is set sufficiently close to 1, ensuring that parameter updates remain large enough in early training to escape poor initialization regions and small enough in late training to stabilize near a local optimum of the expected return objective (Lu et al., 2024; Shahidani et al., 2023). Reward normalization further suppresses gradient variance by standardizing the reward signal to zero mean and unit variance within each update batch, controlling the effective magnitude of each gradient step independently of the absolute reward scale. Together, these three mechanisms bound the variance of the gradient estimator, satisfy the decay conditions for stochastic gradient convergence, and ensure that the policy parameters converge to a locally optimal solution of the expected return objective under the given scheduling environment structure.

To further verify the impact of different learning rate strategies on convergence performance, this paper compares several common learning rate scheduling methods. Table 2 shows the relationship between training epochs and expected reward across different learning-rate strategies.

Table 2. Comparison of training performance under different learning rate strategies

Learning Rate Strategy	Initial Learning Rate	Convergence Rounds	Average Expected Return	Return Standard Deviation
Fixed Learning Rate	0.001	1200	0.842	0.073
Exponential Decay	0.001	850	0.915	0.041
Linear Decay	0.001	930	0.902	0.050

The results in Table 2 show that the dynamic learning rate decay method is significantly better than the fixed learning rate strategy, achieving higher expected returns in fewer training epochs, and the returns tend to stabilize as the number of training epochs increases. This indicates that the learning rate decay strategy can effectively adapt to changes in system load and avoid oscillations caused by excessively high learning rates. Among them, exponential decay shows a better balance between early convergence speed and later stability. Therefore, this paper adopts exponential decay as the default learning rate scheduling method in subsequent training and experiments.

The theoretical guarantees above hold under three verifiable conditions. First, the scheduling environment is modeled as a finite-horizon Markov decision process with bounded rewards, which is satisfied when tenant task arrival rates and resource request magnitudes are drawn from distributions with finite support. Second, the policy network must be sufficiently

expressive to approximate the optimal allocation function over the state space; the two-layer, 128-unit architecture used here is well-suited to the 50-server, 100-tenant configuration but may require widening for substantially larger deployments. Third, the baseline network must be updated sufficiently frequently relative to the policy network to track the current value function, a requirement satisfied by the per-step joint update schedule used during training. When any of these conditions is not met, the convergence guarantee weakens accordingly: unbounded rewards invalidate the variance bound, an under-expressive policy network may converge to a poor local optimum, and a lagging baseline increases gradient variance and slows convergence.

3. Experimental Design and Environment Configuration

3.1. Simulation Platform Architecture and Dataset Selection

The evaluation uses a trace-driven simulator developed in Python and SimPy that replicates a real-world cloud environment comprising 50 physical servers, each equipped with 32-core Intel Xeon processors and 128 GB of memory. Following mainstream cloud deployment specifications, the simulator employs KVM for virtual machine partitioning, VXLAN and Calico for tenant network isolation and Kubernetes for resource orchestration, defining the simulation's capacity limits, isolation rules and scheduling granularity. All operations run on the SimPy discrete-event framework rather than on physical hardware, enabling controllable, reproducible tests using workloads and resource constraints derived from public cluster traces. The platform hosts 100 independent tenants, each with its own task queue, where task arrivals and resource requests follow specific probability distributions. Its monitoring module collects real-time CPU utilization, memory usage and task queue length at second-level intervals and feeds the data into the policy network to generate resource allocation decisions. With a 5 second simulation cycle and a total runtime of 24 hours, the setup unifies the resource domain for model training and inference and guarantees consistent system dynamics, thereby verifying the engineering feasibility of the algorithm's dynamic decision-making in multi-tenant cloud scenarios.

The task load data is derived from two publicly available cluster trace datasets: Google Cluster Data 2019 and Alibaba Cluster Trace 2018. From the Google trace, job submission records with incomplete CPU or memory request fields are removed, and the remaining records are used to fit the CPU-to-memory ratio distribution. From the Alibaba trace, container-level usage records with zero CPU allocation are excluded, and task durations are truncated at the 99th percentile to remove outliers caused by infrastructure anomalies. The two filtered datasets are merged into a unified schema with fields for arrival time, CPU request, memory request, and task duration. Task inter-arrival times are fitted to a Poisson distribution, the CPU-to-memory ratio is fitted to a log-normal distribution, and task durations retain the heavy-tailed empirical distribution of the merged dataset. A hybrid task flow is generated by sampling from these fitted distributions to produce a workload that simultaneously reflects the short-task-intensive characteristics of the Google trace and the long-task-continuous characteristics of the Alibaba trace. The Google Cluster Data 2019 is publicly accessible at https://github.com/google/cluster-data, and the Alibaba Cluster Trace 2018 is publicly accessible at https://github.com/alibaba/clusterdata/blob/master/cluster-trace-v2018. The arrival time interval of tasks follows a Poisson distribution. The CPU-to-memory ratio of the task resource request vector follows a log-normal distribution fitted from the trace data, and the task duration retains the heavy-tailed distribution characteristics of the original dataset to reflect the true tenant heterogeneity. The load intensity is controlled by adjusting the task generation rate to test the algorithm's performance under three operating conditions: the overall utilization rate of the system in the light load range is set to about 50%, in the medium load range to 70%, and in the heavy load range to 90%. Tasks are unevenly distributed among tenants, exhibiting a Zipf-shaped task submission frequency distribution that reflects the business differences between large and small tenants in practice.

The state sampling module aggregates runtime data within a sliding time window and applies feature-wise standardization to mitigate scale-induced instability during policy network training. The resource scheduler is deployed in the form of a reinforcement learning agent. In each scheduling cycle, it outputs the CPU and memory quota allocation ratio at the tenant level based on the state vector. The policy output is submitted to the resource allocation module for execution after being verified by the normalization layer (Mangalampalli et al., 2024). After each round of scheduling, the environment provides feedback on real-time performance indicators, including system throughput, average response latency, and resource utilization, to form the reward signal input for the reward calculation function. The entire training process adopts an online update method to ensure that the parameter adjustment of the policy network and the environment response are synchronized. The training data and evaluation data are strictly isolated across different experimental stages to avoid evaluation bias from data leakage.

To ensure the reproducibility and scientific rigor of the experiments, all experimental configurations are executed under uniform hardware and virtualization conditions, and the random seed remains consistent throughout task generation and network initialization. Each experimental condition is independently repeated 30 times with different random seeds to obtain statistically reliable performance estimates. All reported mean values, standard deviations, and 95% confidence intervals are derived from these 30 independent runs. The 95% confidence intervals are computed using the t-distribution with 29 degrees of freedom. An experimental logging system records system status, policy outputs, immediate rewards, and final performance metrics to support subsequent statistical analyses and error analysis. Load generation and simulation execution are implemented in Python with the SimPy framework, while the policy model is trained and deployed with PyTorch. The training server is equipped with an NVIDIA Tesla V100 GPU, and model parameter updates are synchronously saved after each training round.

3.2. Parameter Setting and Comparison Scheme

In this paper, the policy network adopts a three-layer fully connected structure. The input layer dimension is consistent with the state space; the number of hidden layer nodes is set to 128 and 64; the activation function is ReLU; and the output layer

uses Softmax to generate the resource allocation probability distribution for each tenant. The discount factor γ is set to 0.95 to balance short- and long-term returns; the strategy update batch size is 256; and the optimizer is Adam. The initial learning rate is set to 1×10^{-3}, and an exponential decay mechanism is applied to the dynamic load, with a decay coefficient of 0.98. Reward normalization and baseline correction are enabled simultaneously to reduce policy gradient variance and ensure training stability under high dynamic load conditions. A global update is performed every 200 steps to ensure the statistical reliability of the reward signal. The 24-hour simulation is divided into a training phase covering the first 20 hours and an evaluation phase covering the remaining 4 hours. Policy parameters learned during the training phase are frozen before the evaluation phase begins, and all reported performance metrics are measured exclusively during the evaluation phase to ensure that the results reflect generalization performance rather than training-phase behavior.

Four baseline methods are included for comparison.

a. Static resource allocation strategy: this strategy allocates computing resources at a fixed ratio, representing the basic method of traditional cloud platforms, but lacks adaptive scheduling capabilities.

b. Heuristic scheduling strategy: it allocates resources based on a comprehensive score of resource utilization and task waiting time. It is a common rule-driven method in actual deployments and has a certain degree of dynamism, but is difficult to sustain.

c. Q-learning-based reinforcement scheduling method: it achieves resource optimization through value function learning in the discrete action space and is a classic baseline algorithm in the field of reinforcement learning.

d. DDPG-based continuous action scheduling method: Deep Deterministic Policy Gradient (DDPG) is an actor-critic algorithm designed for continuous action spaces. It maintains a deterministic policy network (actor) and a value-estimation network (critic), and stabilizes training through experience replay and target-network mechanisms. In the resource scheduling task, the actor network outputs a deterministic continuous allocation vector based on the current system state, and the critic network evaluates the quality of the allocation action to guide policy updates. DDPG is introduced as a baseline for actor-critic methods in continuous action spaces, to systematically distinguish the structural differences and performance boundaries between policy gradient and actor-critic methods in multi-tenant resource scheduling tasks.

These comparative methods cover the full spectrum from static policy to rule-based adaptation, discrete reinforcement learning, and continuous action actor-critic learning, reflecting the mainstream implementation approaches of current multi-tenant cloud resource scheduling.

All experiments are performed under a uniform environment configuration, with a fixed random seed to ensure the reproducibility of results. During the experiment, the model's reward trend, convergence speed, and fluctuation amplitude are monitored at different training stages to assess the impact of learning-rate decay and baseline-correction strategies on convergence. To facilitate quantitative comparison, the hyperparameter configurations and key scheduling features of different methods are listed in Table 3.

Table 3 shows that different algorithms differ significantly in their expressive power and training mechanisms for resource allocation decisions. The Reinforce method exhibits greater flexibility and stability in the continuous action space, effectively adapting to complex load-fluctuation scenarios in multi-tenant cloud platforms.

4. Performance Evaluation and Result Analysis

4.1. Resource Utilization Performance

Resource utilization is a core indicator of a multi-tenant cloud platform's efficiency, directly reflecting the extent to which available computing resources are productively used. This section evaluates the resource utilization of each scheduling method under three load conditions to assess their dynamic adaptation behavior. Three load levels are defined for experiments: light, medium and heavy loads, with the task arrival rate set to 20 req/s, 60 req/s and 120 req/s per tenant, respectively. For the platform supporting 100 tenants, the corresponding overall system task arrival rates are around 200, 600 and 1200 tasks/s. The configured tenant-level arrival intensities align with the overall system input rates. Metrics such as throughput reflect the overall task-processing capability relative to system-wide input loads, whereas resource utilization and latency characterize scheduling performance under per-tenant arrival intensities. Compared with static allocation, heuristic scheduling and Q-learning-based reinforcement learning methods, the algorithm's response efficiency and allocation balance in resource utilization under multi-tenant concurrent requests are analyzed.

Experimental results show that the Reinforce algorithm maintains high resource utilization across various load conditions and, even under increased system pressure, maintains stable allocation performance and control of fluctuations. Fig. 3 shows that during the light load phase, the utilization rates of each method are: static allocation 75% ± 2.1%, heuristic 82.1% ± 1.8%, Q-learning 86% ± 1.4%, DDPG 88.5% ± 1.2%, with the Reinforce algorithm reaching the highest at 91% ± 1.0%. As the load increases, the Reinforce algorithm achieves an average utilization rate of 92.3% ± 0.8% during the medium load phase, approximately 5 percentage points higher than Q-learning ± 1.2%; during the heavy load phase, the Reinforce algorithm still maintains a high utilization rate of 92.8% ± 0.6%, while heuristic scheduling shows a decrease to 80.7% ± 1.4%, and static allocation only rises to 80.2% ± 1.7%. Under the same heavy-load condition, DDPG achieves a resource utilization rate of 90.1% ± 0.8%, which is higher than Q-learning by ± 1.0% but lower than the Reinforce method by 2.7 percentage points. This is consistent with the observation that DDPG's replay-based update mechanism introduces delayed policy response during rapid load transitions. The narrowing standard deviations across all methods from light to heavy load indicate that scheduling behavior stabilizes under sustained pressure, with the Reinforce method consistently exhibiting the

lowest variance across all three load levels, confirming its superior stability relative to all baselines. The smoothness of the curve confirms the stability of the policy update, indicating that the Reinforce algorithm strikes an effective balance between computing resources and multi-tenant contention, keeping the system in an ideal state of high utilization and low fluctuation. All reported values represent means across 30 independent runs. Error bars in Fig. 3 denote 95% confidence intervals derived from the t-distribution with 29 degrees of freedom. This verifies the adaptive advantages of the Reinforce algorithm in a multi-tenant cloud environment, providing fundamental support for the comprehensive improvement of subsequent system operating efficiency.

Table 3. Main experimental parameters and control algorithm configurations

Parameter Category	Parameter Name	Value or Configuration	Description
Policy Network	Hidden Layer Size	128 / 64	Controls network capacity and generalization performance
	Activation Function	ReLU / Softmax	Forward propagation and output distribution generation
Learning Parameters	Discount Factor γ	0.95	Balances immediate and long-term rewards
	Initial Learning Rate	1×10^{-3}	Initial value for dynamic decay
	Learning Rate Decay Rate	0.98	Controls convergence speed
	Batch Size	256	Ensures stability of gradient estimation
Scheduling Algorithm	Static Allocation Strategy	Fixed Ratio Method	Does not rely on load feedback
	Heuristic Strategy	Resource – Waiting Time Scoring Method	Adjusts based on thresholds
	Q-learning Strategy	Discrete Action-Value Function	Tabular reinforcement learning method
	Proposed Method (Reinforce)	Policy Gradient Optimization	Continuous action policy learning
	DDPG Strategy	Actor-Critic with Experience Replay	Continuous action deterministic policy

4.2. Average Response Latency Performance

Average response latency is an important dynamic indicator of a cloud platform's operational efficiency and service quality. Changes in response latency directly reflect the scheduling algorithm's effectiveness in task-to-resource matching. In multi-tenant platforms, tenant heterogeneity and stochastic task arrivals cause latency to accumulate under high load when allocation decisions are slow to adapt. The proposed method addresses this by optimizing allocation actions using policy gradients, maintaining dynamic resource balance during high-load phases, and reducing queuing delays.

Compared to static allocation and heuristic scheduling, the Reinforce algorithm, through a continuous policy update mechanism, establishes a stable state-action mapping in the later stages of training, significantly reducing the time from task submission to completion. Fig. 4 shows a comparison of the average response time of different algorithms under three load levels: light load, medium load, and heavy load, demonstrating the effect of policy learning on system latency optimization.

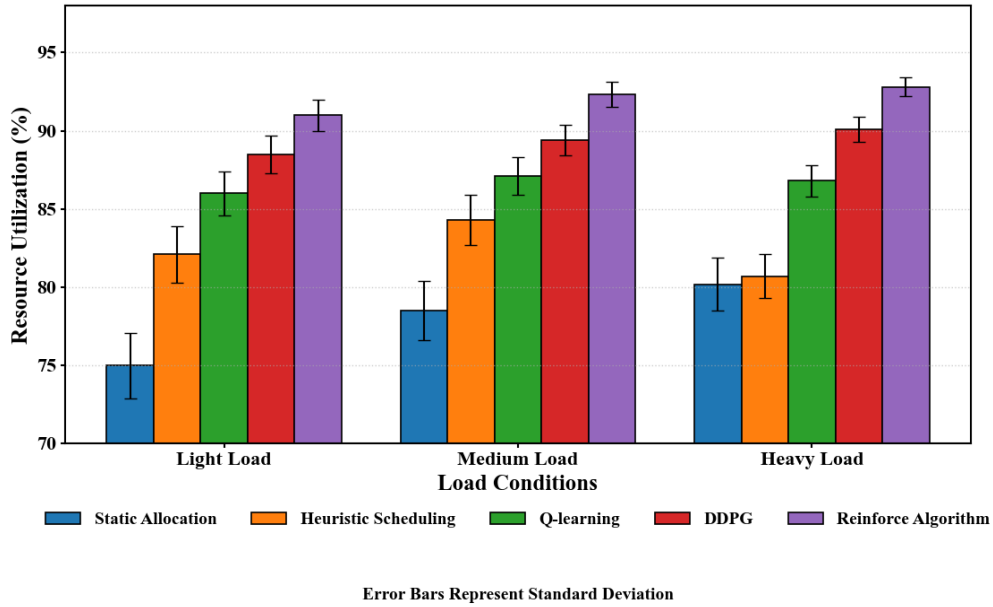

Fig. 3. Resource utilization of different scheduling methods under light-load, medium-load, and heavy-load conditions

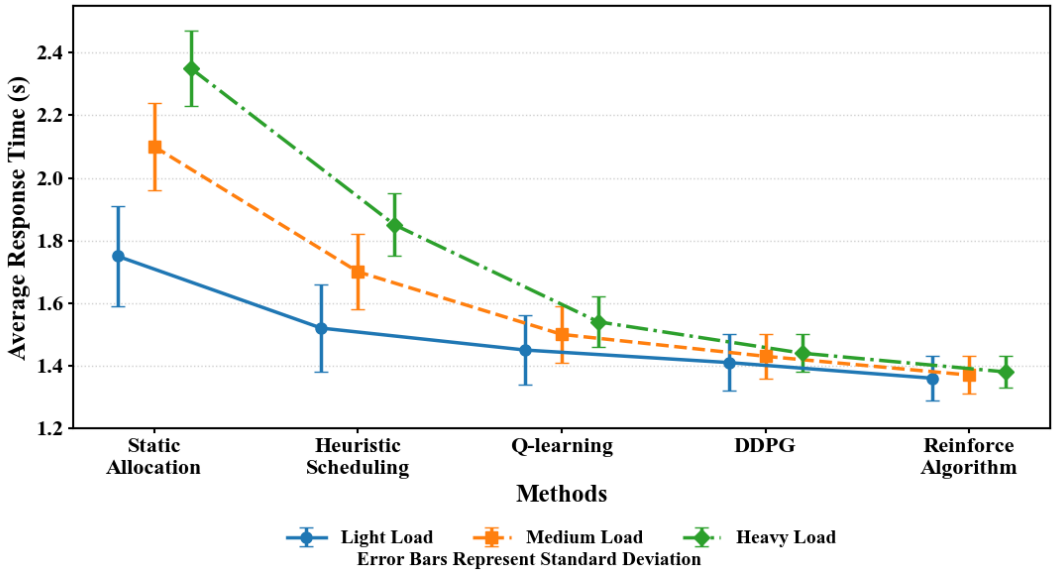

Fig. 4. Average response latency of scheduling methods under light, medium, and heavy load

As shown in Fig. 4, the differences in response time among the algorithms gradually become apparent as load intensity increases. The static allocation method has an average response time of 2.35s ± 0.12s under heavy load, while the heuristic scheduling method is 1.85s ± 0.10s, the Q-learning-based method is 1.54s ± 0.08s, DDPG is 1.44s ± 0.06s, and the Reinforce algorithm remains 1.38s ± 0.05s under the same conditions. The Reinforce method exhibits the smallest standard deviation across all load levels, indicating that its policy gradient update mechanism sustains more consistent latency control than all baselines under dynamic workload conditions. The curve trend shows that during the task-intensive phase, traditional methods are prone to latency accumulation, while the Reinforce algorithm continuously adjusts the resource allocation ratio through a reward feedback mechanism, maintaining a shorter response cycle under high concurrency. All values represent means across 30 independent runs; error bars in Fig. 4 denote 95% confidence intervals derived from the t-distribution with 29 degrees of freedom.

4.3. System Throughput Capacity

System throughput directly reflects the cloud platform's task processing capacity per unit time and is a key indicator for evaluating overall operational efficiency and load-bearing capacity. This metric is closely related to resource utilization and response latency, reflecting the system's task-execution efficiency under load. The proposed method continuously adjusts per-tenant resource ratios via policy gradient optimization, thereby maintaining stable task-processing throughput across varying load levels.

Fig. 5 shows the trend of system throughput for various scheduling methods under different task arrival rates. As shown in Fig. 5, when the task arrival rate is in the range of 200 to 600 tasks per second, the throughput growth of the static allocation method gradually slows down, maintaining at 350 tasks per second ± 13 during the high load stage [95% CI: 345.1, 354.9]; the heuristic scheduling method reaches 420 tasks per second ± 11 [95% CI: 415.9, 424.1]; the Q-learning-based method reaches 460 tasks per second ± 8 [95% CI: 457.0, 463.0]; and DDPG reaches 472 tasks per second ± 6 [95% CI: 469.8, 474.2]. The Reinforce algorithm achieves a maximum throughput of 483 tasks per second ± 5 [95% CI: 481.1, 484.9] under the same conditions. The non-overlapping confidence intervals between the Reinforce method and all baselines at the highest arrival rate confirm that the throughput advantage is statistically reliable across repeated runs. The algorithm maintains high resource scheduling efficiency even under task-intensive conditions, indicating that the policy gradient update achieves more thorough exploration in allocating actions within the action space. All values represent means across 30 independent runs; error bars in Fig. 5 denote 95% confidence intervals derived from the t-distribution with 29 degrees of freedom

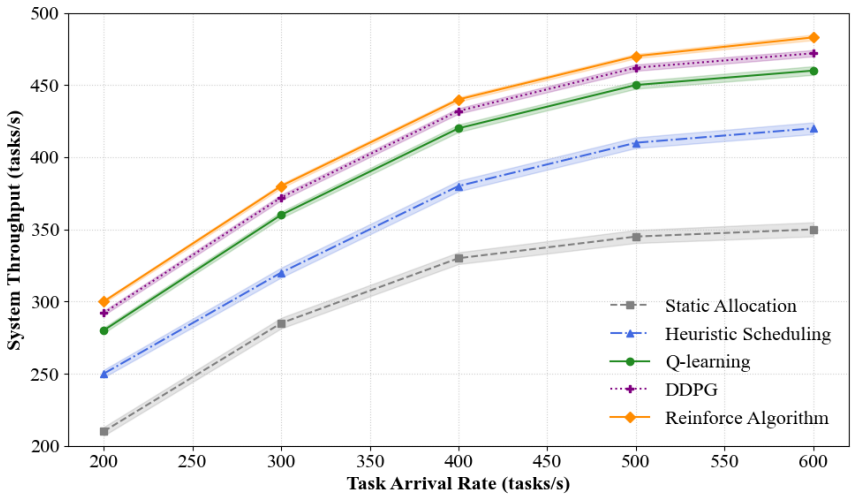

Fig. 5. Throughput comparison of different scheduling methods under different task arrival rates

4.4. Convergence Performance and Training Stability

Convergence performance determines how efficiently an algorithm learns in dynamic environments and how stably it operates after deployment. The proposed method refines its allocation strategy across training rounds to maximize expected cumulative return.

To analyze the convergence speed and stability of the algorithm during training, this paper compares the performance of introducing a baseline correction and combining it with a dynamic learning-rate decay mechanism. Fig. 6 shows the variation of expected reward with training iterations under different scheduling strategies and reflects the convergence characteristics observed during simulation training.

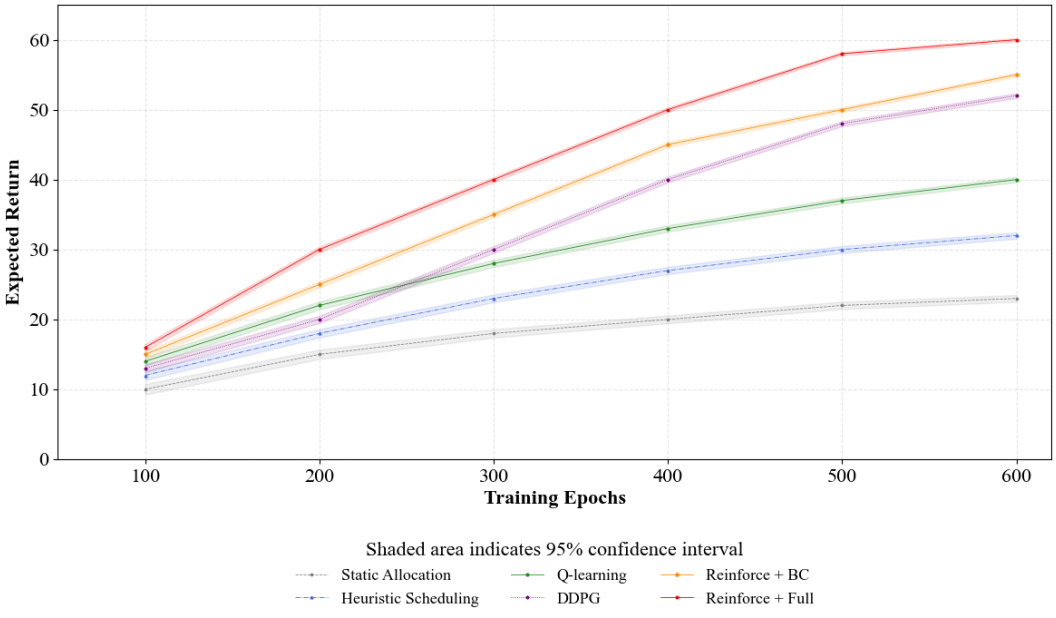

Fig. 6. Convergence performance of different scheduling strategies during training

As shown in Fig. 6, the traditional static allocation method exhibits slow returns in the early stages of training, with the improvement stabilizing after 400 rounds, demonstrating its limitation in adapting to complex load fluctuations. Heuristic scheduling shows some improvement, but the returns still fail to reach a satisfactory level of stability. In contrast, the algorithm incorporating baseline correction and learning rate decay (Reinforce) achieves a return of 60 ± 0.6 [95% CI: 59.78, 60.22] at 600 rounds, approximately 9% higher than the model with baseline correction only (55 ± 0.8 [95% CI: 54.70, 55.30]), 15% higher than DDPG (52 ± 0.9 [95% CI: 51.66, 52.34]), and 50% higher than Q-learning (40 ± 1.0 [95% CI: 39.63, 40.37]). The Reinforce method consistently exhibits the narrowest confidence intervals across all training stages, indicating that the combination of baseline correction and learning rate decay reduces inter-run variance and yields more reproducible convergence behavior than any of the baselines. The return curve shows a smoother, continuously increasing trend, indicating that the algorithm effectively eliminates training noise and improves convergence speed during optimization of the resource allocation strategy. More importantly, through the dynamic learning rate decay mechanism, the algorithm maintains stable convergence under high-load conditions, allowing the expected return to reach the ideal level in less time. All values represent means across 30 independent runs; error bars in Fig. 6 denote 95% confidence intervals derived from the t-distribution with 29 degrees of freedom.

Therefore, the optimization strategy proposed in this paper not only improves the efficiency of resource allocation but also effectively enhances the convergence performance and stability of the algorithm in multi-tenant cloud platforms when dealing with complex load changes, laying a reliable algorithmic foundation for the continuous improvement of platform operating efficiency.

4.5. Ablation Study

To quantitatively assess the contribution of each component in the proposed method, ablation experiments are conducted under heavy-load conditions. Three incremental configurations are evaluated: the base Reinforce model without any enhancement (Base), the model with baseline correction added (Base + BC), the model with both baseline correction and reward normalization (Base + BC + RN), and the full model incorporating all three mechanisms, including dynamic learning rate decay (Base + BC + RN + LRD). Each configuration is trained to convergence under identical hyperparameter settings, and performance is measured after the reward curve stabilizes, in terms of resource utilization, average response time, and throughput. Table 4 presents the results.

Table 4. Ablation study results of different component configurations under heavy-load conditions

Configuration	Resource Utilization (%)	Average Response Time (s)	Throughput (tasks/s)
Base (Reinforce only)	87.3 ± 1.5	1.62 ± 0.11	451 ± 10
Base + BC	90.2 ± 1.0	1.51 ± 0.08	464 ± 8
Base + BC + RN	91.6 ± 0.8	1.44 ± 0.06	475 ± 6
Base + BC + RN + LRD (Full)	92.8 ± 0.6	1.38 ± 0.05	483 ± 5

The results in Table 4 show that each added component produces measurable improvement across all three metrics. Baseline Correction (BC) reduces response time from 1.62 s to 1.51 s and raises resource utilization by 2.9 percentage points, confirming its role in reducing gradient variance and stabilizing the policy update direction. The addition of Reward Normalization (RN) further reduces response time to 1.44 s and increases throughput to 475 tasks/s, demonstrating that suppressing reward-signal fluctuations due to dynamic load changes promotes more consistent gradient updates. The full model with dynamic Learning Rate Decay (LRD) achieves the best performance across all indicators, with resource utilization reaching 92.8% and throughput peaking at 483 tasks/s. The progressive improvement across configurations confirms that the three mechanisms function through distinct and complementary roles in the training process, and that their joint application is necessary to achieve the full scheduling performance of the proposed method.

4.6. Statistical Significance Analysis

To verify that the performance differences observed across scheduling methods are statistically reliable rather than attributable to random variation, Welch's two-sample t-test is applied to compare the proposed Reinforce method against each baseline under heavy-load conditions across three metrics: resource utilization, average response latency, and system throughput. Welch's t-test is adopted in place of the standard equal-variance t-test because it does not assume equal variances across groups, which is appropriate given the differing standard deviations observed among methods. All tests are two-tailed with a significance threshold of $p < 0.05$. Each method is evaluated over 30 independent runs, and the test statistics are computed from the per-run sample means. Table 5 presents the t-statistics, degrees of freedom, and p-values for each pairwise comparison.

As shown in Table 5, all pairwise comparisons yield p-values well below 0.001, confirming that the performance advantages of the proposed method over all four baselines are statistically reliable across all three metrics. The t-statistics for comparisons against Static and Heuristic are substantially larger in magnitude than those against Q-learning and DDPG, reflecting the greater absolute performance gap between rule-based methods and the proposed approach. The comparison

against DDPG yields the smallest t-statistics among the four baselines, consistent with the narrower numerical gap between the two continuous-action methods, yet the differences remain statistically reliable at $p < 0.001$. These results confirm that the improvements reported in Sections 4.1 through 4.3 are not attributable to random variation in the simulation environment.

4.7. Tenant Fairness and Performance Isolation

Tenant fairness and performance isolation are critical to the operational stability of multi-tenant cloud platforms, as inequitable allocation leads to resource contention and cross-tenant performance interference. This section compares the performance of different scheduling methods in terms of tenant resource allocation, response time, and resource preemption, analyzing the advantages of the proposed method in terms of tenant fairness and performance isolation. Fig. 7 illustrates the balance of resource allocation among tenants across different scheduling algorithms, as well as the differences in response time and resource consumption under varying load conditions.

Table 5. Welch's t-test results: reinforce vs. baselines under heavy-load conditions (n = 30)

Baseline	Metric	t-statistic	df	p-value	Significance
Static	Resource Utilization (%)	38.282	36.1	7.81e-31	***
Heuristic	Resource Utilization (%)	43.511	39.3	7.25e-35	***
Q-learning	Resource Utilization (%)	28.180	47.5	2.69e-31	***
DDPG	Resource Utilization (%)	14.789	53.8	1.35e-20	***
Static	Avg. Response Latency (s)	-40.869	38.8	1.71e-33	***
Heuristic	Avg. Response Latency (s)	-23.025	42.6	1.11e-25	***
Q-learning	Avg. Response Latency (s)	-9.289	48.7	2.35e-12	***
DDPG	Avg. Response Latency (s)	-4.208	56.2	9.38e-05	***
Static	Throughput (tasks/s)	52.301	37.4	1.41e-36	***
Heuristic	Throughput (tasks/s)	28.558	40.5	1.89e-28	***
Q-learning	Throughput (tasks/s)	13.353	48.7	6.83e-18	***
DDPG	Throughput (tasks/s)	7.714	56.2	2.24e-10	***

As shown in Fig. 7, the static resource allocation scheme exhibits significant performance limitations during load increases: its resource utilization consistently remains below 85% with a standard deviation of up to 2.0%, and the peak response time reaches 2.6s $\pm$ 0.11s. This indicates a significant bias in resource allocation among tenants, with some receiving substantially more resources than others, leading to greater unevenness in system load and response time differences. The heuristic scheduling method offers a slight improvement, achieving a maximum resource utilization of 87% $\pm$ 1.3%, but the response time remains above 1.7s $\pm$ 0.13s under light load, suggesting that while it alleviates resource imbalance, it struggles to achieve resource isolation between different tenants. DDPG achieves resource utilization of 86% $\pm$ 1.1%, 88% $\pm$ 0.9%, and 90% $\pm$ 0.8% under light, medium, and heavy load, respectively, with corresponding response times of 1.45s $\pm$ 0.08s, 1.47s $\pm$ 0.07s, and 1.50s $\pm$ 0.06s, outperforming heuristic scheduling in both dimensions but showing marginally higher response latency than the Reinforce method across all load levels. In contrast, the resource allocation method based on the Reinforce algorithm, incorporating reward normalization, baseline correction, and dynamic learning-rate decay, demonstrates superior performance across all three load levels. Its resource utilization reaches 80% $\pm$ 1.0%, 85% $\pm$ 0.8%, and 90% $\pm$ 0.7% under light, medium, and heavy load, respectively, with response times of 1.40s $\pm$

0.07s, 1.60s ± 0.06s, and 1.80s ± 0.05s. The Reinforce method maintains the smallest standard deviations in both resource utilization and response time across all load levels, confirming that its policy optimization mechanism produces more consistent tenant-level scheduling behavior than all baselines. All reported values represent means across 30 independent runs. The lower standard deviations of the Reinforce method in both resource utilization and response time across all load levels indicate more consistent quota distribution across tenants, providing quantitative evidence that the proposed policy optimization mechanism reduces inter-tenant allocation disparity compared to all baselines. All reported values represent means across 30 independent runs.

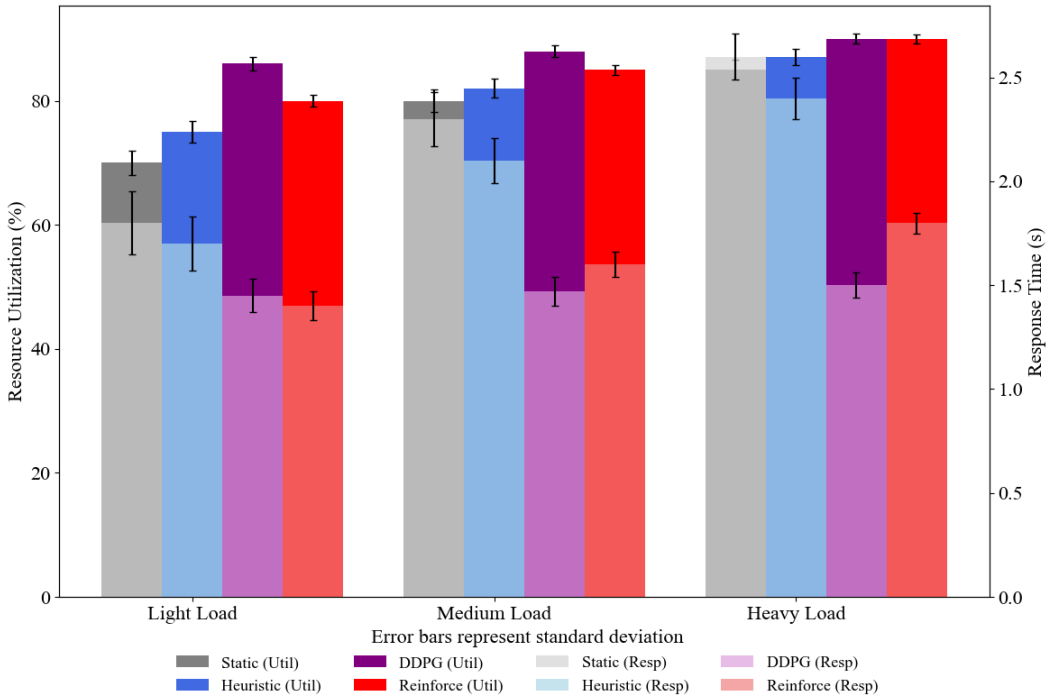

Fig. 7. Comparison of Tenant Fairness/Isolation

5. Limitations

The evaluation in this paper is conducted entirely within a trace-driven simulation environment, and no real-world cloud deployment has been performed. The simulation reproduces the capacity constraints and scheduling granularity of the target infrastructure using public cluster traces, but does not capture runtime factors present in production systems, such as hardware heterogeneity, operating system scheduling jitter, and hypervisor overhead. The actual policy inference latency introduced at each scheduling decision cycle has not been measured, and whether this overhead remains within acceptable bounds in a production orchestration loop has not been verified.

The baseline comparison covers static allocation, heuristic scheduling, Q-learning, and DDPG. Methods such as PPO, A3C, and SAC are not included, and the conclusions about relative performance advantages are therefore limited to the evaluated baseline set.

The simulation does not model node failure, workload distribution drift over time, or tenant churn. The policy is trained and evaluated on stationary load distributions derived from the two public traces, and its behavior under non-stationary or failure-induced workload patterns has not been assessed. Performance under these conditions may differ from the reported results.

The framework is validated at a scale of 50 servers and 100 tenants. The policy network architecture and per-step update schedule have not been tested at substantially larger scales, and computational feasibility at production scale remains an open question for future work.

6. Conclusion

This paper constructs an adaptive resource allocation optimization framework based on the Reinforce algorithm. This framework models the scheduling process as a sequential decision-making task in reinforcement learning, directly optimizing the allocation strategy via the policy gradient method and constructing a closed-loop learning mechanism for state-action-reward. In the state-space design, the algorithm integrates multi-dimensional information, such as tenant demand, system load, and historical allocation states, and uses Monte Carlo rewards and baseline correction to jointly control gradient variance. Simulation-based evaluation results show that under heavy load conditions across 30 independent runs, this method achieves a mean resource utilization of 92.8% ± 0.6%, a mean average system response time of 1.38 s ± 0.05 s, and a mean peak throughput of 483 ± 5 tasks per second, with all performance advantages over the four baselines confirmed as statistically reliable at $p < 0.001$ by Welch's t-test. Compared with the four baseline methods evaluated in this study, this

method exhibits higher resource utilization, lower response latency, higher throughput, and more consistent inter-tenant allocation as reflected by lower standard deviations in tenant-level resource utilization and response time under simulation conditions, with all performance advantages confirmed as statistically reliable at $p < 0.001$ across 30 independent runs. The performance improvement is mainly attributed to the efficiency-oriented policy optimization mechanism and continuous resource allocation strategy adopted in this study. The contribution of this work is situated at the system integration level: the standard Reinforce policy gradient is adapted to the multi-tenant scheduling domain through a domain-specific state representation that encodes tenant heterogeneity and load dynamics, a constrained multi-objective reward function that directly targets operational efficiency, and a Dirichlet-parameterized continuous action space that satisfies quota-sharing constraints without discretization. The ablation study confirms that the joint application of baseline correction, reward normalization, and dynamic learning rate decay is necessary to achieve the observed performance levels, and Welch's t-test results indicate that all performance advantages over the four baselines are statistically significant. The adaptive scheduling mechanism based on the Reinforce algorithm improves resource utilization, scheduling responsiveness, and inter-tenant allocation consistency in the simulated multi-tenant cloud platform, and the observed performance gains are statistically confirmed across 30 independent runs by Welch's t-test. The generalizability of the results is subject to the following boundary conditions.

Relative to recent advanced reinforcement learning scheduling methods, including DDPG, PPO, A2C, and SAC, the core novelty of this work lies in the co-design of a multi-scale tenant-aware state representation, a constrained generalized-mean reward function, and a Dirichlet-parameterized continuous action space within a unified online Reinforce with baseline framework that requires no replay buffer, no critic network, and no surrogate clipping. From a theoretical perspective, the proposed framework establishes that Reinforce with baseline converges to a locally optimal scheduling policy under three verifiable conditions: bounded rewards, sufficient policy network expressiveness, and a Robbins-Monro learning-rate schedule, demonstrating that a Monte Carlo policy-gradient method enhanced by domain-specific variance reduction achieves statistically reliable performance advantages over DDPG and three additional baselines without the stability assumptions required by replay-based or actor-critic methods. The validity of this conclusion is limited to the studied environment, which comprises 50 servers and 100 tenants, with workloads derived from Google Cluster Data 2019 and Alibaba Cluster Trace 2018. While the framework is expected to remain effective within similar scheduling settings, scaling to production environments with thousands of servers and tenants would require distributed policy execution and partitioned state aggregation, and its generalization to substantially different workload characteristics (e.g., GPU-intensive or memory-bandwidth-bound tasks) remains unverified. Furthermore, practical deployment would require integration with production orchestration systems and online safety mechanisms to prevent violations of service-level agreements during early-stage training.

For cloud operations managers and infrastructure planners, the results indicate that replacing static quota assignment or rule-based heuristics with an online policy-gradient scheduling mechanism yields the largest operational gains under heavy-load conditions, where the performance gap relative to static and heuristic baselines is widest. The ablation results indicate that baseline correction, reward normalization, and learning rate decay each contribute independently to scheduling performance and should be treated as required components in any implementation. Deployment decisions should be preceded by pilot evaluation in the target environment, given the simulation-based scope of the current validation and the unverified behavior under node failure, workload drift, and tenant churn.

Funding

This research was funded by the 2024 Jiangsu Province Industry-University-Research Cooperation Project, "A Cloud-Native Network Element Configuration Method and System Patent Transfer," project number BY20240469.

Institutional Review Board Statement

Not applicable.

Declaration of Artificial Intelligence (AI) Tools

The authors used DeepSeek solely for language editing and readability improvement. The authors reviewed and verified all content and take full responsibility for the accuracy and integrity of the manuscript.

References

Anand, J. and Karthikeyan, B. (2025). EADRL: Efficiency-aware adaptive deep reinforcement learning for dynamic task scheduling in edge-cloud environments. *Results in Engineering,* 27(1), 105890-105897.

Baburao, D., Pavankumar, T., and Prabhu, C. S. R. (2023). Load balancing in the fog nodes using particle swarm optimization-based enhanced dynamic resource allocation method. *Applied Nanoscience*, 13(2), 1045-1054.

Badri, S., Alghazzawi, D. M., Hasan, S. H., Alfayez, F., Hasan, S. H., and Rahman, M. (2023). An efficient and secure model using adaptive optimal deep learning for task scheduling in cloud computing. *Electronics,* 12(6), 1441.

Bao, H., Wang, Y., Huo, Y., Li, P., Nie, L., Wu, L., and Yu, G. (2025). Joint task offloading, resource allocation, and service caching in mobile edge computing via soft actor-critic learning. *IEEE Transactions on Vehicular Technology,* 75(3), 4854-4869.

Fu, X. L., Sun, Y., Wang, H. F., and Li, H. H. (2023). Task scheduling of cloud computing based on hybrid particle swarm algorithm and genetic algorithm. *Cluster Computing,* 26(5), 2479-2488.

Godhrawala, H. and Sridaran, R. (2023). A dynamic Stackelberg game based multi-objective approach for effective resource allocation in cloud computing. *International Journal of Information Technology,* 15(2), 803-818.

Gurusamy, S. and Selvaraj, R. (2024). Resource allocation with efficient task scheduling in cloud computing using hierarchical auto-associative polynomial convolutional neural network. *Expert Systems with Applications,* 249, 123554.

He, Y., Fang, J. C., Yu, F. R., and Leung, V. C. (2024). Large language models (LLMs) inference offloading and resource allocation in cloud-edge computing: An active inference approach. *IEEE Transactions on Mobile Computing,* 23(12), 11253-11264.

Hou, X. W., Wang, J. J., Jiang, C. X., Meng, Z. Z., Chen, J. R., and Ren, Y. (2023). Efficient federated learning for metaverse via dynamic user selection, gradient quantization and resource allocation. *IEEE Journal on Selected Areas in Communications,* 42(4), 850-866.

Hu, H., Wu, D. G., Zhou, F. H., Zhu, X. W., Hu, R. Q., and Zhu, H. B. (2023). Intelligent resource allocation for edge-cloud collaborative networks: A hybrid DDPG-D3QN approach. *IEEE Transactions on Vehicular Technology,* 72(8), 10696-10709.

Huang, C., Chen, G. J., Xiao, P., Xiao, Y., Han, Z., and Chambers, J. A. (2024). Joint offloading and resource allocation for hybrid cloud and edge computing in SAGINs: A decision assisted hybrid action space deep reinforcement learning approach. *IEEE Journal on Selected Areas in Communications,* 42(5), 1029-1043.

Huang, J. W., Wan, J. Y., Lv, B. F., Ye, Q., and Chen, Y. (2023). Joint computation offloading and resource allocation for edge-cloud collaboration in internet of vehicles via deep reinforcement learning. *IEEE Systems Journal,* 17(2), 2500-2511.

Khani, M., Sadr, M. M., and Jamali, S. (2024). Deep reinforcement learning-based resource allocation in multi-access edge computing. *Concurrency and Computation: Practice and Experience*, 36(15), e7995.

Lahza, H., BR, S., and Lahza, H. F. M. (2024). Adaptive multi-objective resource allocation for edge-cloud workflow optimization using deep reinforcement learning. *Modelling,* 5(3), 1298-1313.

Li, Y. H., Zhang, X. X., Zeng, T. Y., Duan, J. P., Wu, C., and Wu, D. (2023). Task placement and resource allocation for edge machine learning: A gnn-based multi-agent reinforcement learning paradigm. *IEEE Transactions on Parallel and Distributed Systems,* 34(12), 3073-3089.

Liu, Q., Luo, R., Liang, H. R., and Liu, Q. L. (2023). Energy-efficient joint computation offloading and resource allocation strategy for ISAC-aided 6G V2X networks. *IEEE Transactions on Green Communications and Networking,* 7(1), 413-423.

Lu, J. L., Yang, J., Li, S. B., Li, Y. J., Jiang, W., and Dai, J. T. (2024). A2C-DRL: Dynamic scheduling for stochastic edge–cloud environments using A2C and deep reinforcement learning. *IEEE Internet of Things Journal,* 11(9), 16915-16927.

Mangalampalli, S., Karri, G. R., Kumar, M., Khalaf, O. I., Romero, C. A. T., and Sahib, G. M. A. (2024). DRLBTSA: Deep reinforcement learning based task-scheduling algorithm in cloud computing. *Multimedia Tools and Applications,* 83(3), 8359-8387.

Mangalampalli, S., Karri, G. R., Ratnamani, M. V., Mohanty, S. N., Jabr, B. A., Ali, Y.A., Ali, S., and Abdullaeva, B. S. (2024). Efficient deep reinforcement learning based task scheduler in multi cloud environment. *Scientific Reports,* 14(1), 21850-21857.

Mikram, H., El Kafhali, S., and Saadi, Y. (2024). HEPGA: A new effective hybrid algorithm for scientific workflow scheduling in cloud computing environment. *Simulation Modelling Practice and Theory,* 130, 102864.

Mittal, P., Kumar, S., and Sharma, S. (2024). Revolutionizing Cloud-Based Task Scheduling: A Novel Hybrid Algorithm for Optimal Resource Allocation and Efficiency in Contemporary Networked Systems. *International Journal of Computing and Digital Systems, 15*(1), 1551-1563.

Moazeni, A., Khorsand, R., and Ramezanpour, M. (2023). Dynamic resource allocation using an adaptive multi-objective teaching-learning based optimization algorithm in cloud. *IEEE Access,* 11, 23407-23419.

Petrovska, I. and Kuchuk, H. (2023). Adaptive resource allocation method for data processing and security in cloud environment. *Advanced Information Systems,* 7(3), 67-73.

Pirozmand, P., Jalalinejad, H., Hosseinabadi, A. A. R., Mirkamali, S., and Li, Y. Q. (2023). An improved particle swarm optimization algorithm for task scheduling in cloud computing. *Journal of Ambient Intelligence and Humanized Computing,* 14(4), 4313-4327.

Raghavendar, K., Batra, I., and Malik, A. (2023). A robust resource allocation model for optimizing data skew and consumption rate in cloud-based IoT environments. *Decision Analytics Journal,* 7, 100200.

Saidi, K., and Bardou, D. (2023). Task scheduling and VM placement to resource allocation in Cloud computing: challenges and opportunities. *Cluster Computing,* 26(5), 3069-3087.

Shahidani, F. R., Ghasemi, A., Haghighat, A. T., and Keshavarzi, A. (2023). Task scheduling in edge-fog-cloud architecture: a multi-objective load balancing approach using reinforcement learning algorithm. *Computing,* 105(6), 1337-1359.

Simaiya, S., Lilhore, U. K., Sharma, Y. K., Rao, K. B. V. B., Rao, V. V. R. M., and Baliyan, A. (2024). A hybrid cloud load balancing and host utilization prediction method using deep learning and optimization techniques. *Scientific Reports,* 14(1), 1337.

Singh, N. (2025). Adaptive Learning Rate Strategies in Deep Reinforcement Learning Agents. *International Journal of Advanced Research in Computer Science and Engineering (IJARCSE),* 1(1), 9-15.

Sumiea, E. H., Abdulkadir, S. J., Alhussiana, H. S., Al-Selwia, S. M., Alqushaibia, A., and Ragaba, M. G. (2024). Deep deterministic policy gradient algorithm: A systematic review. *Heliyon,* 10(9), e30697.

Sundar, D. (2025). Reinforcement Learning Techniques for Autonomous Cloud Optimization and Adaptive Resource Management. *International Journal of Artificial Intelligence, Data Science, and Machine Learning,* 6(3), 134-145.

Wang, J. Y., Lu, T. Y., Li, L., and Huang, D. C. (2024). Enhancing personalized search with ai: a hybrid approach integrating deep learning and cloud computing. *Journal of Advanced Computing Systems,* 4(10), 1-13.

Wang, S. K., Zheng, H. T., Wen, X., and Shang, F. (2024). Distributed high-performance computing methods for accelerating deep learning training. *Journal of Knowledge Learning and Science Technology,* 3(3), 108-126.

Wang, Z., Chen, S., Bai, L., Gao, J., Tao, J., Bond, R. R., and Mulvenna, M. D. (2023). Reinforcement learning based task scheduling for environmentally sustainable federated cloud computing. *Journal of Cloud Computing,* 12(1), 174-181.

Wu, Y. X., Cai, C. J., Bi, X. M., Xia, J. J., Gao, C. Z., Tang, Y. J., and Lai, S. W. (2023). Intelligent resource allocation scheme for cloud-edge-end framework aided multi-source data stream. *EURASIP Journal on Advances in Signal Processing*, 2023(1), 56.

Yakubu, I. Z. and Murali, M. (2023). An efficient meta-heuristic resource allocation with load balancing in IoT-Fog-cloud computing environment. *Journal of Ambient Intelligence and Humanized Computing*, 14(3), 2981-2992.

Zhang, H. J., Wang, H. Y., Li, Y. B., Long, K. P., and Nallanathan, A. (2023). DRL-driven dynamic resource allocation for task-oriented semantic communication. *IEEE Transactions on Communications,* 71(7), 3992-4004.

Zhang, H., Wang, J., Zhang, H., and Bu, C. (2024). Security computing resource allocation based on deep reinforcement learning in serverless multi-cloud edge computing. *Future Generation Computer Systems,* 151(1), 152-161.

Zhou, G. Y., Tian, W. H., Buyya, R., Xue, R. N., and Song, L. (2024). Deep reinforcement learning-based methods for resource scheduling in cloud computing: A review and future directions. *Artificial Intelligence Review,* 57(5), 124.

Zhou, H., Wang, Z. N., Zheng, H. T., He, S. B., and Dong, M. X. (2023). Cost minimization-oriented computation offloading and service caching in mobile cloud-edge computing: An A3C-based approach. *IEEE Transactions on Network Science and Engineering,* 10(3), 1326-1338.

Yue Cao received the master's of science in software engineering from Sichuan University, Chengdu, China. He is currently a Senior Experimentalist with the School of Information Engineering, Nanjing Polytechnic Institute, Nanjing, China. His current research interests include cloud-native technologies, intelligent IoT systems, and deep learning applications.