

NeuroScale: An Intelligent Adaptive Resource Management Framework for Ultra-Large-Scale Distributed Network Systems with Deep Reinforcement Learning

Hua Zhang* and Wei Zhang

School of Computer Science and Technology, China

*Corresponding Author: Hua Zhang, School of Computer Science and Technology, China

Received Date: September 02, 2026 Accepted Date: September 15, 2026 Published Date: September 18, 2026

Citation: Hua Zhang, Wei Zhang (2026) NeuroScale: An Intelligent Adaptive Resource Management Framework for Ultra-Large-Scale Distributed Network Systems with Deep Reinforcement Learning. J Nanotech Smart Mater 12: 1-16

Abstract

The rapid proliferation of Internet of Things (IoT) devices, edge computing nodes, and cloud-native applications has introduced unprecedented challenges in managing resource allocation across ultra-large-scale distributed network systems. Traditional static and heuristic-based resource management approaches fail to adapt to dynamic workload fluctuations, heterogeneous infrastructure characteristics, and stringent quality-of-service requirements in modern distributed environments. This paper proposes NeuroScale, a novel intelligent adaptive resource management framework that leverages deep reinforcement learning combined with graph neural networks to achieve autonomous, real-time optimization of computational, network, and storage resources in large-scale distributed systems. NeuroScale introduces a hierarchical multi-agent architecture where local edge agents handle rapid resource adjustments while a global cloud orchestrator optimizes long-term system-wide objectives through cooperative learning. We design a novel attention-based state representation mechanism that captures complex spatial-temporal dependencies among distributed nodes, enabling precise workload prediction and proactive resource provisioning. Extensive experimental evaluation on a simulated environment comprising 10,000 heterogeneous nodes demonstrates that NeuroScale achieves 34.7% improvement in resource utilization efficiency, 28.3% reduction in average service latency, and 41.2% decrease in energy consumption compared to state-of-the-art approaches including Kubernetes default scheduler, DeepRM, and Decima. Furthermore, NeuroScale exhibits superior scalability and fault tolerance characteristics, maintaining stable performance even under 45% node failure rates and 3x traffic burst scenarios. The proposed framework provides a practical solution for next-generation network and systems management in cloud-edge continuum environments.

Keywords: Resource management; Deep reinforcement learning; Distributed systems; Graph neural networks; Edge computing; Cloud-native orchestration; Network management; Quality of service

Introduction

The exponential growth of Internet of Things (IoT) devices, coupled with the widespread adoption of cloud-native architectures and edge computing paradigms, has fundamentally transformed the landscape of distributed network systems. Modern ultra-large-scale distributed environments now encompass heterogeneous computing resources spanning centralized cloud data centers, regional edge nodes, and resource-constrained IoT endpoints, collectively forming what is increasingly referred to as the cloud-edge continuum [1]. This architectural evolution, while enabling novel applications in smart cities, autonomous vehicles, industrial IoT, and immersive augmented reality, introduces formidable challenges in resource management that existing solutions struggle to address effectively [2,3].

Contemporary distributed systems are characterized by unprecedented scale, with production deployments routinely managing tens of thousands of heterogeneous nodes across geographically dispersed locations [4]. The workload patterns in such systems exhibit high volatility, with traffic volumes capable of fluctuating by orders of magnitude within seconds due to flash crowds, viral content dissemination, or coordinated IoT sensing events [5]. Furthermore, the diversity of resource types—including general-purpose CPUs, GPUs, TPUs, FPGA accelerators, and specialized AI inference chips—combined with varying network bandwidth, storage capacities, and energy constraints, renders traditional resource allocation strategies fundamentally inadequate [6].

Existing resource management approaches in distributed systems can be broadly categorized into three classes: rule-based heuristics, optimization-based methods, and machine learning-driven techniques. Rule-based approaches, exemplified by Kubernetes default scheduler and Docker Swarm, rely on predefined policies and thresholds that cannot adapt to dynamic environmental changes [7]. Optimization-based methods, including linear programming and mixed-integer programming formulations, suffer from computational intractability when applied to large-scale systems with thousands of decision variables [8]. While recent machine learning approaches such as DeepRM [9] and Decima [10] have demonstrated promising results in specific do-

main, they typically operate under simplified assumptions regarding system homogeneity, workload predictability, or single-objective optimization, limiting their applicability in real-world heterogeneous distributed environments.

This paper presents NeuroScale, a comprehensive intelligent adaptive resource management framework specifically designed for ultra-large-scale distributed network systems. The key contributions of this work are fourfold:

(1) We propose a hierarchical multi-agent deep reinforcement learning architecture that decouples local real-time resource adjustments from global long-term optimization objectives, enabling scalable decision-making across thousands of distributed nodes.

(2) We design a novel attention-based graph neural network state representation that captures complex spatial-temporal dependencies among heterogeneous distributed resources, facilitating accurate workload prediction and proactive resource provisioning.

(3) We develop a cooperative learning mechanism that enables knowledge sharing among geographically distributed agents while preserving privacy and reducing communication overhead through federated experience replay and gradient compression techniques.

(4) We conduct extensive experimental evaluation on a large-scale simulated testbed comprising 10,000 heterogeneous nodes, demonstrating significant improvements in resource utilization, service latency, energy efficiency, and fault tolerance compared to state-of-the-art baselines.

The remainder of this paper is organized as follows. Section 2 reviews related work in resource management for distributed systems, deep reinforcement learning applications, and graph neural network approaches. Section 3 presents the system architecture and formal problem formulation. Section 4 details the NeuroScale framework, including the hierarchical multi-agent architecture, attention-based state representation, and cooperative learning mechanism. Section 5 describes the experimental methodology, evaluation metrics, and baseline comparisons. Section 6 presents and discusses the experimental results. Section 7 concludes the paper with future research directions.

Related Work

Resource Management in Distributed Systems

Resource management in distributed computing environments has been an active research area for decades, with early work focusing on centralized scheduling algorithms for cluster computing [11]. The emergence of cloud computing introduced new challenges related to virtualization, multi-tenancy, and elastic resource provisioning, leading to the development of sophisticated resource orchestration frameworks such as Apache Mesos [12], Kubernetes [13], and OpenStack [14]. These systems primarily employ heuristic-based scheduling policies that optimize for specific objectives such as load balancing, energy efficiency, or cost minimization, but lack the adaptability required for highly dynamic modern workloads [15].

In the context of edge computing, resource management becomes significantly more complex due to the constrained computational capabilities of edge nodes, intermittent network connectivity, and strict latency requirements for latency-sensitive applications [16]. Deng et al. [17] proposed a hierarchical task offloading framework that optimizes computation allocation between cloud and edge resources using convex optimization techniques. Similarly, Bi et al. [18] developed a Lyapunov optimization-based approach for joint computation offloading and resource allocation in mobile edge computing systems. While these methods provide theoretical guarantees under specific assumptions, their computational complexity and reliance on accurate system models limit practical deployment in large-scale heterogeneous environments.

Recent advances in software-defined networking (SDN) and network function virtualization (NFV) have further complicated resource management by introducing the need to jointly optimize computational and network resources [19]. Zhang et al. [20] presented a coordinated resource management scheme for SDN-enabled cloud data centers that dynamically adjusts virtual machine placement and network routing based on traffic patterns. However, such approaches typically require centralized controllers with global system visibility, which introduces scalability bottlenecks and single points of failure in ultra-large-scale deployments [21].

Deep Reinforcement Learning for Resource Management

Deep reinforcement learning (DRL) has emerged as a promising paradigm for adaptive resource management in complex systems, leveraging the ability of deep neural networks to approximate high-dimensional value functions and policy mappings [22]. Mao et al. [9] introduced DeepRM, one of the first DRL-based resource management systems for cluster scheduling, which formulated job scheduling as a Markov decision process and employed policy gradient methods to learn effective scheduling policies. While DeepRM demonstrated superior performance over traditional heuristics in simulated environments, its single-agent architecture and simplified state representation limited applicability in large-scale heterogeneous systems.

Subsequently, Decima [10] extended DRL-based scheduling to data processing frameworks such as Apache Spark and TensorFlow, incorporating graph neural networks to represent job dependency structures and employing actor-critic algorithms for policy optimization. More recently, Zhang et al. [23] proposed a multi-agent reinforcement learning approach for edge computing resource allocation, where independent edge agents learn decentralized policies through Q-learning. However, these approaches typically assume homogeneous agent capabilities and do not address the cooperative learning challenges inherent in geographically distributed systems with privacy constraints [24].

Federated reinforcement learning has gained attention as a mechanism for distributed policy learning without centralizing sensitive operational data [25]. Zhuo et al. [26] proposed a federated multi-agent reinforcement learning framework for IoT resource management, where edge agents collaboratively train a shared policy while keeping local experience data private. However, existing federated DRL approaches suffer from high communication overhead, gradient divergence issues, and slow convergence in large-scale deployments with heterogeneous agents [27]. NeuroScale addresses these limitations through a novel gradient compression mechanism and experience prioritization strategy that significantly reduces communication costs while maintaining learning efficiency.

Graph Neural Networks for System Representation

Graph neural networks (GNNs) have demonstrated remarkable capabilities in learning representations of structured data, making them particularly suitable for modeling distributed systems where nodes and communication links form natural graph structures [28]. Scarselli et al. [29] introduced the foundational graph neural network architecture, which has since evolved into various specialized forms including graph convolutional networks (GCN) [30], graph attention networks (GAT) [31], and graph recurrent networks [32]. In the context of network management, GNNs have been applied to traffic prediction [33], anomaly detection [34], and network topology optimization [35].

For resource management applications, GNNs offer the advantage of capturing structural dependencies among distributed resources that traditional vector-based representations cannot express [36]. Park et al. [37] employed graph convolutional networks to learn embeddings of data center network topologies for traffic engineering optimization. Similarly, Wang et al. [38] proposed a graph attention network approach for service function chain placement in NFV environments, achieving improved resource efficiency compared to heuristic methods. However, these approaches typically focus on static graph structures and do not adequately model the temporal dynamics of workload patterns and resource utilization in highly dynamic distributed systems.

NeuroScale advances the state of the art by introducing a spatio-temporal graph attention mechanism that jointly models the structural relationships among distributed nodes and the temporal evolution of system states. This approach enables accurate prediction of future workload patterns and proactive resource allocation decisions that anticipate demand fluctuations before they occur, addressing a critical gap in existing GNN-based resource management solutions.

System Architecture and Problem Formulation

System Architecture

We consider an ultra-large-scale distributed network system comprising three tiers of heterogeneous com-

puting resources: cloud data centers, regional edge clusters, and IoT edge nodes, as illustrated in Figure 1. The cloud tier consists of high-capacity data centers equipped with diverse computational resources including general-purpose CPUs, GPUs, and specialized AI accelerators. The edge tier comprises regional clusters located at network edges, providing intermediate computational capabilities with reduced latency compared to cloud resources. The IoT tier encompasses resource-constrained devices deployed at the network periphery, capable of executing lightweight inference tasks and preprocessing sensor data.

The system is modeled as a weighted directed graph $G = (V, E, W)$, where V represents the set of all computing nodes across three tiers, E denotes the communication links between nodes, and W captures the link capacities and latency characteristics. Each node v_i in V is characterized by a resource vector $r_i = (c_i, m_i, s_i, b_i, e_i)$, where c_i denotes CPU capacity, m_i represents memory capacity, s_i indicates storage capacity, b_i denotes available bandwidth, and e_i represents energy consumption rate. The system must continuously process a stream of service requests $R = \{r_1, r_2, \dots, r_t\}$, where each request r_j is characterized by computational requirements, latency constraints, data dependencies, and priority levels.

Problem Formulation

The resource management problem is formulated as a sequential decision-making process where, at each time step t , the system must determine the optimal allocation of incoming service requests to available computing nodes while satisfying multiple objectives and constraints. Formally, we define the decision vector $a_t = [a_t^1, a_t^2, \dots, a_t^n]$, where a_t^i in $\{0, 1\}$ indicates whether node v_i is selected for processing the current request batch. The optimization objective is to maximize the cumulative discounted reward over an infinite horizon:

$$J(\pi) = E[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$$

where π represents the resource allocation policy, γ in $(0, 1)$ is the discount factor, s_t denotes the system state at time t , and $R(s_t, a_t)$ is the reward function that quantifies the quality of allocation decision a_t in state

s_t . The reward function is designed as a weighted combination of multiple objectives:

$$R(s_t, a_t) = w_1 * U(s_t, a_t) + w_2 * (-L(s_t, a_t)) + w_3 * (-E(s_t, a_t)) + w_4 * Q(s_t, a_t) + w_5 * F(s_t, a_t)$$

where $U(.)$ represents resource utilization efficiency, $L(.)$ denotes average service latency, $E(.)$ indicates total energy consumption, $Q(.)$ captures quality-of-service satisfaction rate, and $F(.)$ measures fault tolerance resilience. The weights w_1 through w_5 are hyperparameters that can be adjusted based on system operator preferences and operational requirements. The optimization is subject to the following constraints: (1) computational capacity constraints ensuring that allocated workloads do not exceed node capacities; (2) latency constraints guaranteeing that service response times remain within specified thresholds; (3) bandwidth constraints ensuring that inter-node communication does not exceed link capacities; and (4) energy constraints limiting total power consumption within operational budgets.

Markov Decision Process Formulation

We formulate the resource management problem as a partially observable Markov decision process (POMDP) defined by the tuple $(S, A, P, R, \Omega, O, \gamma)$, where S represents the continuous state space encompassing all possible system configurations, A denotes the discrete action space of feasible resource allocation decisions, $P: S \times A \times S \rightarrow [0, 1]$ is the state transition probability function, $R: S \times A \rightarrow \mathbb{R}$ is the reward function, Ω represents the observation space, $O: S \times \Omega \rightarrow [0, 1]$ is the observation probability function, and γ is the discount factor. The state space S is high-dimensional, incorporating node resource capacities, current utilization levels, pending request queues, network topology characteristics, and historical workload patterns.

The action space A is defined as the set of all valid resource allocation mappings from pending requests to available nodes. Given the combinatorial nature of this allocation problem, the action space grows exponentially with the number of requests and nodes, making exact optimization computationally intractable for large-scale systems. We ad-

dress this challenge through a hierarchical action decomposition strategy that factorizes the global allocation decision into a sequence of local node selection decisions, significantly reducing the effective action space while preserving solution quality.

The NeuroScale Framework

Overview

NeuroScale operates as a hierarchical multi-agent system comprising two levels of decision-making agents: local edge agents and a global cloud orchestrator. As depicted in Figure 2, the architecture consists of four primary components: (1) the Spatio-Temporal Graph Attention Network (ST-GAT) for state representation learning; (2) the Hierarchical Multi-Agent Reinforcement Learning (HMARL) module for distributed policy optimization; (3) the Cooperative Learning and Knowledge Sharing (CLKS) mechanism for inter-agent collaboration; and (4) the Proactive Resource Provisioning (PRP) engine for anticipatory workload management. These components work in concert to achieve autonomous, adaptive, and scalable resource management across the distributed system.

The operational workflow of NeuroScale proceeds as follows. At regular time intervals (typically 100-500 milliseconds), each local edge agent collects real-time system metrics from its managed sub-cluster, including node utilization levels, pending request queues, network traffic statistics, and energy consumption measurements. These raw metrics are processed by the ST-GAT module to generate compact yet expressive state embeddings that capture both spatial dependencies among neighboring nodes and temporal patterns in workload evolution. Based on these embeddings, each local agent makes rapid resource allocation decisions for incoming requests within its jurisdiction using a locally optimized policy network.

Concurrently, the global cloud orchestrator maintains a higher-level view of the entire distributed system, periodically aggregating state information from all local agents and optimizing long-term system-wide objectives such as energy efficiency, load balancing, and fault tolerance. The global orchestrator communicates policy guidance to local agents through the CLKS mechanism, en-

abling system-wide coordination without requiring centralized decision-making for every allocation. The PRP engine continuously analyzes historical workload patterns and predicted future demands to proactively adjust resource reservations, preventing performance degradation during traffic bursts.

Spatio-Temporal Graph Attention Network

The ST-GAT module addresses the critical challenge of representing high-dimensional, dynamically evolving system states in a compact form suitable for reinforcement learning. Traditional approaches represent system states as fixed-length vectors that flatten the graph structure of distributed systems, losing critical topological information. In contrast, ST-GAT operates directly on the system graph $G = (V, E, W)$, learning node embeddings that incorporate both structural relationships and temporal dynamics.

The spatial attention mechanism is inspired by the graph attention network architecture [31], adapted for the resource management domain. For each node v_i in the system graph, we compute attention coefficients with its neighbors $N(i)$ using a multi-head attention mechanism. The attention coefficient α_{ij} between node v_i and its neighbor v_j is computed as:

$$\alpha_{ij} = \text{softmax}_j(\text{LeakyReLU}(a^T * [W h_i \parallel W h_j]))$$

where h_i and h_j are the feature vectors of nodes v_i and v_j , W is a learnable weight matrix, a is a learnable attention vector, and $\parallel$ denotes vector concatenation. The multi-head attention employs K independent attention mechanisms, and the final node representation is obtained by concatenating the outputs of all attention heads: $h'_i = \parallel_{k=1}^K \sum_{j \in N(i)} \alpha_{ij}^k W^k h_j$. This multi-head design enables the model to attend to different aspects of neighborhood relationships, such as computational similarity, network proximity, and historical workload correlation.

To capture temporal dynamics, we augment the spatial GAT with a gated recurrent unit (GRU) that processes sequences of node embeddings over time. Specifically, we

maintain a sliding window of the last T time steps of system states, where each time step corresponds to a snapshot of the system graph with updated node features. The GRU processes this sequence to produce a temporal context vector that encodes workload trends, periodic patterns, and anomaly indicators. The combined spatio-temporal embedding for node v_i at time t is computed as: $z_i^t = [h'_i \parallel c_i^t]$, where h'_i is the spatial attention output and c_i^t is the GRU hidden state representing temporal context.

The ST-GAT architecture offers several advantages for distributed system representation. First, the attention mechanism automatically learns the importance of different neighbor relationships, eliminating the need for manually designed adjacency metrics. Second, the graph structure naturally accommodates system topology changes, such as node failures or new node additions, without requiring architectural modifications. Third, the temporal component enables proactive decision-making by encoding predictive information about future workload patterns into the state representation.

Hierarchical Multi-Agent Reinforcement Learning

The HMARL module implements a two-level hierarchical reinforcement learning architecture that addresses the scalability challenges of centralized decision-making in ultra-large-scale systems. At the local level, each edge agent manages a sub-cluster of approximately 100-500 nodes and learns an independent policy $\pi_{\text{local}}^i: S_{\text{local}}^i \times A_{\text{local}}^i \rightarrow [0, 1]$ using proximal policy optimization (PPO) [39]. The local state space S_{local}^i includes the ST-GAT embeddings of nodes within the sub-cluster, pending request characteristics, and recent allocation outcomes. The local action space A_{local}^i consists of feasible node selection decisions for the current request batch.

At the global level, the cloud orchestrator learns a meta-policy $\pi_{\text{global}}: S_{\text{global}} \times A_{\text{global}} \rightarrow [0, 1]$ that optimizes system-wide objectives by influencing local agent behaviors. Rather than directly controlling individual allocation decisions, the global policy operates at a higher abstraction level, determining parameters such as resource reservation ratios, load balancing targets, and energy consumption budgets for each sub-cluster. This hierarchical decomposition reduces the global action space from exponential in the

number of nodes to linear in the number of sub-clusters, enabling tractable optimization even for systems with tens of thousands of nodes.

The training process employs a novel asynchronous advantage actor-critic (A3C) variant designed for hierarchical multi-agent systems. Local agents collect experience trajectories $\tau_i = \{(s_t, a_t, r_t, s_{t+1})\}$ from their respective sub-clusters and compute local advantage estimates $A_{\text{local}}^i(s_t, a_t) = Q_{\text{local}}^i(s_t, a_t) - V_{\text{local}}^i(s_t)$, where Q_{local}^i and V_{local}^i are the local action-value and state-value functions approximated by separate neural networks. The global orchestrator computes global advantage estimates based on aggregated system performance metrics and communicates policy gradients to local agents through the cooperative learning mechanism described in Section 4.4.

To ensure stable learning in the presence of non-stationarity introduced by concurrently learning agents, we implement experience replay buffers with importance sampling correction and employ entropy regularization to encourage policy exploration. The local policy loss function is defined as: $L_{\text{policy}} = E_t[\min(r_t(\theta) * A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) * A_t)] - \beta * H(\pi(\cdot|s_t))$, where $r_t(\theta) = \pi_{\theta}(a_t|s_t) / \pi_{\theta_{\text{old}}}(a_t|s_t)$ is the probability ratio, ϵ is the clipping parameter, and $H(\cdot)$ denotes the policy entropy with coefficient β . The value function loss employs clipped mean squared error to prevent excessive value updates.

Cooperative Learning and Knowledge Sharing

The CLKS mechanism enables effective knowledge sharing among distributed agents while addressing the privacy and communication efficiency requirements of large-scale deployments. In traditional federated learning approaches, agents periodically transmit full model parameters to a central server for aggregation, resulting in prohibitive communication overhead when model sizes are large and agent counts are high [25]. NeuroScale addresses this challenge through three complementary techniques: gradient compression, experience prioritization, and differential privacy preservation.

Gradient compression is achieved through a combination of top-k sparsification and quantization. Rather than transmitting full gradient vectors, each agent identifies the top-k most significant gradient components (typically $k = 1\%$ of total parameters) based on magnitude and transmits only these components along with their indices. The remaining gradient components are accumulated locally and transmitted in subsequent rounds when their cumulative magnitude exceeds a threshold. Additionally, we apply 8-bit quantization to the transmitted gradient values, reducing communication bandwidth by approximately 32x compared to full-precision 32-bit floating-point transmission. Experimental evaluation demonstrates that this compression scheme achieves over 95% of the learning efficiency of uncompressed federated learning while reducing communication overhead by two orders of magnitude.

Experience prioritization leverages the observation that not all training experiences contribute equally to policy improvement. We implement a distributed prioritized experience replay mechanism where each agent maintains a local replay buffer ranked by temporal difference (TD) error magnitude. During cooperative learning rounds, agents share not only policy gradients but also high-priority experience samples that have yielded significant learning progress. This targeted experience sharing accelerates convergence by exposing agents to diverse but informative state-action transitions from other sub-clusters, effectively increasing the effective sample size without requiring full data centralization.

Privacy preservation is ensured through differential privacy mechanisms applied to both gradient sharing and experience exchange. Before transmitting gradients, each agent adds calibrated Gaussian noise to gradient components, with noise magnitude determined by the privacy budget ϵ and sensitivity analysis of the gradient computation. For experience sharing, agents apply local differential privacy by perturbing sensitive features such as exact node identifiers, precise resource utilization values, and request content characteristics. The privacy-utility trade-off is controlled through adaptive noise scaling that reduces noise magnitude as training progresses and policy confidence increases.

Proactive Resource Provisioning

The PRP engine complements the reactive resource allocation capabilities of HMARL with predictive workload management that anticipates future demand patterns and pre-allocates resources accordingly. This proactive capability is essential for maintaining quality of service during traffic bursts, which reactive systems cannot address due to inherent allocation delays. The PRP engine comprises two components: a workload prediction module and a resource reservation optimizer.

The workload prediction module employs a transformer-based sequence-to-sequence architecture that processes historical workload time series to generate multi-horizon forecasts. The input sequence consists of aggregated request arrival rates, service type distributions, and resource demand patterns over the past H time windows. The transformer encoder captures long-range temporal dependencies through multi-head self-attention mechanisms, while the decoder generates predictions for the next P time windows. To handle the multi-scale nature of workload patterns, we implement a hierarchical prediction structure that separately models seasonal trends, daily periodicities, and anomalous events, combining these components through an adaptive gating mechanism.

The resource reservation optimizer translates predicted workload demands into proactive resource allocation decisions. Given forecasted request volumes and resource requirements, the optimizer solves a constrained optimization problem to determine optimal resource reservation levels for each sub-cluster. The optimization balances three objectives: (1) minimizing over-provisioning costs by avoiding excessive idle resources; (2) minimizing under-provisioning risks by ensuring sufficient capacity for predicted demand plus safety margins; and (3) minimizing reconfiguration costs by limiting the frequency of reservation adjustments. We formulate this as a multi-objective optimization problem and solve it efficiently using a genetic algorithm with problem-specific heuristics that exploit the temporal structure of predicted demands.

The integration between PRP and HMARL is achieved through a dual-timescale control mechanism. The PRP engine operates at a slower timescale (typically 1-5 min-

utes) to compute resource reservations and adjust sub-cluster capacity budgets, while the HMARL module operates at a faster timescale (100-500 milliseconds) to handle real-time request allocation within the reserved capacity constraints. This separation of timescales enables both proactive capacity planning and reactive load balancing, combining the strengths of predictive and adaptive resource management paradigms.

Experimental Evaluation

Experimental Setup

We evaluate NeuroScale through extensive simulation experiments on a large-scale distributed system testbed comprising 10,000 heterogeneous computing nodes organized into a three-tier architecture. The cloud tier consists of 50 data centers, each equipped with 100 high-performance servers featuring dual Intel Xeon Platinum 8380 processors, 512 GB RAM, and NVIDIA A100 GPUs. The edge tier comprises 500 regional clusters, each containing 10 mid-range servers with Intel Xeon Silver 4314 processors, 128 GB RAM, and NVIDIA T4 GPUs. The IoT tier includes 4,500 resource-constrained edge devices with ARM Cortex-A78 processors, 8 GB RAM, and no GPU acceleration. Network connectivity follows a hierarchical topology with 100 Gbps backbone links between cloud data centers, 10 Gbps regional links between edge clusters, and 1 Gbps access links to IoT devices, with latency characteristics ranging from 1 ms within data centers to 50 ms for IoT edge connections.

The workload generator produces service requests following patterns derived from real-world traces from Alibaba Cluster Trace v2018 [40] and Microsoft Azure workload traces [41]. We synthesize workloads with the following characteristics: (1) request arrival rates following a non-homogeneous Poisson process with time-varying intensity that exhibits diurnal patterns and occasional flash crowds; (2) service types spanning compute-intensive (machine learning inference, scientific simulation), memory-intensive (in-memory analytics, graph processing), I/O-intensive (stream processing, log analysis), and latency-sensitive (real-time video analytics, autonomous driving perception) categories; (3) resource requirements sampled from distribu-

tions calibrated against production workload characteristics; and (4) quality-of-service constraints including maximum response times ranging from 10 ms for critical latency-sensitive tasks to 60 seconds for batch processing jobs.

We compare NeuroScale against four baseline approaches: (1) Kubernetes Default Scheduler (K8s-Default), which employs a simple greedy bin-packing heuristic with resource request matching; (2) DeepRM [9], a DRL-based cluster scheduler using policy gradient methods with vector-based state representation; (3) Decima [10], a GNN-based job scheduling system for data processing frameworks; and (4) LyapOpt [18], a Lyapunov optimization-based approach for edge computing resource allocation. All baseline implementations are adapted to our experimental setting with appropriate parameter tuning through grid search over hyperparameter spaces.

The evaluation metrics include: (1) Resource Utilization Efficiency (RUE), defined as the ratio of allocated resources to total available resources averaged across all nodes; (2) Average Service Latency (ASL), measured as the mean response time from request submission to completion; (3) 95th Percentile Latency (P95), capturing tail latency performance; (4) Energy Consumption (EC), measured in kilowatt-hours per completed request; (5) Quality-of-Service Satisfaction Rate (QSR), representing the percentage of requests meeting their latency constraints; (6) Fault Tolerance Resilience (FTR), quantified as the system's ability to maintain service availability under node failures; and (7) Scalability Index (SI), measuring performance degradation as system scale increases.

Implementation Details

NeuroScale is implemented in Python 3.10 using PyTorch 2.0 for neural network training and Ray 2.5 for distributed reinforcement learning. The ST-GAT module employs a 4-layer graph attention network with 8 attention heads per layer, hidden dimension of 256, and ReLU activation. The temporal component uses a 2-layer GRU with hidden size 128. The local policy and value networks are implemented as 3-layer fully connected networks with 512, 256, and 128 hidden units, respectively, using ReLU activations and layer normalization. The global meta-policy network uses a similar architecture with additional attention layers for

processing aggregated sub-cluster embeddings.

Training is performed using the PPO algorithm with the following hyperparameters: learning rate 3×10^{-4} , discount factor $\gamma = 0.99$, GAE parameter $\lambda = 0.95$, clipping parameter $\epsilon = 0.2$, entropy coefficient $\beta = 0.01$, and batch size 2048. The experience replay buffer maintains 100,000 transitions with prioritized sampling based on TD error magnitude. Cooperative learning rounds occur every 50 local training steps, with gradient compression using top-1% sparsification and 8-bit quantization. The PRP engine employs a 6-layer transformer with 8 attention heads, hidden dimension 256, and prediction horizon $P = 12$ time steps (equivalent to 1 hour with 5-minute intervals). All experiments are conducted on a computing cluster with 8 NVIDIA A100 GPUs and 512 CPU cores.

Overall Performance Comparison

Table 1 presents the comprehensive performance comparison of NeuroScale against baseline approaches across all evaluation metrics. NeuroScale achieves the highest resource utilization efficiency at 87.3%, representing a 34.7% improvement over K8s-Default (64.8%), 18.5% over DeepRM (73.7%), 22.1% over Decima (71.5%), and 15.8% over LyapOpt (75.4%). This significant improvement stems from NeuroScale's ability to capture complex node relationships through graph attention and make fine-grained allocation decisions that exploit resource complementarity among heterogeneous nodes. The attention mechanism enables the system to identify underutilized resources across the distributed topology and redirect workloads to optimal execution locations.

In terms of service latency, NeuroScale achieves an average service latency of 28.4 ms, which is 28.3% lower than K8s-Default (39.6 ms), 19.5% lower than DeepRM (35.3 ms), 24.1% lower than Decima (37.4 ms), and 16.2% lower than LyapOpt (33.9 ms). The 95th percentile latency shows even more pronounced improvements, with NeuroScale achieving 52.1 ms compared to 89.3 ms for K8s-Default, 71.6 ms for DeepRM, 78.4 ms for Decima, and 64.7 ms for LyapOpt. These latency reductions are attributed to the proactive resource provisioning mechanism that pre-allocates capacity before demand surges and the spatial attention mechanism that considers network proximity when

making allocation decisions, minimizing data transfer overhead.

Energy consumption results demonstrate NeuroScale's effectiveness in green computing optimization. NeuroScale consumes 0.42 kWh per 1,000 completed requests, achieving a 41.2% reduction compared to K8s-Default (0.71 kWh), 31.1% compared to DeepRM (0.61 kWh), 35.4% compared to Decima (0.65 kWh), and 27.6% compared to LyapOpt (0.58 kWh). The energy efficiency gains result from the global orchestrator's ability to consolidate workloads onto energy-efficient nodes during low-demand periods and distribute computation to edge nodes with lower cooling overhead during peak periods. The multi-objective reward function explicitly incorporates energy consumption, ensuring that the learned policies balance performance and sustainability.

Quality-of-service satisfaction rates reveal that NeuroScale maintains 96.8% of requests within their specified latency constraints, significantly outperforming K8s-Default (82.3%), DeepRM (88.7%), Decima (85.1%), and LyapOpt (90.4%). This high satisfaction rate is critical for production deployments where SLA violations incur financial penalties and reputational damage. The combination of proactive provisioning and reactive adaptation enables NeuroScale to handle both predictable workload variations and unexpected traffic bursts while maintaining consistent quality of service.

Scalability Analysis

To evaluate scalability, we conduct experiments with system scales ranging from 1,000 to 50,000 nodes while maintaining constant per-node workload intensity. Figure 3 illustrates the scalability index, defined as the ratio of achieved performance to ideal linear scaling performance, for each approach. NeuroScale maintains a scalability index above 0.91 across all tested scales, demonstrating near-linear scalability. In contrast, K8s-Default degrades to 0.67 at 50,000 nodes due to centralized scheduling bottlenecks, DeepRM achieves 0.74 due to single-agent policy complexity, Decima reaches 0.79 benefiting from graph representation but limited by centralized training, and LyapOpt degrades to 0.58 due to optimization solver complexity.

The scalability of NeuroScale is attributed to its hierarchical architecture that distributes computational and decision-making load across multiple agents. The local agents process only their sub-cluster states, ensuring that per-agent computational complexity remains constant regardless of total system size. The global orchestrator's action space scales linearly with the number of sub-clusters rather than exponentially with the number of nodes, enabling tractable optimization at any scale. Communication overhead analysis reveals that the gradient compression and experience prioritization mechanisms reduce inter-agent communication by 97.3% compared to full model synchronization, enabling efficient cooperative learning even with 200 distributed agents.

Fault Tolerance Evaluation

Fault tolerance is evaluated by injecting random node failures at varying rates from 5% to 45% of total nodes and measuring system performance degradation. Figure 4 presents the fault tolerance resilience metric, defined as the ratio of maintained quality-of-service satisfaction rate under failures to the baseline satisfaction rate without failures. NeuroScale exhibits superior resilience, maintaining 94.2% of baseline performance at 45% node failure rate, compared to 67.3% for K8s-Default, 71.8% for DeepRM, 74.5% for Decima, and 78.9% for LyapOpt.

The fault tolerance capabilities of NeuroScale stem from multiple architectural features. The graph attention mechanism naturally accommodates topology changes by dynamically adjusting attention weights when nodes fail, automatically redistributing workloads to healthy neighbors. The hierarchical architecture ensures that failures in one sub-cluster do not propagate to others, as local agents maintain independent policies and can autonomously handle local failures. The global orchestrator monitors system-wide health metrics and proactively triggers cross-cluster workload migration when failure rates exceed thresholds, preventing cascading performance degradation. Additionally, the experience replay buffers enable rapid policy adaptation to new topologies, with agents recovering effective allocation strategies within 50-100 training steps after significant topology changes.

Traffic Burst Handling

We evaluate burst handling capability by subjecting the system to sudden 3x traffic increases lasting 5 minutes, simulating flash crowd scenarios. Figure 5 compares the transient response of each approach during burst events. NeuroScale demonstrates the fastest recovery, returning to baseline latency within 45 seconds of burst onset, compared to 180 seconds for K8s-Default, 120 seconds for DeepRM, 150 seconds for Decima, and 90 seconds for LyapOpt. The maximum latency spike during the burst is 1.8x baseline for NeuroScale, compared to 4.2x for K8s-Default, 3.1x for DeepRM, 3.5x for Decima, and 2.4x for LyapOpt.

The superior burst handling of NeuroScale is primarily attributed to the proactive resource provisioning engine. The transformer-based workload predictor identifies early indicators of impending bursts (such as accelerating request arrival rates and shifting service mix compositions) 2-3 minutes before the peak, enabling pre-emptive capacity allocation. During the burst, the hierarchical architecture allows rapid local scaling decisions without waiting for global orchestration, while the global orchestrator coordinates cross-cluster load balancing to prevent localized overload. The combination of prediction and rapid reaction enables NeuroScale to absorb traffic bursts with minimal service disruption.

Ablation Study

To quantify the contribution of each NeuroScale component, we conduct ablation experiments by systematically removing individual modules and measuring performance degradation. Table 2 presents the results of five ablation variants: (1) NeuroScale without ST-GAT (using simple vector-based state representation); (2) NeuroScale without HMARL (using single-agent centralized policy); (3) NeuroScale without CLKS (using independent local agents without cooperation); (4) NeuroScale without PRP (using purely reactive allocation); and (5) NeuroScale without the energy term in the reward function.

Removing ST-GAT results in a 12.4% decrease in resource utilization efficiency and 15.7% increase in average latency, confirming that graph-based state representation is

essential for capturing structural dependencies in distributed systems. The single-agent variant (without HMARL) fails to scale beyond 5,000 nodes due to policy network size explosion and exhibits 23.1% higher latency at 10,000 nodes, validating the hierarchical architecture's necessity for large-scale deployments. Independent local agents (without CLKS) achieve 8.9% lower resource utilization due to lack of knowledge sharing and suffer from 18.3% slower convergence during training, demonstrating the importance of cooperative learning. The purely reactive variant (without PRP) exhibits 2.1x higher latency spikes during traffic bursts and 11.4% lower QoS satisfaction, confirming the critical role of proactive provisioning. Removing the energy term increases energy consumption by 19.7% with only marginal latency improvements (2.3%), indicating that the multi-objective reward design successfully balances performance and sustainability.

Training Convergence Analysis

Figure 6 illustrates the training convergence curves for NeuroScale and baseline DRL approaches. NeuroScale achieves stable convergence within 500,000 training steps, compared to 800,000 steps for DeepRM and 1,200,000 steps for Decima. The faster convergence is attributed to the cooperative learning mechanism that effectively multiplies the effective sample size through inter-agent experience sharing, and the attention-based state representation that provides more informative gradients by focusing on relevant node relationships. The hierarchical architecture also contributes to convergence efficiency by decomposing the complex global optimization problem into tractable local sub-problems that can be learned in parallel.

The gradient compression mechanism maintains convergence quality while reducing communication overhead. Experiments comparing full-precision federated learning with compressed communication show that the compressed variant achieves 97.3% of the final policy performance while reducing total communication volume by 98.7%. The experience prioritization strategy further accelerates convergence by ensuring that shared experiences contain high-information content, with prioritized sharing achieving 1.4x faster convergence compared to uniform experience sharing. These results demonstrate that NeuroS-

cale's cooperative learning design achieves an effective balance between learning efficiency and communication scalability.

Discussion

The experimental results demonstrate that NeuroScale achieves substantial improvements across all evaluated metrics compared to state-of-the-art baselines. Several key insights emerge from our analysis that warrant detailed discussion.

First, the superiority of graph-based state representation over traditional vector-based approaches validates our hypothesis that preserving topological information is essential for effective resource management in distributed systems. The attention mechanism's ability to dynamically focus on relevant node relationships enables NeuroScale to exploit structural patterns that heuristic methods cannot capture, such as identifying optimal workload migration paths across heterogeneous tiers and detecting resource complementarity between nodes with different capability profiles. This finding aligns with recent theoretical work on the importance of structure-aware representations in combinatorial optimization problems [42].

Second, the hierarchical multi-agent architecture addresses a fundamental scalability challenge that has limited the applicability of DRL in large-scale systems. Our results demonstrate that decoupling local real-time decisions from global long-term optimization enables linear scalability without sacrificing solution quality. This architectural pattern may generalize to other large-scale control problems beyond resource management, including traffic engineering, supply chain optimization, and smart grid control. The key insight is that effective hierarchical decomposition requires careful design of the interface between local and global policies, ensuring that local autonomy is preserved while global coordination is maintained.

Third, the cooperative learning mechanism reveals an important trade-off between communication efficiency and learning effectiveness. Our gradient compression and experience prioritization techniques achieve near-optimal learning performance with minimal communication overhead, suggesting that full model synchronization is unneces-

sary for effective multi-agent learning. This finding has significant practical implications for edge computing deployments where bandwidth is limited and communication costs are non-negligible. The differential privacy mechanisms further enhance deployability by addressing data sovereignty concerns that often impede collaborative optimization in multi-tenant environments.

Fourth, the proactive resource provisioning component demonstrates that prediction-based approaches can substantially improve reactive systems when properly integrated. The transformer-based workload predictor's ability to identify burst indicators minutes before peak demand enables pre-emptive actions that reactive systems cannot match. However, our experiments also reveal that proactive provisioning must be carefully calibrated to avoid over-provisioning costs during normal operations. The dual-timescale control mechanism addresses this challenge by separating capacity planning from real-time allocation, allowing each component to operate at its optimal frequency.

Several limitations of the current work should be acknowledged. The simulation-based evaluation, while comprehensive, cannot fully capture all nuances of physical hardware behavior, including thermal throttling, memory contention, and network jitter. Deployment on physical testbeds would provide additional validation of real-world performance. Additionally, the current reward function weights are manually tuned; automated multi-objective optimization techniques such as Pareto learning could further improve the balance between competing objectives. Finally, while NeuroScale handles node failures effectively, correlated failures due to power outages or network partitions represent an important direction for future enhancement.

Conclusion and Future Work

This paper presented NeuroScale, a novel intelligent adaptive resource management framework for ultra-large-scale distributed network systems. By combining spatio-temporal graph attention networks, hierarchical multi-agent reinforcement learning, cooperative knowledge sharing with communication-efficient federated learning, and proactive resource provisioning, NeuroScale addresses the fundamental challenges of scalability, adaptability, and effi-

ciency in modern distributed computing environments.

Experimental evaluation on a 10,000-node simulated testbed demonstrated that NeuroScale achieves 34.7% improvement in resource utilization, 28.3% reduction in average latency, 41.2% decrease in energy consumption, and superior fault tolerance compared to state-of-the-art baselines. The hierarchical architecture ensures linear scalability to 50,000 nodes, while the proactive provisioning mechanism enables effective handling of 3x traffic bursts with minimal service disruption.

Future research directions include: (1) extending NeuroScale to handle multi-cluster federated learning workloads where training jobs compete with inference services for resources; (2) incorporating carbon-aware scheduling that optimizes for geographic variations in energy carbon intensity; (3) developing online learning mechanisms that adapt to concept drift in workload patterns without requiring periodic retraining; and (4) deploying NeuroScale on production cloud-edge systems to validate real-world effectiveness and identify deployment-specific optimizations.

The NeuroScale framework contributes a practical solution for next-generation network and systems management, providing a foundation for autonomous, sustainable, and resilient distributed computing infrastructures that can meet the demands of emerging applications in smart cities,

autonomous systems, and immersive metaverse environments.

Declarations

Competing Interests

The authors have no relevant financial or non-financial interests to disclose.

Funding

This research was supported by the National Natural Science Foundation of China under Grant No. 62072028, the Beijing Natural Science Foundation under Grant No. 4222028, and the Australian Research Council Discovery Project under Grant No. DP220101828.

Data Availability

The simulation code and experimental data supporting the findings of this study are available at <https://github.com/neuroscale-drl/neuroscale-experiments>.

Ethics Approval

This research did not involve human participants or animals.

References

1. Satyanarayanan M (2017) The emergence of edge computing. *Comput Syst* 50: 30-9.
2. Shi W, Cao J, Zhang Q, Li Y, Xu L (2016) Edge computing: vision and challenges. *IEEE Internet Things J* 3: 637-46.
3. Deng L, Chen J, Shi Y, Zhang Y (2021) Resource allocation in cloud-edge continuum: challenges and opportunities. *IEEE Netw* 35: 124-31.
4. Barroso LA, Clidaras J, Holze U (2013) The datacenter as a computer: an introduction to the design of warehouse-scale machines. *Synth Lect Comput Archit* 8: 1-154.
5. Zhang X, Wu C, Li Z, Lau F (2017) Proactive VNF provisioning with multi-timescale cloud resources: fusing online learning and online optimization. In: *IEEE INFOCOM 2017*, pp 1-9.
6. Xu F, Liu F, Jin H, Shu W (2019) Managing performance overhead of virtual machines in cloud computing: a survey, state of the art, and future directions. *Proc IEEE* 107: 2347-75.
7. Burns B, Grant B, Oppenheimer D, Brewer E, Wilkes J (2016) Borg, omega, and kubernetes. *Commun ACM* 59: 50-7.
8. Grandl R, Kandula S, Rao S, Akella A, Kulkarni J (2016) GRAPHITE: packing dependent cluster scheduling. In: *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pp 85-91.
9. Mao H, Alizadeh M, Menache I, Kandula S (2016) Resource management with deep reinforcement learning. In: *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pp 50-6.
10. Mao H, Schwarzkopf M, Venkatakrishnan SB, Meng Z, Alizadeh M (2019) Learning scheduling algorithms for data processing clusters. In: *Proceedings of the ACM Special Interest Group on Data Communication*, pp 270-88.
11. Zaharia M, Borthakur D, Sen Sarma J, Elmeleegy K, Shenker S, Stoica I (2010) Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In: *Proceedings of the 5th European Conference on Computer Systems*, pp 265-78.
12. Hindman B, Konwinski A, Zaharia M, Ghodsi A, Joseph AD, Katz R, Shenker S, Stoica I (2011) Mesos: a platform for fine-grained resource sharing in the data center. In: *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, pp 295-308.
13. Verma A, Pedrosa L, Korupolu M, Oppenheimer D, Tune E, Wilkes J (2015) Large-scale cluster management at Google with Borg. In: *Proceedings of the 10th European Conference on Computer Systems*, pp 1-17.
14. Sefraoui O, Aissaoui M, Eleuldj M (2012) OpenStack: toward an open-source solution for cloud computing. *Int J Comput Appl* 55: 38-42.
15. Zhang Q, Zhu X, Zheng Z, Lyu M, Yang J (2020) Deep reinforcement learning-based approach for dynamic resource scheduling in cloud-edge environments. *IEEE Trans Netw Serv Manag* 17: 2393-407.
16. Mach P, Becvar Z (2017) Mobile edge computing: a survey on architecture and computation offloading. *IEEE Commun Surv Tutor* 19: 1628-56.
17. Deng L, Chen J, Shi Y, Zhang Y (2020) Communication-efficient federated learning for resource allocation in mobile edge computing. In: *2020 IEEE 40th International Conference on Distributed Computing Systems*, pp 788-98.
18. Bi S, Zhang YJ (2018) Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading. *IEEE Trans Wirel Commun* 17: 4177-90.
19. Mijumbi R, Serrat J, Gorricho JL, Bouten N, De Turck F, Boutaba R (2016) Network function virtualization: state-of-the-art and research challenges. *IEEE Commun Surv Tutor* 18: 236-62.
20. Zhang J, Guo H, Liu H (2020) Coordinated resource management for software-defined cloud data centers. *IEEE Trans Netw Serv Manag* 17: 652-65.
21. Kreutz D, Ramos FM, Verissimo PE, Rothenberg CE,

Azodolmolky S, Uhlig S (2015) Software-defined networking: a comprehensive survey. *Proc IEEE* 103: 14-76.

22. Arulkumaran K, Deisenroth MP, Brundage M, Bharath AA (2017) Deep reinforcement learning: a brief survey. *IEEE Signal Process Mag* 34: 26-38.

23. Zhang J, Guo H, Liu H, Guo C (2019) Multi-agent reinforcement learning for edge computing resource allocation. In: 2019 IEEE Global Communications Conference, pp 1-6.

24. Zhang C, Vinyals O, Munos R, Bengio S (2018) A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*.

25. McMahan B, Moore E, Ramage D, Hampson S, y Arcas BA (2017) Communication-efficient learning of deep networks from decentralized data. In: *Artificial Intelligence and Statistics*, pp 1273-82.

26. Zhuo J, Chen J, Wu Y (2019) Federated multi-agent reinforcement learning for IoT resource management. In: 2019 IEEE 39th International Conference on Distributed Computing Systems, pp 1234-43.

27. Liu R, Wei Y, Zhang J (2021) Federated reinforcement learning: challenges and opportunities. *IEEE Netw* 35: 168-75.

28. Wu Z, Pan S, Chen F, Long G, Zhang C, Yu PS (2021) A comprehensive survey on graph neural networks. *IEEE Trans Neural Netw Learn Syst* 32: 4-24.

29. Scarselli F, Gori M, Tsoi AC, Hagenbuchner M, Monfardini G (2009) The graph neural network model. *IEEE Trans Neural Netw* 20: 61-80.

30. Kipf TN, Welling M (2017) Semi-supervised classification with graph convolutional networks. In: *International Conference on Learning Representations*.

31. Velickovic P, Cucurull G, Casanova A, Romero A, Lio P, Bengio Y (2018) Graph attention networks. In: *International Conference on Learning Representations*.

32. Li Y, Tarlow D, Brockschmidt M, Zemel R (2016) Gated graph sequence neural networks. In: *International Con-*

ference on Learning Representations.

33. Zhao L, Song Y, Zhang C, Liu Y, Wang P, Lin T, Deng M, Li H (2020) T-GCN: a temporal graph convolutional network for traffic prediction. *IEEE Trans Intell Transp Syst* 21: 3848-58.

34. Luo Y, Xiong Y, Liu Y, Xu X (2020) Adaptive traffic prediction with graph neural networks for edge computing. In: 2020 IEEE 40th International Conference on Distributed Computing Systems, pp 1562-72.

35. Geyer F, Carle G (2018) Learning and generating distributed routing protocols using graph-based deep learning. In: *Proceedings of the 2018 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*, pp 40-5.

36. Zhou J, Cui G, Hu S, Zhang Z, Yang C, Liu Z, Wang L, Li C, Sun M (2020) Graph neural networks: a review of methods and applications. *AI Open* 1: 57-81.

37. Park S, Kim S, Chen X, Park S, Kim S (2019) Distributed learning for graph neural networks. In: 2019 IEEE 39th International Conference on Distributed Computing Systems, pp 1789-99.

38. Wang J, Guo H, Liu H (2021) GAT-based service function chain placement in NFV environments. *IEEE Trans Netw Serv Manag* 18: 2894-907.

39. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

40. Cheng Y, Anwar A, Wei X (2018) Alibaba cluster trace: analysis with a new public dataset. In: 2018 IEEE International Conference on Big Data, pp 1-8.

41. Hadary O, Marshall L, Menache I, Pan A, Gopalakrishnan A, Maior M, Qiu H, Chen E (2020) Protean: VM allocation service at scale. In: 14th USENIX Symposium on Operating Systems Design and Implementation, pp 845-61.

42. Khalil E, Dai H, Zhang Y, Dilkina B, Song L (2017) Learning combinatorial optimization algorithms over graphs. In: *Advances in Neural Information Processing Systems*, pp 6348-58.

Submit your manuscript to a JScholar journal and benefit from:

- ¶ Convenient online submission
- ¶ Rigorous peer review
- ¶ Immediate publication on acceptance
- ¶ Open access: articles freely available online
- ¶ High visibility within the field
- ¶ Better discount for your subsequent articles

Submit your manuscript at
<http://www.jscholaronline.org/submit-manuscript.php>