Task Scheduling in Edge Computing Environments: a Hierarchical Cluster-based Federated Deep Reinforcement Learning Approach

Latifah Alsalem*
School of Computer Science, University of Leeds
Leeds, UK
Shaqra University
K.S.A
Ml19laaa@leeds.ac.uk
lalsalem@su.edu.sa

Karim Djemame
School of Computer Science, University of Leeds
Leeds, UK
k.djemame@leeds.ac.uk

ABSTRACT

Edge computing (EC) reduces latency and response times, thereby enhancing the quality of service (QoS) for internet of things (IoT) applications and services. However, scheduling heterogeneous tasks with low latency can be challenging for EC in distributed and resource-constrained infrastructures. Thus, this paper offers a hierarchical federated deep reinforcement learning (HFLDRL) scheduling architecture with feedback-driven closed-loop learning between scheduling levels. A global agent (federated deep Q-network, FL-DQN$_c$) selects clusters for incoming tasks, while each cluster uses a locally pre-trained global agent (FL-DQN$_n$) to assign tasks to appropriate nodes. Experimental findings show that the hierarchical federated deep Q-network framework (HFL-DQN) outperforms HDQN scheduling under various simulated workloads. The proposed approach results in 50% more cumulative incentives for the global agent than the single agent, proving its scheduling effectiveness. Across nodes, it reduces average task delay by 87.80% and execution time by 85.27%. It also reduces the makespan by up to 40.4% across varying workloads and network latencies. These findings show that HFLDRL enhances distributed edge learning latency, scheduling, and scalability.

CCS CONCEPTS

• **Computer systems → Cloud computing**; • **Computer systems organization →** *Edge computing*; • **Computing methodologies →** *Reinforcement learning*; *Federated learning*.

KEYWORDS

Edge computing, task scheduling, hierarchical approach, federated learning, deep Q-network

1 INTRODUCTION

Edge computing (EC) is a distributed paradigm that moves processing and storage closer to user data to reduce latency and burden on cloud servers. Having closeness to the end user is crucial for latency-sensitive applications like real-time video analytics, industrial automation, augmented reality, and smart cars, as delays can hinder functionality and performance [7]. In EC systems, task scheduling is essential for distributing heterogeneous workloads, meeting deadlines, optimising resources, and maintaining users' experiences. Due to task heterogeneity, variable network conditions, limited resources, and distributed infrastructure, task scheduling is challenging, making standard heuristics or centralised techniques insufficient.

The decision-making process is a key element of task scheduling in EC environments. This process involves the selection of the appropriate edge node and the determination of the timing of task execution, particularly in the presence of multiple heterogeneous nodes [8]. Advanced machine learning techniques (ML), particularly reinforcement learning (RL) and deep reinforcement learning (DRL), improve decision-making by adjusting scheduling policies in real time to enhance latency, energy consumption, and load balancing more than heuristic methods [1]. Advanced ML techniques frequently outperform traditional methods due to their ability to learn from experience [15]. In addition, research show that collaboration of federated learning (FL) and DRL techniques can reduce task completion delays and improve the scheduling performance by optimising a global scheduling policy for heterogeneous workloads and low-latency targets at the EC environments [17, 19].

Although promising, most task scheduling methods focus on one level, usually the node level in the EC environment, which is inefficient. Node-level schedulers optimise CPU consumption and task queue length locally but ignore higher-level dynamics such as workload distribution. Inefficient decision-making in resource-constrained environments can result from a lack of awareness of local delays [4]. Single-level frameworks lack cross-layer coordination and knowledge transfer; thus, node agents learn from local experiences without affecting cluster-wide scheduling. Reduced task delays at the node level do not affect cluster-level decisions, leading to disconnected learning and missed latency reduction possibilities [16].

To address the challenge of reducing overall task latency at both the node and cluster levels within the EC environment, this paper introduces a hierarchical scheduling framework that integrates the FL and the DRL approaches at both node and cluster levels. The FL strategy employs a feedback mechanism that integrates local node results with global cluster policies to enhance coordination across levels. The objective is to minimise heterogeneous task delays by considering resource demands and task execution duration in a flexible, two-level scheduling system.

The task scheduling problem is modelled as a Markov Decision Process (MDP), a mathematical framework for decision-making when both randomness and agent decisions impact outcomes [10]. At the cluster level, a deep Q-network (DQN)-based task scheduling model is subsequently built to determine an efficient execution time, thereby reducing task delays across edge clusters. Based on

*Corresponding author.

this, an FL approach is proposed for DQN agents trained in heterogeneous environments to collaborate via multiple edge clusters. The federated averaging model (FedAvg), commonly used in FL applications [9], aggregates update parameters at the cluster level. Deploying the pre-trained FL-DQN$_n$ agent at the node level allows each cluster node to fine-tune it based on its local states and task profiles. The feedback loop enables continual improvement at both levels by frequently transmitting node-level updates back to the cluster-level scheduler.

According to simulation results, the hierarchical federated deep Q-network (HFL-DQN)-based scheduler enhances the agent's learning curve during training compared to a hierarchical deep Q-network (HDQN)-based scheduler. It also enhances latency, scheduling efficiency, and scalability in the EC environments. The contributions of this paper are as follows:

(1) Develop a scheduling architecture by conducting task scheduling at the cluster and node levels.
(2) Implement the FL-DQN$_c$ scheduler at the cluster level to enable collaborative scheduling among edge clusters.
(3) Integrate the cluster-level and node-level schedulers with a closed-loop feedback system to allow node-level performance metrics to inform cluster-level policies, and vice versa.

These demonstrate the viability of the proposed approach in efficiently coordinating hierarchical task scheduling within multi-cluster edge computing environments.

The paper is structured as follows: Section 2 reviews studies on task scheduling in EC environments, finds gaps, and explains how this study addresses them. Section 3 features a system model architecture that demonstrates heterogeneous task scheduling using DQN and FL techniques. Section 4 covers experimental evaluation, framework setup, and outcomes analysis. Section 5 finishes with a discussion on framework outcomes and future development.

2 RELATED WORK

Task scheduling is crucial in EC environments because resource-heterogeneous tasks must be effectively distributed throughout a network of resource-constrained edge devices. Shen et al. [13] introduce a Q-learning-based scheduler for task allocation in an End-Edge-Cloud (EEC) environment. Matrix representations are employed to formalise task and environmental states, and Q-learning is employed to dynamically allocate tasks in order to reduce task completion and queuing delay. Shahidani et al. [12] propose a multi-objective task scheduling framework in an edge-fog-cloud architecture using Reinforcement Learning with Feature Selection (RLFS), a two-stage RL method. This study addresses delay, load imbalance, and poor resource use in IoT-enabled systems due to edge device task loads increasing. Anand and Karthikeyan [3] propose the Efficiency-Aware Adaptive Deep Reinforcement Learning (EADRL) framework for dynamic task scheduling in edge–cloud environments. This framework solves real-time latency, resource limits, and fluctuating workloads. In order to optimise system load and response time in heterogeneous edge-fog environments, Wang et al. [15] provide a DRL-based IoT scheduling algorithm (DRLIS). This study examines the issue of congested edge-fog servers, dependence limitations in IoT workflows, and significantly fluctuating execution configurations. Zeng et al. [18] present an enhanced Double Deep Q-Network (DDQN) model, G-DDQN, which differentiates action selection from Q-value estimation over two networks. This study addresses the challenge of efficiently scheduling dynamic workloads in the EC environment to minimise the makespan and the maximum completion time across virtual machines.

The FL technique also allows resource-constrained edge nodes to collaborate and decentralise model training, reducing bandwidth and improving work scheduling policies. Choppara and Mangalampalli [6] introduce an adaptive fog computing task scheduling framework, combining F-LDQN and K-means clustering for better task prioritisation and decentralised learning. This study tackles the issues of latency, Service Level Agreement (SLA) breaches, and resource inefficiencies in a diverse fog environment. Alsalem and Djemame [2] introduce a FLDRL-based scheduling framework to reduce task delays in heterogeneous edge computing environments with time-sensitive applications. They schedule tasks for edge nodes with different resource capabilities and heterogeneous task requirements. Their approach combines models across nodes using FedAvg and a DQN scheduler at each edge cluster. Each cluster trains a local DQN agent via node-level state transitions, constantly sharing parameters with a global model through FL rounds. In an Unmanned Aerial Vehicle (UAV)-assisted Internet of Vehicles (IoV) environment, Wang et al. [14] propose a dependency-aware framework for task offloading and resource allocation using generative AI. Multi-agent Deep Deterministic Policy Gradient (MADDPG) facilitates real-time offloading decisions that involve dependency fulfilment, UAV mobility, and execution delay. The intelligent data caching and compute offloading (Fed-IDCCO) assist distant IoV agents in coordinating while preserving privacy.

Hierarchical task scheduling distributes decision-making into device, edge server, and cloud layers. This enables each layer to optimise the system's latency and usefulness while controlling local workloads and resource volatility. Sagar et al. [11] propose Hierarchical Adaptive Federated Reinforcement Learning (HAFedRL), a dual-tier architecture for effective resource allocation and task scheduling in hierarchical IoT networks. This study tackles the challenge of synchronising choices among distributed edge hosts with diverse resources, ensuring data privacy and enhancing convergence. Cai et al. [5] provide a framework for cloud-edge-end task decomposition and hierarchical scheduling to enhance service responsiveness in distributed collaborative computing. This method improves task scheduling across a three-tier architecture comprising end devices, edge servers, and the cloud in heterogeneous, latency-sensitive environments.

Current research illustrates significant adaptability and latency-aware scheduling at the node or edge–cloud level through the application of RL, DRL, and FL. Existing research also proposes a hierarchical learning framework to develop autonomous, collaborative-free node-level models for hierarchical task scheduling. These frameworks typically span the IoT, edge, and cloud layers, distributing subtasks across tiers based on processing capacity and latency. Although such multi-layer federation improves global coordination, it can also increase latency and communication overhead due to long-distance model exchanges between layers. Compared with

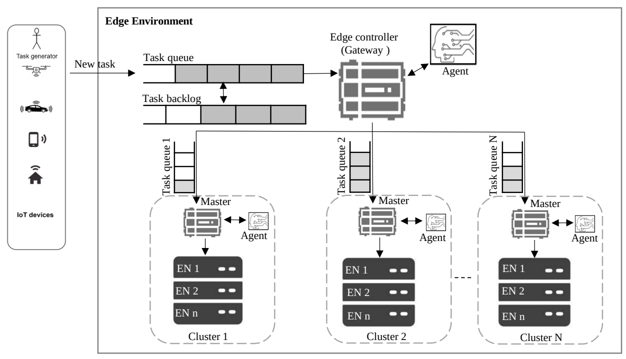

Figure 1: An overview of the system model architecture.

flat, single-tier task allocation, these solutions generally achieve lower end-to-end latency and higher execution reliability.

In contrast, the paper focuses exclusively on the edge layer and introduces a two-level hierarchical framework that separates decision-making between the cluster and node levels. This separation enables more fine-grained scheduling, where cluster-level agents optimise inter-cluster task distribution and node-level agents refine intra-cluster allocation. The proposed HFL-DQN architecture incorporates a closed-loop feedback mechanism between these two levels, allowing node-level performance metrics such as latency and queue efficiency to dynamically inform cluster-level policies. By maintaining all federated interactions within the same edge layer, the framework reduces communication delay between the FL server and local agents. Consequently, the dual-level, feedback-driven design enhances scheduling efficiency, coordination, scalability, and overall end-to-end performance in dynamic and heterogeneous edge environments.

3 SYSTEM MODEL ARCHITECTURE

This paper assumes that multiple EC environment instances participate in the training and testing phase. Each environment instance consists of an edge controller (gateway) managing multiple clusters with many nodes. Each cluster has its own waiting queue and consists of a number of nodes with different specifications. Nodes are classified by resource type (*small*, *imbalanced*, or *large*) and active time. The edge controller controls a backlog k and global queue q of heterogeneous tasks, each with different resource requirements and execution time. This system generates tasks dynamically using a *Poisson distribution*, reflecting stochastic arrival patterns in the EC environments. At each discrete timestep, incoming tasks are added up to a specified limit. If the global queue q reaches capacity, new tasks are temporarily stored in a backlog k. Figure 1 illustrates an overview of the system model architecture.

This framework trains and tests each DQN model in a separate EC environment instance with different state transitions. The FL technique distributes DQN agents across many EC environment instances to facilitate their joint learning of different policies. The FedAvg model aggregates parameters from all local agents tested across these EC environment instances to create a global agent (HFL-DQN-based scheduler). At the cluster level, the FL-DQN$_c$ scheduler assigns tasks to clusters with the shortest queue lengths and average resource utilisation. Subsequently, at the node level, the FL-DQN$_n$ scheduler assigns tasks from the cluster queue cq to nodes, prioritising the earliest response time and resource availability. Some tasks may be queued within occupied clusters to guarantee deferred execution. The FL approach establishes a feedback loop across multiple EC environment instances, enabling coordinated learning that reinforces both cluster-level and node-level scheduling decisions.

The main objective is to reduce the total delay of tasks in the EC environment. The total delay D_i from task generation to completion is defined in Eq 1 as the sum of transmission, waiting, processing decision, and execution delays.

$$D_i = d_t + d_{wq} + d_{wk} + d_p + d_e \tag{1}$$

The transmission delay d_t denotes the inter-arrival duration of tasks, characterised by a *Poisson distribution* with an arrival rate λ, where λ indicates the task arrival rate (tasks per second):

$$d_t = \frac{1}{\lambda} \tag{2}$$

The waiting delay consists of the task's waiting time in the backlog delay d_{wk}, the edge controller queue (global queue), and the cluster queue. The overall queuing delay d_{wq} comprises two hierarchical elements: the cluster-level delay d^c_{wq} and the node-level delay d^n_{wq}:

$$d_{wq} = d^c_{wq} + d^n_{wq} \tag{3}$$

The processing decision delay d_p denotes the duration utilised by the edge controller to identify a suitable cluster, as well as the time required by the cluster to allocate a task to a specified node:

$$d_p = d^c_p + d^n_p \tag{4}$$

Where d^c_p denotes the cluster selection delay, and d^n_p indicates the node selection delay. Although the processing decision delay d_p, which includes the time needed for cluster and node selection, contributes to the overall task delay, it is negligible compared to the queuing and execution delays. Consequently, to simplify the model and enhance its clarity, d_p is integrated into the hierarchical queuing delay d_{wq}, which already encompasses delays at both the cluster and node levels.

Upon finalising the scheduling decisions, the execution delay at node d_e is calculated as:

$$d_e = \frac{C}{f_n} \tag{5}$$

In this context, C denotes the quantity of CPU cycles necessary for task execution, while f_n represents the CPU speed of the node in cycles per second.

The total delay D_i for task i is calculated as:

$$D_i = \frac{1}{\lambda} + d_{wk} + d^c_{wq} + d^n_{wq} + d^c_p + d^n_p + \frac{C}{f_n} \tag{6}$$

With the integration of the d_p into the d_{wq}, the simplified delay D_i for task i is represented as:

$$D_i = \frac{1}{\lambda} + d_{wk} + d_{wq} + \frac{C}{f_n} \tag{7}$$

3.1 Problem Formulation

In this framework, the task scheduling problem is defined via an MDP model, which characterises the system as a collection of environmental states, actions, and rewards.

State. The system models the dynamic EC environment using a hierarchy of matrices that represent task characteristics and resource states at two levels. Task matrices record resource demands and duration times for the global queue and backlog tasks. Cluster matrices show how node-level capacities are allocated within each cluster. Node matrices show how nodes use resources over time to handle demands. Task, cluster-level resource, and node-level utilisation matrices form a layered EC environment representation.

Action space. The action considers cluster-level scheduling represented as a hierarchical scheduling level. The scheduler selects a cluster and allocates a task to it at each time step, considering queue length and average resource utilisation. If the scheduler selects an invalid action (no task is scheduled), time advances, accumulating new tasks in the queue and executing current tasks at cluster nodes. Equation 8 defines the action space size, which includes all conceivable cluster-level actions. All task-cluster scheduling and incorrect operations are included.

$$m \cdot q + 1 \tag{8}$$

where m denotes the number of clusters, q is the length of the global task queue, and the additional +1 accounts for the invalid action.

Reward. The reward function is designed to guide agents in minimising overall task delay across both cluster and node levels. Negative rewards correspond to increased system delays, whereas reduced delays yield comparatively higher rewards. Equation 9 computes the reward for each action within the system, including delays at both the node and cluster levels.

$$\begin{aligned}\text{Reward} = -\Bigg(\sum_{c} \Bigg(\frac{\alpha_i}{\sum_l \sum_{i_l} t_{i_{c,l}}} + \frac{\beta}{\sum_{q_n} \left(t_{i_q^n} + t_{i_p^n} \right)} \Bigg) \\ + \frac{\gamma}{\sum_{q_c} \left(t_{i_q^c} + t_{i_p^c} \right)} + \frac{\delta}{\sum_k t_{i_k}} + \frac{\zeta}{\sum t_{i_t}} \Bigg)\end{aligned} \tag{9}$$

$i_{c,l}$ represents all tasks scheduled for node l in cluster c. q_n represents node-level tasks, q_c represents cluster-level tasks, k represents all backlog tasks, and t represents the transmission time. Task duration t_i includes transmission, queue waiting, and execution time. Weights are introduced to provide flexibility in balancing the relative influence of different delay components on the overall task duration. All weights α_{i_c}, β_c, γ, δ, and ζ are set to 1 to ensure equal influence of transmission, queue waiting, and execution time, providing a neutral baseline without bias towards any component. The system can also adjust these weights to prioritise specific delay components, thereby enhancing scheduling performance at both cluster and node levels.

3.2 Hierarchical Deep Q Network-based Task Scheduling Model

In the HDQN scheduler, two DQN agents enable a two-level task scheduling framework in the EC environment. Each DQN agent evaluates Q-values and makes decisions using a multi-layer Convolutional Neural Network (CNN). The main function of each DQN is to dynamically interact with its hierarchically arranged cluster- and node-level states. Over multiple episodes, the cluster-level DQN$_c$ agent learns an efficient scheduling policy. It dynamically assigns tasks after training according to its knowledge, as shown in Figure 2. A pre-trained DQN$_n$ agent manages intra-cluster task scheduling at the node level. This agent is refined during deployment to enhance its adaptation to the different attributes of each cluster. For limited exploration, a minimal epsilon value ($\varepsilon = 0.01$) is used during fine-tuning to help the agent improve policy without compromising performance.

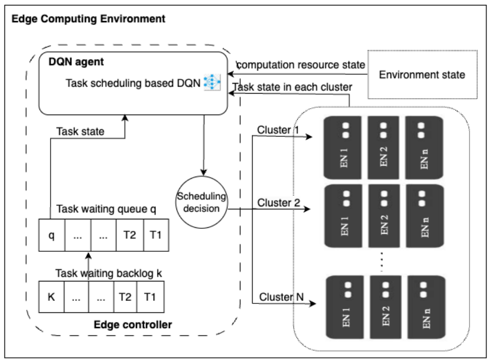

Figure 2: A Deep Q-Network-based task scheduling model at the cluster level.

Algorithm 1 represents HDQN-based task scheduling with a Feedback Loop.

Algorithm 1 HDQN-Based Task Scheduling with a Feedback Loop

```
1:  Input: Environment state $s_c$
2:  Output: HDQN model, scheduled tasks to cluster nodes
3:      Initialise DQN$_c$ parameters and CNN model
4:      Create experience buffer $\mathcal{B}$ with capacity $D$
5:      Set initial cluster state $s_c$
6:  while true do
7:      for episode $< E$ do
8:              Select cluster-level action $a_c$ using $\varepsilon$-greedy policy
9:              Schedule tasks to nodes, reward $r_n$, delay $d_n$
10:             Collect node-level feedback $r_n, d_n$
11:             Observe next cluster state $s'_c$, reward $r_c$, delay $d_c$
12:             Store transition $(s_c, a_c, r_c, d_c, s'_c)$ in buffer $\mathcal{B}$
13:             $r \leftarrow r_c + r_n, d \leftarrow d_c + d_n$
14:         if $|\mathcal{B}| > D$ then
15:             Sample mini-batch $(s_c, a_c, r, d, s'_c)$ from $\mathcal{B}$
16:             Perform gradient descent to minimise loss
17:             Update DQN$_c$ parameters
18:         end if
19:         State update: $s_c \leftarrow s'_c$
20:     end for
21: end while
22: Return: HDQN model
```

3.3 Hierarchical Federated Deep Q Network-Based Task Scheduling Model

In the HFL-DQN scheduler, several DQN agents are trained and tested in various environments to create an FL-DQN scheduler that meets level objectives. The FL technique at the cluster level enables the decentralised training of DQN$_c$ agents across multiple environments to benefit from shared knowledge, policy updates, and experience. The training process proceeds until the global model

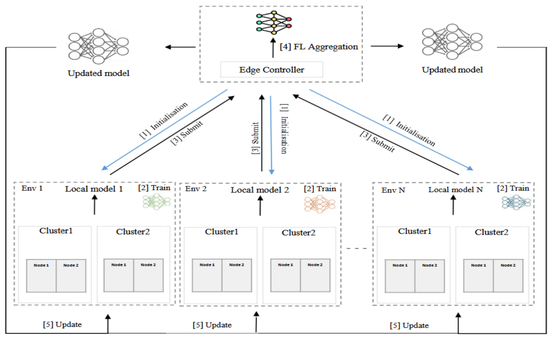

Figure 3: Federated learning training process at the cluster level.

converges or the maximum iteration limit is attained. Upon achieving convergence, the FL-DQN$_c$ model is implemented in the testing environment to execute task scheduling. Figure 3 illustrates the cluster-level FL training process, involving the central FL server and distributed environments with several clusters and nodes. Within the same strategy, the node-level FL-DQN$_n$ agent is trained under a different scheduling framework and subsequently employed for task scheduling inside clusters as part of the hierarchical framework.

Algorithm 2 represents HFL-DQN-based task scheduling with a Feedback Loop.

Algorithm 2 HFL-DQN-Based Task Scheduling with a Feedback Loop

```
1:  Input: Environment instances (Envs, each with pre-trained FL-DQN$_n$), FL aggregator
2:  Output: Global model HFL-DQN, scheduled tasks to cluster nodes
3:  Initialise DQN$_c$ model
4:  Distribute DQN$_c$ to all Envs
5:  for each round $r = 1$ to $R$ do
6:      for each DQN$_c$ in Envs do
7:          Select cluster-level action $a_c$ using $\epsilon$-greedy policy from DQN$_c$
8:          Offload $M$ tasks to the selected cluster $c$
9:          Schedule tasks to nodes, reward $r_n$, delay $d_n$
10:         Integrate node-level $(r_n, d_n)$ into cluster-level $(r_c, d_c)$
11:         Store transition $(s_c, a_c, r_c, d_c, s'_c)$ in buffer $\mathcal{B}$
12:         Update DQN$_c$ using gradient descent on mini-batches
13:     end for
14:     Aggregate updated models $\{\text{DQN}_c\}$ using FedAvg to obtain FL-DQN$_c$
15:     Distribute FL-DQN$_c$ to all Envs for the next round
16: end for
17: Return: Global model HFL-DQN
```

4 EXPERIMENTAL EVALUATION

The scheduling framework is built using Python (3.11) and TensorFlow (2.14.1). This framework is executed on a local high-performance computing infrastructure.

4.1 Implementation Setup

The current scheduling framework is trained and tested using dynamic Poisson-distributed synthetic data as a workload. The workload consists of three distinct task categories, each with a different duration time and resource requirements. First, *small tasks* require three distinct resources: CPU cores, memory (GB), and disc space (GB), along with a short duration. Second, *imbalanced tasks* have imbalanced resource requirements due to storage-heavy, memory-intensive, or CPU-bound placement, or it needs a long duration time. Third, *large tasks* necessitate substantial resource capacities

Table 1: Environment simulation parameters

Simulation Parameter	Value
Task duration average	(2 to 90) Seconds
Workload size	(250 to 5000)
Node active time	(10 to 100) Seconds
Number of clusters in each environment	(2 to 16)
Number of nodes in each cluster	(2 to 16)
Cluster queue, Global queue, Backlog (length)	5, 10, 20

Table 2: The enhanced CNN structure

Layer	Parameters	Activation
Input	$(H \times W \times 1)$ (spatial size)	–
Conv2D	16 filters, 3×3 kernel	ReLU
MaxPooling2D	Default stride	–
Dropout	0.2 drop rate	–
Conv2D	32 filters, 3×3 kernel	ReLU
MaxPooling2D	Default stride	–
Dropout	0.2 drop rate	–
Dense	256 units	ReLU
Output	*output_shape* (Q-values)	Linear

Table 3: Scheduling hyper-parameters

Parameter	Value
episodes per agent	500
Learning rate α	0.01
batch size	32
exploration probability ϵ	0.99
FL rounds	(4 to 12)
FL technique	FedAvg
FedAvg aggregation	After every round
FedAvg frequency update	32 steps
Number of local agents	(2 to 8)

and long durations. Within the scheduling framework, the task generator generates workload by adjusting batch sizes for each task type, sending it online to a fixed-length global queue q and backlog k. The framework models heterogeneous multi-cluster environments with different node specifications. Table 1 captures the hyper-parameters of the simulation environment, encompassing the workload description. All hyper-parameters for each framework environment instance are identified through multiple tests, all of which yield satisfactory outcomes.

In the HFL-DQN framework, the CNN model is designed to support multi-dimensional task-node summary matrices and accept dynamic input sizes. The enhanced CNN is added as a component of the HDQN-based scheduling model. Table 2 illustrates the enhanced CNN structure for dynamic input as a component of the HDQN-based scheduling model.

Then, the FL technique is integrated into the scheduling framework by training HDQN agents independently in different environmental scenarios throughout multiple FL rounds. Table 3 outlines the hyper-parameters specific to HFL-DQN scheduling that have been configured within the system.

4.2 Results and Discussion

The results compare the performance of the global agent (HFL-DQN scheduler) and the single agent (HDQN scheduler) during training and testing across multiple scheduling experiments in multi-cluster environments. During training, Figure 4 illustrates the performance

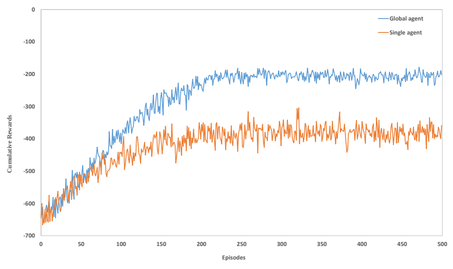

Figure 4: Cumulative rewards over episodes.

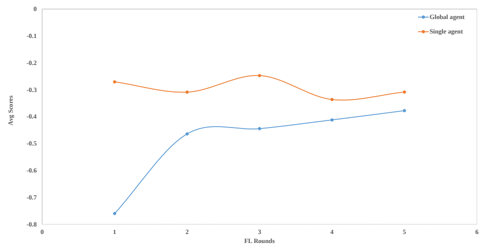

Figure 5: Average scores over federated rounds.

evolution of the two agents across 500 episodes. Both agents initially achieve comparable cumulative rewards; however, as training progresses, their performance diverges markedly. The global agent, trained through federated aggregation across multi-cluster environments, exhibits a steeper learning curve, indicating a faster learning rate and more effective policy adaptation. Specifically, the global agent converges after approximately 200 episodes, whereas the single agent requires around 250 episodes to reach stability. This behaviour confirms that federated aggregation accelerates policy learning by enabling cross-cluster knowledge sharing, leading to faster convergence and higher cumulative rewards. By the end of training, the global agent attains substantially better performance, achieving an average cumulative reward of approximately −200, compared to −400 for the single agent an improvement of about 50%. The single agent, by contrast, shows slower convergence and lower reward stability, highlighting the advantages of hierarchical federated learning in distributed edge environments.

To evaluate the proposed scheduling framework's accuracy during the training phase, this experiment compares the scheduling accuracy measured by the average scores of a global agent and a single agent across federated learning (FL) rounds. Figure 5 shows that the global agent begins with a lower average score (approximately −0.77) than the single agent (approximately −0.35), indicating limited initial policy generalisation. However, the global agent exhibits consistent improvement over five FL rounds, eventually achieving an average score of approximately −0.42. In contrast, the single agent remains relatively stable, fluctuating between −0.35 and −0.25. This represents an improvement of about 45.45% in the global agent's final average score compared to its initial performance, highlighting superior adaptability and policy refinement relative to the single agent's restricted progress under isolated training.

Despite the scalability effect introduced by federated learning, the global agent demonstrates efficient convergence behaviour, reaching stable performance after approximately the third federated round in this experiment. It is important to note that convergence time may increase proportionally with the environment size and the number of federated participants, as larger systems typically introduce greater heterogeneity and communication overhead.

In the testing phase, this experiment evaluates the efficiency of the HFL-DQN scheduling framework by comparing global and single agents waiting times at three task arrival rates (λ = 2, 5, and 10). The total waiting time includes backlog, global, and cluster queue times. Table 4 compares total waiting times at various task arrival rates, showing that the global agent (G) outperforms the single agent (S) in system delay reduction. At a low arrival rate $\lambda = 2$, the global agent records slightly decreased waiting times for backlog, global queue, and cluster queue, showing more efficient scheduling even with light workloads. As the arrival rate increases to $\lambda = 5$, single agent delays increase dramatically, particularly in the global queue (2951 vs. 2547), showing a reduced capacity to handle increased demand. This trend remains at $\lambda = 10$, with the global agent performing consistently and the single agent experiencing delays in all system queues. The HFL-DQN approach for cross-cluster coordination and policy dissemination improves global agent scalability and responsiveness under system stress. Conversely, the single agent struggles to adapt, resulting in long wait times, especially as task arrival rates increase.

Table 4: Total waiting time at different arrival rates (λ).

Metric	G λ=2	S λ=2	G λ=5	S λ=5	G λ=10	S λ=10
Backlog	4178	4248	4291	4311	4291	4331
Global queue	2456	2559	2547	2951	2756	3019
Cluster queue	522	544	543	578	566	611

In the second experiment, this experiment evaluates the efficiency of the HFL-DQN scheduling framework by analysing the average task delay in milliseconds (ms) of global (G) and single (S) agents as the number of nodes per cluster increases, as illustrated in Table 5. As the number of nodes increases from 2 to 16, the global agent consistently achieves reduced average task delay, indicating superior scalability and efficiency in distributed environments. At two nodes, the single agent demonstrates a markedly greater delay of 106.32 *ms*, in contrast to 15.43 *ms* for the global agent. While both agents benefit from heightened node counts, the global agent's delay significantly decreases from 15.43 *ms* to 10.52 *ms*, whereas the single agent's delay stays comparatively elevated, as seen in the table. On average, the global agent achieves an 87.80% reduction in task delay compared to the single agent, highlighting its enhanced adaptability through the HFL-DQN technique to augment scalability and decrease scheduling latency as system resources increase.

The third experiment examines the scaling of average execution time per task in relation to system size by altering the number of nodes within a cluster. The comparison between global and single agents demonstrates a distinct performance disparity across all configurations, as seen in Table 5. As the node count increases from 2 to 16, the global agent regularly attains markedly reduced average

Table 5: Average delay and execution time per task across different node counts

Nodes	G avg delay	S avg delay	G avg exec	S avg exec
2	15.43	106.32	1.8	11.8
4	12.26	104.22	1.6	11.0
8	11.74	98.87	1.6	10.8
16	10.52	98.79	1.5	10.5

Table 6: Makespan across different workloads and network latencies with percentage improvement of the global model.

Tasks	Latency	G makespan	S makespan	Improvement (%)
250	0 ms	861	1445	40.41
250	5 ms	2235	2927	23.63
250	10 ms	3117	3365	7.37
500	0 ms	1159	1163	0.34
500	5 ms	3641	3825	4.81
500	10 ms	6105	6729	9.29
1000	0 ms	2145	2185	1.83
1000	5 ms	7063	7133	0.98
1000	10 ms	12431	12511	0.64
5000	0 ms	10151	10193	0.41
5000	5 ms	35311	35461	0.42
5000	10 ms	60307	60381	0.12

execution times, decreasing from 1.85 *ms* to 1.5 *ms* per task. Conversely, the single agent demonstrates elevated, albeit diminishing, execution times, varying from 11.8 *ms* to 10.5 *ms*. On average, the global agent achieves an 85.27% reduction in average execution time per task compared to the single agent, demonstrating that although both agents gain from supplementary nodes, the global agent is significantly more adept at utilising greater system sizes to enhance task scheduling and minimise execution latency. These results highlight the global agent's enhanced scalability throughout the scheduling phase, particularly when the underlying cluster capacity increases.

The fourth experiment assesses the scalability of the makespan, which denotes the overall time required to complete a workload, under different workloads and network latencies for both global (G) and single (S) agents. Since a lower makespan indicates faster task completion and better scheduling performance, the results in Table 6 shows that the global agent consistently outperforms the single agent in all configurations. The most pronounced relative improvement occurs with 250 tasks at 0 *ms* latency, where the global agent achieves a makespan of 861 *ms* compared to 1445 *ms* for the single agent, a reduction of approximately 40.4%. While the percentage difference decreases as task volumes and latencies increase, the global agent demonstrates enhanced resilience, indicating a more effective management of inter-cluster communication delays. On average across all workloads and network latencies, the global agent achieves a 7.5% reduction in makespan compared to the single agent, further demonstrating its efficiency in completing tasks more quickly under diverse conditions. These findings validate the global agent's superior scalability and stability in scheduling during load and network stress.

4.3 Scalability, Communication Overhead, and Computational Complexity

Training Phase. During training, the computational complexity of the proposed HFL-DQN framework increases linearly with the number of participating agents, the workload size, and the environment scale, as both local DQN updates and federated aggregation contribute to the overall runtime. Each training round consists of local model training for the DQN$_c$ agents, followed by a FedAvg aggregation step, resulting in a total complexity of $O(E \times C \times N)$, where E, C, and N denote the number of episodes, clusters, and nodes, respectively.

On the HPC setup used in this study (NVIDIA A100 GPU), complete training of two agents within an environment comprising four clusters and two nodes per cluster required approximately 2–3 hours, whereas scaling to eight agents within an environment comprising sixteen clusters increased the duration to approximately 32–48 hours. The communication overhead associated with model exchanges during the FedAvg process accounted for less than 10% of the total runtime, confirming the scalability of the framework under larger multi-cluster configurations. This efficiency is primarily attributed to the co-location of the federated server and participating agents within the same edge layer, which minimises network latency and communication cost.

Testing Phase. During inference, the computational load is significantly reduced because the scheduler performs a single forward pass through the trained network to select optimal actions. Communication overhead at test time is negligible, as the global model parameters are loaded once and used directly for task scheduling without repeated exchanges with the FL server. The overall inference time depends primarily on the workload size and increases approximately linearly with the number of tasks to be scheduled. In practical evaluation, the scheduling latency per decision remained on the order of milliseconds, demonstrating that the proposed hierarchical framework enables real-time task allocation with minimal computational and communication cost.

5 CONCLUSION

This paper investigated the problem of reducing heterogeneous task delay in a heterogeneous EC environment, specifically within a multi-cluster architecture. A hierarchical task scheduling framework was proposed to minimise task delay at two levels: the cluster level and the node level. The framework is based on the HFL-DQN technique, which periodically aggregates multiple HDQN agents trained in different environment instances. The proposed framework focuses on two-level scheduling and incorporates closed-loop feedback learning, an iterative process through which insights acquired at one level (node or cluster) reinforce learning and improve performance at the other. Simulation experiments demonstrate that the HFL-DQN-based scheduling framework is suitable for delay-sensitive tasks. The experimental results show that the global agent consistently outperforms the single agent by achieving faster convergence, lower task delay, and improved scalability across multi-cluster environments. These findings represent the first stage of evaluating the proposed scheduling framework. The current experiments employ synthetic workloads to ensure a controlled assessment of scalability and learning behaviour. Future work will incorporate real-world datasets and compare the proposed framework with additional scheduling baselines, including heuristic and meta-heuristic approaches, to provide a more comprehensive empirical validation. Furthermore, future research will extend the framework using alternative federated learning strategies, such as Federated Proximal Optimisation (FedProx), which is designed to enhance robustness under heterogeneous data distributions and computing conditions.

REFERENCES

[1] Zahra Jalali Khalil Abadi, Najme Mansouri, and Mohammad Masoud Javidi. 2024. Deep Reinforcement Learning–Based Scheduling in Distributed Systems: A Critical Review. *Knowledge and Information Systems* 66, 10 (2024), 5709–5782. https://doi.org/10.1007/s10115-024-02167-7

[2] Latifah Alsalem and Karim Djemame. 2024. Federated Learning–Based Heterogeneous Task Scheduling in Edge Computing Environments. In *Proceedings of the 2024 IEEE/ACM 17th International Conference on Utility and Cloud Computing (UCC)*. IEEE. https://doi.org/10.1109/UCC63386.2024.00079

[3] J. Anand and B. Karthikeyan. 2025. EADRL: Efficiency-Aware Adaptive Deep Reinforcement Learning for Dynamic Task Scheduling in Edge-Cloud Environments. *Results in Engineering* (2025), 105890. https://doi.org/10.1016/j.rineng.2024.105890

[4] Amin Avan, Akramul Azim, and Qusay H. Mahmoud. 2023. A State-of-the-Art Review of Task Scheduling for Edge Computing: A Delay-Sensitive Application Perspective. *Electronics* 12, 12 (2023). https://doi.org/10.3390/electronics12122599

[5] Jun Cai, Wei Liu, Zhongwei Huang, and Fei Richard Yu. 2024. Task Decomposition and Hierarchical Scheduling for Collaborative Cloud–Edge–End Computing. *IEEE Transactions on Services Computing* 17, 6 (2024), 4368–4382. https://doi.org/10.1109/TSC.2024.10412345

[6] Prashanth Choppara and S. Sudheer Mangalampalli. 2025. Adaptive Task Scheduling in Fog Computing Using Federated DQN and K-Means Clustering. *IEEE Access* 13 (2025), 75466–75492. https://doi.org/10.1109/ACCESS.2025.3563487

[7] Fateneh Golpayegani, Nanxi Chen, Nima Afraz, Eric Gyamfi, Abdollah Malekjafarian, Dominik Schäfer, and Christian Krupitzer. 2024. Adaptation in Edge Computing: A Review on Design Principles and Research Challenges. *ACM Transactions on Autonomous and Adaptive Systems* 19, 3 (2024), 1–43.

[8] Nidhi Kumari, Anirudh Yadav, and Prasanta K. Jana. 2022. Task Offloading in Fog Computing: A Survey of Algorithms and Optimization Techniques. *Computer Networks* 214 (2022), 109137. https://doi.org/10.1016/j.comnet.2022.109137

[9] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS) (Proceedings of Machine Learning Research, Vol. 54)*, Aarti Singh and Jerry Zhu (Eds.). PMLR, Fort Lauderdale, FL, USA, 1273–1282. https://proceedings.mlr.press/v54/mcmahan17a.html

[10] Martin L. Puterman. 1990. Markov Decision Processes. In *Handbooks in Operations Research and Management Science*. Vol. 2. Elsevier, 331–434.

[11] A. S. M. Sharifuzzaman Sagar, Amir Haider, and Hyung Seok Kim. 2025. A Hierarchical Adaptive Federated Reinforcement Learning for Efficient Resource Allocation and Task Scheduling in Hierarchical IoT Network. *Computer Communications* (2025). https://doi.org/10.1016/j.comcom.2024.107349

[12] Zahra Ramezani Shahidani, Mohammad Reza Khayyambashi, Mohammad Hossein Yaghmaee, and Amir Masoud Rahmani. 2023. RLFS: Multi-objective Reinforcement Learning–based Task Scheduling in Fog-Cloud Environments. *Computing* 105, 5 (2023), 1497–1525. https://doi.org/10.1007/s00607-022-01147-5

[13] Wangbo Shen, Weiwei Lin, Wentai Wu, Haijie Wu, and Keqin Li. 2025. Reinforcement Learning–Based Task Scheduling for Heterogeneous Computing in End-Edge-Cloud Environment. *Cluster Computing* 28, 3 (2025), 179. https://doi.org/10.1007/s10586-024-04828-2

[14] Xing Wang, Chao He, Wenhui Jiang, Wanting Wang, and Xiaoyan Liu. 2025. Generative AI–Based Dependency-Aware Task Offloading and Resource Allocation for UAV-Assisted IoV. *IEEE Open Journal of the Communications Society* 6 (2025), 3932–3949.

[15] Zhiyu Wang, Mohammad Goudarzi, Mingming Gong, and Rajkumar Buyya. 2024. Deep Reinforcement Learning–Based Scheduling for Optimizing System Load and Response Time in Edge and Fog Computing Environments. *Future Generation Computer Systems* 152 (2024), 55–69. https://doi.org/10.1016/j.future.2023.10.012

[16] Qi Xia, Winson Ye, Zeyi Tao, Jindi Wu, and Qun Li. 2021. A Survey of Federated Learning for Edge Computing: Research Problems and Solutions. *High-Confidence Computing* 1, 1 (2021). https://doi.org/10.1016/j.hcc.2021.100008

[17] Shuai Yu, Xu Chen, Zhi Zhou, Xiaowen Gong, and Di Wu. 2020. When Deep Reinforcement Learning Meets Federated Learning: Intelligent Multi-timescale Resource Management for Multiaccess Edge Computing in 5G Ultradense Network. *IEEE Internet of Things Journal* 8, 4 (2020), 2238–2251. https://doi.org/10.1109/JIOT.2020.3012159

[18] Lei Zeng, Qi Liu, Shigen Shen, and Xiaodong Liu. 2024. Improved Double Deep Q Network–Based Task Scheduling Algorithm in Edge Computing for Makespan Optimization. *Tsinghua Science and Technology* 29, 3 (2024), 806–817. https://doi.org/10.26599/TST.2023.9010058

[19] Xu Zhao, Yichuan Wu, Tianhao Zhao, Feiyu Wang, and Maozhen Li. 2024. Federated Deep Reinforcement Learning for Task Offloading and Resource Allocation in Mobile Edge Computing-Assisted Vehicular Networks. *Journal of Network and Computer Applications* 229 (2024), 103941. https://doi.org/10.1016/j.jnca.2024.103941