Proceedings Paper:
Alsalem, L. and Djemame, K. (2025) Federated Learning-Based Approach for Heterogeneous Task Scheduling in Edge Computing Environments. In: 2024 IEEE/ACM 17th International Conference on Utility and Cloud Computing (UCC). 17th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2024), 16-19 Dec 2024, Sharjah, UAE. IEEE, pp. 509-516. ISBN: 979-8-3503-6721-8.

https://doi.org/10.1109/UCC63386.2024.00079

Federated learning-based approach for heterogeneous task scheduling in edge computing environments

Latifah Alsalem*†, Karim Djemame*
*School of Computer Science, University of Leeds, Leeds, UK
Ml19laaa@leeds.ac.uk, k.djemame@leeds.ac.uk
†Shaqraa University, K.S.A
lalsalem@su.edu.sa

***Abstract*—Edge computing (EC) aims to facilitate internet of things (IoT) applications and services with low latency at the edge, thereby reducing response time and providing quality of service (QoS). However, increasing user demand for low-latency applications has highlighted the need to reduce task completion time delay in EC environments. Therefore, this paper introduces an approach for heterogeneous task scheduling in heterogeneous EC environments to minimise task delay for time-sensitive applications. A combination of deep reinforcement learning (DRL) and federated learning (FL) techniques is used to build the scheduling framework. Initially, the deep Q network (DQN)-based scheduling framework is employed to reduce the delay of tasks generated on an edge cluster within heterogeneous nodes. For collaborative learning, DQN agents are trained in different edge clusters for multiple FL rounds. The federated averaging model (FedAvg) is applied to calculate the average of the parameters of each trained agent in every FL round to generate a global agent that improves task completion time across the entire system. Compared to the standard DQN-based scheduling model, simulation results show that using the FL technique improves the learning curve over training time, reducing task delay and speeding up processing by about 50% in scheduling tests. Furthermore, collaborative learning by all trained agents confirms the global agent's stability and improvement, in contrast to the individual agent's fluctuating performance.**

***Index Terms*—edge computing, task scheduling, federated learning, deep Q network.**

I. Introduction

Delay-sensitive tasks generated by internet of things (IoT) devices have recently become easier to execute due to the advancement of computing resources and storage capacities at the edge of the network [8]. The increased focus on this progress in edge computing (EC) is mainly due to its capacity to fulfil the growing requirements of time-critical IoT applications [18]. Edge AI, which involves the direct deployment of artificial intelligence (AI) algorithms on edge devices, has primarily driven this advancement. Furthermore, the distributed structure of the EC environment is well-suited to the attributes of the federated learning (FL) technique, an advanced machine learning (ML) technique [15]. In the FL approach, edge nodes locally train the AI model required by the FL server, transmitting only model updates, such as gradients and weights, to the FL server for aggregation. The FL server then transmits the combined parameters for the subsequent training cycle. This iterative process continues until the model achieves an acceptable efficiency level.

FL techniques have been demonstrated to be effective in a variety of applications, particularly resource-constrained environments such as the EC environment [17]. Incorporating FL techniques into collaborative edge computing effectively reduces communication overhead by enabling edge nodes to locally train models. Therefore, it eliminates the need to transfer large datasets to a central server, a crucial feature in environments with limited resources. In addition, FL can customise models to the local environment, particularly in scenarios where centralised training may be hindered by significant unpredictability or system heterogeneity. It also increases the ability to handle larger workloads and system durability, reduces time delay, improves real-time task processing, enhances the quality of learning results, and accelerates convergence.

In terms of enhancing EC quality of service (QoS) provision, the implementation of efficient task scheduling algorithms is a crucial element of the EC environment, particularly to reduce task delay. Current research primarily focuses on the spatial dimension, which involves offloading requirements to more capable edge nodes. Several studies have proposed a variety of approaches for task scheduling, including traditional, heuristic, and ML-based methods [1]. The ML approach typically assumes that the environment has a fixed state transition. However, in real-world scenarios, an agent is required to handle simultaneously different state transitions in multiple environments [7], such as EC environments. As a result, learning scheduling policies surpasses the previously mentioned standard assumption.

Heterogeneous environments such as EC nodes have different capabilities and active times. In the EC environment, optimising task delay for heterogeneous task scheduling remains a significant challenge. This is because incoming tasks have heterogeneous resource requirements and deadlines. Therefore, this paper considers the heterogeneity of tasks, each with

This research is sponsored by Shaqraa University, KSA.

different requirements and estimated time. In addition, each agent is trained in a heterogeneous environment, represented by a cluster with multiple nodes. The main contribution of this paper is a scheduling framework that builds upon applying the FL technique to a deep reinforcement learning (DRL)-based task scheduling model, which is a deep Q network (DQN)-based task scheduling model. The DQN model is a form of DRL that involves using neural networks to approximate Q-values, aiding in decision-making processes. The focus is on minimising the total delay of tasks, considering each task's resource requirements and execution time within a flexible scheduling framework. To reduce task delay, the task scheduling problem is formulated as a Markov Decision Process (MDP), a mathematical construct used to model decision-making in contexts where results are influenced by both chance and the agent's choices [11]. The MDP model is commonly used in reinforcement learning (RL) to model decision-making problems. The DQN-based task scheduling model is then developed to determine an efficient execution time, minimising task delay across the cluster nodes.

Building on this, an FL technique is introduced for collaboration between multiple DQN agents trained in different instances of cluster environments. The federated averaging model (FedAvg) is a widely used FL algorithm that commonly forms the fundamental framework in several FL applications [9]. Compared to a single DQN agent in the cluster, the simulation experiments demonstrate that applying the FL technique improves the agent's learning curve across the training phase. The FL-based task scheduling framework reduces the total task delay and increases stability over several rounds, leading to higher system performance efficiency. Additionally, incorporating the FL technique improves the scheduler performance while increasing the number of heterogeneous nodes in the cluster.

This paper is structured as follows: Section 2 summarises relevant research on task scheduling methods in EC environments, identifies the gap in previous studies, and outlines the role this paper will play in addressing it. Section 3 explains the system model architecture, including the approach for scheduling various tasks using a single DQN and FL model. Section 4 presents the experimental evaluation, which involves the framework implementation details and a discussion of the results. Section 5 concludes with the proposed framework results, limitations, and future work.

II. Related work

Task scheduling in EC environments can be challenging due to IoT application mobility, network hierarchy, restricted resource capabilities, heterogeneity, and stochastic behaviour patterns. Most of the proposed algorithms focus on distributing workloads among geographically dispersed edge devices. Due to the intricate and ever-changing nature of these environments, the ML and FL techniques have proven to be highly effective in enhancing task-scheduling processes. Tuli et al. [13] developed an RL-based stochastic dynamic scheduler using an asynchronous-advantage-actor-critic (A3C) algorithm for decentralised learning of many agents in an edge-cloud environment. They also used a residual recurrent neural network (R2N2) framework to harness temporal patterns for hybrid scheduling. Simulations with iFogSim and CloudSim showed that its scheduler can rapidly adjust to a dynamic environment and improve energy use, reaction time, and cost.

Based on the concept of optimising task scheduling in edge computing, Gazori et al. [5] suggested scheduling fog-based IoT tasks using the DRL method to reduce energy consumption, latency, and computation cost while meeting resource and task deadlines. They used double deep Q-learning (DDQL) scheduling and gateways as schedulers and agents. Many virtual machines (VMs) assigned to incoming tasks shared the compute node's resources. In a broad state-action space, the DQN and double DQN algorithms outperformed the simple Q-Learning algorithm. Further advancing the application of DRL in task scheduling, Meng et al. [10] addressed the problem of latency-sensitive task scheduling and resource management on the server side in multi-user of the MEC. They introduced an online DRL-based method to reduce latency and timeout period of queued tasks. They also created a reward function called double neural network (DNN) to let the algorithm schedule tasks and manage resources directly through experience. The simulation results demonstrated that their method outperformed several traditional algorithms, such as FCFS, SJF, Packer, and Random, and had a significant advantage in terms of intelligence and environmental awareness.

Recent EC studies have proposed a federated or collaborative learning technique where different edge deployments share ML models, enabling them to leverage all accessible datasets without transferring them. This approach has demonstrated potential for improving collaborative task scheduling in edge/cloud environments [20]. For instance, Awada et al. [6] introduced AerialEdge, a framework for FL-based orchestration of an aerial EC system. They demonstrated a federated multi-output linear regression model (LR) to estimate task durations and resource requirements. They choose the drone implementation with the most resources and the longest flight time to complete tasks at any given time. Extensive studies utilise task dependencies and Alibaba's cluster trace data. AerialEdge used cluster resources more efficiently and executed multi-tasks faster than Spear, Graphene, Tetris, and the random approach.

Expanding upon the idea of federated learning in edge environments, Wang et al. [16] demonstrated the paradigm of wireless-powered mobile edge computing (WP-MEC). They combined FL and DRL with WP-MEC to simultaneously maximise computational and communication resources. In terms of average task execution latency and task completion ratio, the superiority of the created framework was evaluated and compared to the energy-aware scheduling (EAS) and local computing (LC) benchmark algorithms. In addition, Xia et al. [19] investigated the FL approach to improve edge intelligence and maintain data privacy, addressing the demand for efficient edge network processing. Their work optimised FL user scheduling strategies to handle training delays using update importance and latency reduction methods. In cases without prior node information, they used the multi-armed

bandit to balance exploration and exploitation. Simulation-based evaluations confirmed these strategies' efficacy. To schedule tasks for multiple edge nodes (EN) in an MEC system, Shi et al. [12] used the FL technique as a collaborative framework. The work offers a DQN-based task scheduling approach to enhance edge node computational task execution order. The study uses the FL approach to aggregate edge nodes to extract global parameters that optimise task completion latency across the MEC system. Simulations prove the proposed algorithm's superiority in latency-sensitive tasks and reveal the key elements affecting system performance.

Previous research has examined the application of FL and ML techniques in EC environments for task scheduling, with each study focusing on specific objectives to improve the edge QoS. While these studies have made significant contributions to understanding a variety of aspects of the FL technique in task scheduling, they have not explicitly addressed the issues associated with heterogeneous task delay in heterogeneous environments. As a result, there is still a significant gap in the scheduling of heterogeneous tasks for delay-sensitive applications in heterogeneous environments. This gap presents a substantial opportunity to implement a collaborative learning approach by combining the DQN and FL techniques to improve task scheduling and consider the heterogeneity of task resource requirements and deadlines and the heterogeneity of environment specifications and active time. The main objective is to improve the scheduling framework for minimising the total task delay in heterogeneous environments over time.

III. System model architecture

This paper considers deploying an FL-based task scheduling framework in heterogeneous EC environments. In this framework, each ML model (DQN agent) undergoes training and testing in a separate EC environment with different state transitions. The FL approach is included in the scheduling framework to distribute the DQN agents among all EC environments and acquire knowledge from different policies in each EC environment. Then, a global agent is created for the FL-based scheduler by aggregating all agents' parameters in EC testing environments.

A heterogeneous environment involves a cluster with multiple nodes, each possessing distinct resource types (e.g., small, imbalanced, large) and active time for each task. Within this cluster is a queue of heterogeneous tasks (q), each of which requests a predetermined quantity of different resources for a specified duration of time. The tasks are generated with different resource requirements and estimation times and then sent to be scheduled on cluster nodes. Each discrete time step lodges incoming tasks in the queue q of a fixed length. When the queue is full, the number of additional tasks is placed in a backlog k. In a concurrently running process, the agent (scheduler) selects and allocates tasks to appropriate nodes with the earliest response time and availability of resources. The scheduler assigns a task to an occupied node to be processed later. As long as the queue is full, the scheduler continues to assign new tasks, progressively moving the scheduled tasks forward. According to this framework, the FL technique among multiple cluster instances is considered an aggregator between environments and supports the scheduling decision. Fig. 1 provides an overview of the system model.

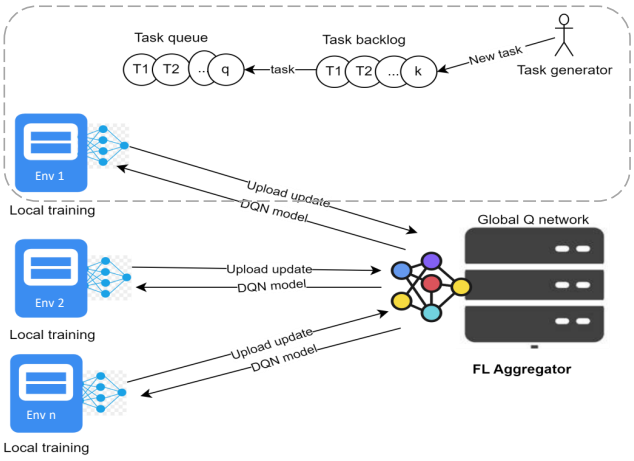

Fig. 1: The overview of system architecture.

The main objective is to minimise the total delay of tasks in the EC system. Equation (1) defines the total delay D_i of a task from its generation to completion consisting of execution delay and waiting delay as follows.

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

The execution delay is related to the task's execution time in the edge node (d_e), while the waiting delay is related to the task's waiting time in the queue (d_{wq}) and the task's waiting time in the backlog (d_{wk}).

A. Problem Formulation

This paper presents the task scheduling problem MDP model that represents the system as an assemblage of cluster state, action, and reward. Table I shows the mathematical notations used for their description.

TABLE I: Mathematical notations

Notation	Meaning
D_i	Total task delay
d_e	Execution delay
d_{wq}	Waiting delay in queue
d_{wk}	Waiting delay in backlog
t	Time
l	Node
q	Task queue length
k	Task backlog length
n	Number of nodes
i_l	Scheduled tasks
α_i	Transmission speed
β	Discount factor

State. The cluster state denotes the task description (resource requirements and duration time) and the node specification (resource capacity and active time). The cluster state is considered binary matrices representing data regarding nodes and tasks. State includes the current status of node resources in the

cluster and the task resource requirements in the queue, which collectively constitute the system state at a certain time with a different value for each cluster instance. Fig. 2 illustrates the representation of the cluster state. Colours are employed only to represent distinct tasks. Coloured squares denote occupied resource slots, while white squares denote vacant resource slots. The node matrices indicate the status of scheduled tasks across various nodes. The task slot matrices illustrate the resource requirements of the queued tasks.

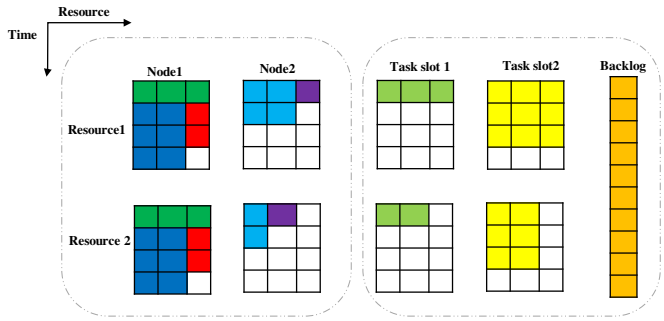

Fig. 2: State representation.

Action space. Action is defined as scheduling a task from the queue to a node in the cluster. During each time step, the scheduler schedules one task at the earliest available time in the node for each valid action, thereby altering the system state. When the scheduler selects an invalid action (no task is scheduled), time progresses, resulting in the addition of new tasks to the queue and the processing of tasks at the nodes in each cluster instance. Equation (2) defines the action space size, which represents the total number of possible actions, including all possible task-node scheduling and the invalid action, as follows:

$$n \cdot q + 1 \tag{2}$$

n is the number of cluster nodes, and q is the length of the queue.

Reward. The reward is designed to help the agent minimise the total task delay over time. Equation (3) calculates the reward for each action in each cluster instance is defined as:

$$\text{Reward} = -\sum_{l} \left(\frac{\alpha_i}{\sum_{i_l} t_{i_l}} + \frac{\beta}{\sum_{q} t_q} + \frac{\gamma}{\sum_{k} t_k} \right) \tag{3}$$

All tasks in the queue are represented by q, all tasks in the backlog by k, and all scheduled tasks for node l are represented by i_l, as well as time for tasks represented by t. The negative reward is intended to indicate that reduced delays should lead to increased rewards (less negative).

It is important to observe that at the initial scheduling framework, α_i, β, and γ are all set to 1. The cumulative reward over time is exactly equal to the negative sum of task delays, reducing the average task delay. In this approach, to prioritise the completion of tasks at the earliest opportunity, a higher penalty is imposed on tasks in the queue by assigning a slightly larger value to $\beta = 2$. It also uses several α_i to represent the distinct transmission speeds from the task queue to different nodes.

B. Deep Q Network-based Task Scheduling Model

Within this framework, DQN uses a multi-layer convolutional neural network (CNN) to approximate Q-values and support decision-making. The DQN agent's primary role is to interact dynamically with an environment, which is conceptualised as a series of states. As mentioned earlier, the environment state is formalised as binary matrices to simplify data handling and processing by the DQN agent. This formulation allows for organised and straightforward input to the CNN model, aiding in learning patterns, making decisions, and optimising the DQN agent's behaviour. In addition, the integration of CNN and DQN into the task scheduling model aims to optimise decision-making by learning and selecting actions that yield the most beneficial rewards in a given state.

Regarding local training and scheduling on the cluster with heterogeneous nodes, the tasks are sent to a queue with a fixed length to the cluster for scheduling on nodes. The state of tasks and nodes is then sent to the DQN agent, which uses the policy gradient method to learn and schedule tasks on the nodes. After training the DQN agent on the cluster environment through several episodes, the agent schedules tasks on the nodes based on its experience. Fig. 3 illustrates such a DQN agent's decision-making process.

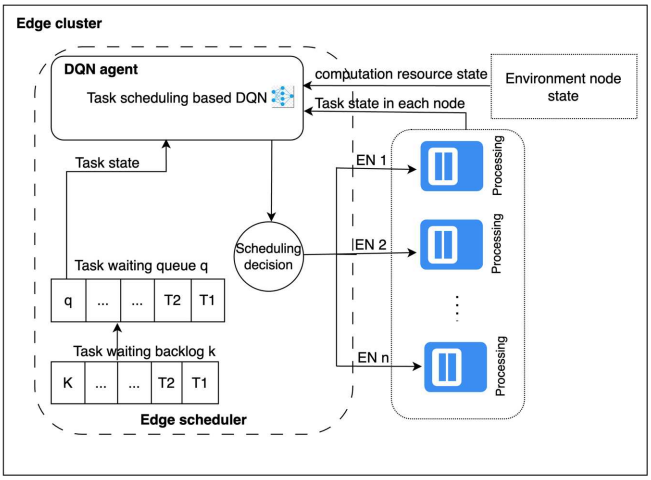

Fig. 3: DQN-based task scheduling model.

Algorithm 1 demonstrates the deployment of DQN-based task scheduling on the cluster.

The ϵ-greedy approach [2] entails that the agent has a probability of ϵ to randomly explore the action space. Otherwise, it will select the action with the greatest current Q-value to find a balance between exploration and exploitation. The experience buffer stores a collection of state transition information (s, s', a, r), which is randomly taken in small batches to reduce the correlation between samples. When the experience buffer reaches its maximum capacity and new experiences are added, older experiences will be systematically added and removed. In

Algorithm 1 DQN-based task scheduling algorithm

Input: cluster state (task requirements and node specifications)
Output: scheduling tasks to nodes
Initialisation:
  Cluster state
  DQN parameters and CNN model
  Experience buffer B with capacity D
while true **do**
  for episode $< E$ **do**
    Initialise M tasks considering the length of queue (q) and backlog (k)
    Select an action (a) with ϵ-greedy strategy
    Perform action (a), obtain next state (s'), reward (r), task delay (s_i)
    Store transition (s,a,r,s_i,s') in B
    Accumulate r += (r), and s_i += -(r+)
    if amount of samples in $B > D$ **then**
      Update experience buffer (s,a,r,s_i,s')
      Perform gradient descent on the loss function
      Update DQN parameters
    end if
    Update the cluster state (s) to ($s+1$)
  end for
end while

addition, gradient descent is an optimisation process employed to minimise a loss function by iteratively modifying the parameters of the agent. It is a fundamental technique in the ML approach and is extensively employed for training neural network models [3].

C. Federated Learning-based Task Scheduling Model

The scheduling framework integrates the FL technique at this level by training and testing multiple DQN agents in different environments. In this approach, task scheduling occurs after training multiple DQN agents in multiple instances of the environment, all of which have identical specifications but are in different states. Consequently, the FL-based task scheduling is determined by acquiring knowledge policies and experiences from different environments to inform task scheduling decisions, resulting in a scheduling framework that is more adaptable and resilient. The process of the FL technique can be succinctly summarised in the subsequent steps:

1) During the setup phase, the FL server generates an initial DQN agent and distributes it to each environment instance. These instances can later access the global agent.
2) Every instance trains an individual DQN agent based on its unique states.
3) The FL server receives updates on agent parameters.
4) The FL server aggregates the agent parameter updates using the FedAvg model.
5) The integrated model is distributed to all agents in order to build a global agent.

The process iterates until the model converges or reaches the maximum number of iterations. Finally, the global model applied task scheduling in global test environment, as illustrated in Fig. 4.

Algorithm 2 presents the FL technique for task scheduling, which involves collaborating with multiple DQN agents in different cluster instances to improve task scheduling decisions.

IV. Experimental evaluation

This section evaluates the suggested scheduling framework by running simulations to show how well it works with different

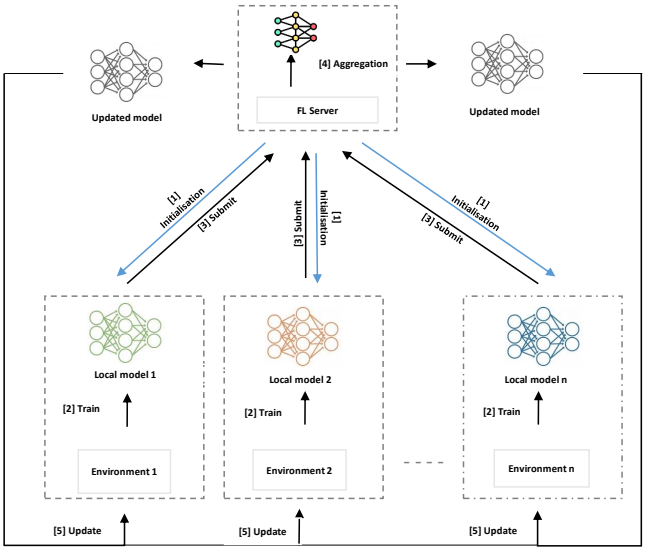

Fig. 4: FL training process across multiple environments

Algorithm 2 FL-based task scheduling algorithm

Input: $Agents$ & $Envs$
Output: $Global\ agent$ - global scheduling
Initialisation:
Initialise N cluster instances ($Envs$) & $global\ agent$
Set total rounds R
for rounds in 0 to $R-1$ **do**
  Initialise $agents$ with their $Envs$
  for each $agent\ i$ in $agents$ **do**
    Train $agent\ i$ in its Env
    Test $agent\ i$ in its $test_Env$
    Add $score$ to $scores$ list
  end for
  Aggregate $agent\ rewards$ **to update** $global\ agent$
  Distribute updated parameters to all $agents$
  Update each $agent$ **with** $global\ agent\ parameters$ **for next** R
end for
Calculate $Avg\ Scores$

tasks in different EC environments. Initially, the first level of scheduling in the framework is inspired by the standard DQN-based task scheduling [4]. The framework is developed to accept multiple heterogeneous EC environments, represented by several cluster instances with different state transitions. The FedAvg model is then employed to apply the FL technique to the framework at the next scheduling level. The proposed scheduling framework undergoes training and testing using synthetic data, including task pattern generation as a workload. The components of the scheduling framework are implemented using Python (version 3.11) and TensorFlow (version 2.14.1). The framework depends on several essential libraries, including NumPy (version 1.26.4) for numerical computation, Pillow (version 10.2.0) for visual representation, and Matplotlib (version 3.8.3) for results visualization. This framework is deployed on the University of Leeds high-performance computing infrastructure (ARC3/4) [14]. ARC3/4 provides the computational resources necessary to perform large-scale task-scheduling experiments. The machine configuration used in the experiments is a single V100 card, 10 CPU cores, and 48GB of system

memory with 170GB of storage.

The main goal of this evaluation is to demonstrate that the FL-based task scheduling model improves the agent learning curve, reduces task delay over time, and increases scalability over FL rounds compared to the DQN scheduling model. In particular, this evaluation aims to demonstrate that the global agent, trained using the FedAvg model, outperforms a single agent in terms of cumulative rewards, the average of task delay, scalability, and achieves superior average scores in different scenarios.

Additionally, this section provides an implementation setup including the key configurations of the task description, simulation environment parameters, and ML models' parameters in the scheduling framework. Following this, a sub-section presents and discusses the simulation results.

A. Implementation Setup

The present scheduling framework is trained and evaluated using synthetic data as a workload that includes three distinct task types, each with varying resource demands and estimation time. Small tasks require a small size of three different resources (CPU cores, memory (GB), and disc (GB)) to be completed within a short time (e.g., a task requires 1 CPU core, 32 GB of memory, and 10 GB of storage to be completed within 4 seconds as estimation time). Small tasks with skewed resources necessitate three distinct resources of small sizes. However, some tasks exhibit imbalanced resource requirements, such as demanding large-size resources or a lengthy execution time (e.g., a task requires 1 CPU core, 128 GB of memory, and 50 GB of storage to be completed within 7 seconds as estimation time). This task type has three different batches, each considering varying resource allocation needs. Large tasks come with large-size resource demands and consume considerable time (e.g., a task requires 1 CPU core, 256 GB of memory, and 1000 GB of storage to be completed within 25 seconds as estimation time). In this way, the categorisation of tasks allows the scheduling framework to properly handle a wide variety of resource allocation scenarios, ensuring enhanced performance and efficient use of system resources.

In the scheduling framework, the task generator is responsible for generating the workload by creating 500 batch sizes for each task type, which are then transmitted online to a queue of fixed length $q = 15$ and a backlog of fixed length $k = 10$. The workload comprises 2,500 tasks with random arrival times following a Poisson distribution, three resource requirements of varying sizes, and a completion time. In addition, this framework represents heterogeneous cluster environments with different node specifications. Table II summarises the related parameters to the simulation environment, including the workload description. All the hyper-parameters are selected through several trials that produce reasonable results for each cluster instance in the framework.

Table III illustrates the CNN structure as a component of the DQN-based scheduling model.

Table IV shows the hyper-parameters selected for the DQN-based scheduling model after ten tests were run to find the appropriate hyper-parameters for the scheduling framework. The objective was to optimise the agent learning curve in order to enhance the scheduling decision as an inference model.

TABLE II: Environment simulation parameters.

Simulation Parameter	**Value**
Number of task resource requirements	3
Task duration average	(1 to 20) Seconds
Task type	3
Workload size	2,500 Heterogeneous tasks
Node type	Heterogeneous
Node active time	(10 to 25) Seconds
Number of node resources	3
Number of cluster nodes	1,2,4,6,8
Queue length	15
Backlog length	10

TABLE III: CNN model structure.

Layer	**Parameters**	**Activation Function**
Input Layer	*input_shape*	-
Convolutional	16 filters, (3, 3) kernel size	ReLU
Pooling	Default pooling size	-
Dropout	0.2 (drop rate)	-
Flatten	-	-
Dense	256 units	ReLU
Output Layer	*output_shape*	Linear

TABLE IV: DQN model hyper-parameters.

Parameters	**Description**	**Value**
episode	Number of iterations	300
σ	Learning rate	0.01
D	Max. of experience replay buffer	10,000
γ	Factor discounting future rewards	0.99
Update steps	Copy and save parameters update	32
ϵ	Agent exploration probability	0.99

In order to integrate the FL technique into the scheduling framework, the technique is first implemented by training and testing the DQN agents across various environment instances over a series of FL rounds, as explained in the previous section. Task scheduling is implemented in accordance with the updated DQN agent to evaluate the agents' improved learning, and average scores are then listed as a result of testing the global agent. The hyper-parameters employed in the FL technique are shown in Table V.

TABLE V: FL model hyper-parameters.

Parameters	**Value**
FL rounds	5
FL technique	FedAvg model
Frequency update	32
Number of agents	3
Number of cluster instances	3
Number of scheduling tests	3
Test environment for scheduling	3
Global agent (aggregated 3 agents)	1
Single agent	1

B. Results and Discussion

This subsection presents the experimental findings obtained by implementing the proposed task scheduling framework, which is based on the FL technique. The structure of the results provides insights into the relative effectiveness of global agent training and testing versus single-agent training and testing in various scheduling scenarios within the cluster environments. Regarding the training phase comparison, Fig. 5 depicts the total

rewards obtained by a single agent compared to a global agent during a sequence of 300 episodes. The upward trajectory of each agent's respective curve indicates a general improvement in performance. At first, the total rewards for both agents are comparable, but as the episodes advance, the disparity between the two agents increases. At the end of the 300 episodes, the global agent's total rewards are far greater than those earned by the single agent. The global agent's curve indicates a steeper incline, suggesting a more rapid pace of learning and adjustment to the cluster environment. Conversely, the single agent shows a gradual rise in cumulative rewards, indicating slower learning and potentially less effective adaptation. To summarise, the global agent outperforms the single agent in terms of cumulative rewards throughout the training period, indicating the potential benefits of employing a global learning strategy in task scheduling scenarios in the cluster environment.

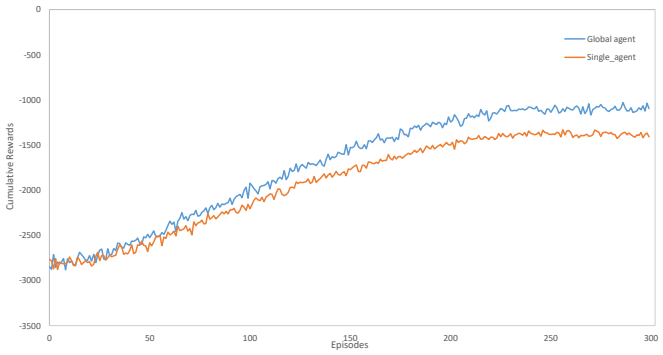

Fig. 5: Cumulative rewards in training phase

In order to evaluate the proposed task scheduling framework in the testing phase, the first scenario includes eight experiments to compare task scheduling by using a single agent and a global agent (aggregated of three agents) based on the number of nodes in the environment. Fig. 6 demonstrates that as the number of nodes increases, the average task delay of both the global agent and the single agent decreases. Nevertheless, at all node levels, the global agent always shows a lower average task delay than the single agent. The single agent delay is approximately (-0.3), while the global agent delay is about (-0.2) with a single node, which represents a 33.33% improvement for the global agent. The global agent delay further decreases to approximately (-0.6) as the number of nodes reaches eight, while the single agent's delay lowers to just below (-0.4), indicative of a 50% improvement.

The global agent shows the most notable improvement when transitioning from one to two nodes. As a result, the improvement gradually increases across all levels of nodes. Overall, the global agent outperforms the single agent by more efficiently managing tasks across multiple nodes and attaining lower task delays. The reason for this is that the global agent acquires knowledge from a variety of policies during training and testing on many rounds in different environments with varying states.

In the second scenario, five experiments are conducted to test

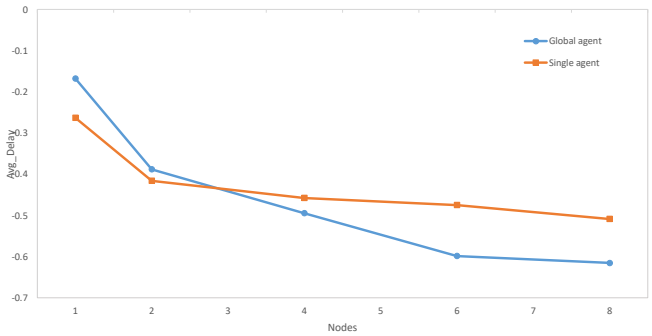

Fig. 6: Average task delay based on node number

three agents trained in different cluster instances. The goal is to obtain a global agent within five FL rounds. Fig. 7 shows the performance disparity between the single agent and the global agent across multiple FL rounds. At first, the single agent shows a significant decline in performance, reaching its lowest point by the second round. In contrast, the global agent maintains a more consistent trajectory, albeit with a consistently lower score. The single agent undergoes substantial fluctuations as the rounds progress, experiencing a significant peak and trough in the third and fourth rounds, respectively. While the global agent displays a consistent rate of improvement. In comparison to the global agent, which continues its steady ascent, the single agent presents a minor recovery by the fifth round but remains more volatile.

This comparison highlights the global agent's stability and improvement in contrast to the single agent's oscillating performance. Global agent stability is ascribed to the fact that its parameters are updated after each round, thus incorporating learning from the various conditions of the three agents. This method facilitates task scheduling during testing and elevates the average scores in each round. In contrast, the single agent operates individually in each round, with task scheduling contingent upon the trained model from that particular round.

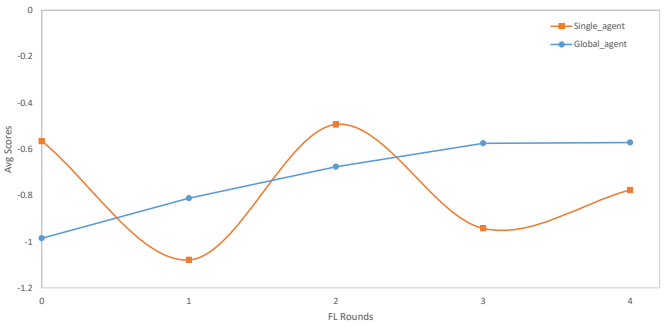

Fig. 7: Average scores across multiple FL rounds.

V. Conclusion

This paper has addressed the problem of reducing heterogeneous task delay in a heterogeneous environment, specifically in the EC environment. A task scheduling framework is proposed

to reduce task delay on clusters with multiple nodes. This scheduling framework is based on the FL technique, which involves the aggregation of multiple DQN agents that have been trained in different environments periodically. Simulation experiments prove that the FL-based task scheduling framework is suitable for delay-sensitive tasks. The results show that the global agent (FL-based scheduler) outperforms a single DQN agent (DQN-based scheduler) in terms of improving the agent learning curve, reducing task delay over time, and increasing scalability. The proposed framework has been trained and tested on synthetic data, with a focus on scheduling at the node level. Future work will evaluate the proposed scheduling framework using real-world data. In addition, the proposed framework will incorporate feedback closed-loop learning, which is an iterative process in which insights acquired at one level (node or cluster) can reinforce learning and improve performance at the other level.

References

[1] AR Arunarani, Dhanabalachandran Manjula, and Vijayan Sugumaran. 2019. Task scheduling techniques in cloud computing: A literature survey. *Future Generation Computer Systems* 91 (2019), 407–415.

[2] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47 (2002), 235–256.

[3] Pierre Baldi. 1995. Gradient descent learning algorithm overview: A general dynamical systems perspective. *IEEE Transactions on neural networks* 6, 1 (1995), 182–195.

[4] Weijia Chen, Yuedong Xu, and Xiaofeng Wu. 2017. Deep reinforcement learning for multi-resource multi-machine job scheduling. *arXiv preprint arXiv:1711.07440* (2017).

[5] Pegah Gazori, Dadmehr Rahbari, and Mohsen Nickray. 2020. Saving time and cost on the scheduling of fog-based IoT applications using deep reinforcement learning approach. *Future Generation Computer Systems* 110 (2020), 1098–1115.

[6] Zhiming Hu, James Tu, and Baochun Li. 2019. Spear: Optimized dependency-aware task scheduling with deep reinforcement learning. In *2019 IEEE 39th international conference on distributed computing systems (ICDCS)*. IEEE, 2037–2046.

[7] Hao Jin, Yang Peng, Wenhao Yang, Shusen Wang, and Zhihua Zhang. 2022. Federated reinforcement learning with environment heterogeneity. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 18–37.

[8] Nouha Kherraf, Hyame Assem Alameddine, Sanaa Sharafeddine, Chadi M Assi, and Ali Ghrayeb. 2019. Optimized provisioning of edge computing resources with heterogeneous workload in IoT networks. *IEEE Transactions on Network and Service Management* 16, 2 (2019), 459–474.

[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 *Artificial intelligence and statistics*. PMLR, 1273–1282.

[10] Hao Meng, Daichong Chao, Ru Huo, Qianying Guo, Xiaowei Li, and Tao Huang. 2019. Deep reinforcement learning based delay-sensitive task scheduling and resource management algorithm for multi-user mobile-edge computing systems. In *Proceedings of the 2019 4th International Conference on Mathematics and Artificial Intelligence*. 66–70.

[11] Martin L Puterman. 1990. Markov decision processes. *Handbooks in operations research and management science* 2 (1990), 331–434.

[12] Tianyi Shi, Hongfeng Tian, Tiankui Zhang, Jonathan Loo, Jiangtao Ou, Chengyuan Fan, and Dingcheng Yang. 2022. Task Scheduling with Collaborative Computing of MEC System Based on Federated Learning. In *2022 IEEE 95th Vehicular Technology Conference:(VTC2022-Spring)*. IEEE, 1–5.

[13] Shreshth Tuli, Shashikant Ilager, Kotagiri Ramamohanarao, and Rajkumar Buyya. 2020. Dynamic scheduling for stochastic edge-cloud computing environments using a3c learning and residual recurrent neural networks. *IEEE transactions on mobile computing* 21, 3 (2020), 940–954.

[14] University of Leeds Research Computing Team. 2024. ARC Documentation - Welcome. https://arcdocs.leeds.ac.uk/welcome.html Accessed: 2024-09-10.

[15] Lun Wang, Yang Xu, Hongli Xu, Min Chen, and Liusheng Huang. 2022. Accelerating decentralized federated learning in heterogeneous edge computing. *IEEE Transactions on Mobile Computing* 22, 9 (2022), 5001–5016.

[16] Xiaojie Wang, Shupeng Wang, Yongjian Wang, Zhaolong Ning, and Lei Guo. 2021. Distributed Task Scheduling for Wireless Powered Mobile Edge Computing: A Federated-Learning-Enabled Framework. *IEEE Network* 35, 6 (2021), 27–33.

[17] Jie Wen, Zhixia Zhang, Yang Lan, Zhihua Cui, Jianghui Cai, and Wensheng Zhang. 2023. A survey on federated learning: challenges and applications. *International Journal of Machine Learning and Cybernetics* 14, 2 (2023), 513–535.

[18] Tiago CS Xavier, Flavia C Delicato, Paulo F Pires, Claudio L Amorim, Wei Li, and Albert Zomaya. 2022. Managing Heterogeneous and Time-Sensitive IoT Applications through Collaborative and Energy-Aware Resource Allocation. *ACM Transactions on Internet of Things* 3, 2 (2022), 1–28.

[19] Wenchao Xia, Wanli Wen, Kai-Kit Wong, Tony QS Quek, Jun Zhang, and Hongbo Zhu. 2021. Federated-learning-based client scheduling for low-latency wireless communications. *IEEE Wireless Communications* 28, 2 (2021), 32–38.

[20] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)* 10, 2 (2019), 1–19.