Dynamic Optimal Coding and Scheduling for Distributed Learning over Wireless Edge Networks

Nguyen Van Huynh, Dinh Thai Hoang, Diep N. Nguyen, and Eryk Dutkiewicz
School of Electrical and Data Engineering, University of Technology Sydney, Australia

***Abstract*—This paper proposes a novel framework that can effectively address key challenges for the development of distributed learning over wireless edge networks. In particular, we first introduce a highly effective distributed learning model leveraging the most recent advanced coded distributed computing algorithm together with collaborative computing resources from wireless edge nodes to securely and effectively execute learning tasks. To minimize the average delay of learning tasks, the coding and scheduling policies must be jointly optimized. However, determining the optimal coding scheme together with the optimal edge nodes for different learning tasks is NP-hard due to the dynamics and uncertainty of the wireless environment and straggling problems at the computing nodes. Thus, we develop a highly effective approach utilizing advances of both reinforcement learning algorithms and the dueling network architecture to quickly find the optimal coding scheme together with the best edge nodes for different learning tasks without requiring completed information about the surrounding environment and straggling parameters in advance. Through extensive simulation results, we show that our proposed framework can reduce the average delay for the whole system up to 66% compared with other conventional learning and optimization approaches.**

***Index Terms*—Coded computing, wireless edge networks, distributed learning, and deep reinforcement learning.**

I. Introduction

Recently, the coded computing technique has been emerging as a prominent solution to deal with straggling problems in distributed learning over wireless edge networks [1], [2]. In particular, the coded computing technique adds data/computation redundancy to learning tasks before offloading them to edge nodes for processing. In this way, this technique does not require all edge nodes to send back their computed results. Instead, only computed results from a number of edge nodes are required to decode the final result. In other words, the computation latency is determined by a best set of edge nodes [1]. As such, the coded computing technique can significantly mitigate the straggling problem at edge nodes. Moreover, the communication delay can be also reduced as the coded computing technique can mitigate straggling problems caused by unstable wireless links. Finally, with the coded computing technique, learning tasks are encoded (with data/computation redundancy) before offloading to edge nodes, thereby greatly increasing the data privacy of the system.

The coded computing technique has been widely adopted in distributed learning systems recently [1]–[5]. The authors in [1] introduce a novel maximum distance separable (MDS) code for matrix multiplication and data shuffling which are the most common tasks in machine learning. By adding data redundancy, the MDS code can mitigate the effect of stragglers and communication bottlenecks. Similarly, in [3], the authors propose to encode datasets with built-in data redundancy to mitigate the straggling problem in linear regression tasks. However, these works ignore the effects of wireless communications which can lead to serious degradation in the system performance [4]. For that, in [5], the authors study both wireless and computing impairments when designing coding mechanism to jointly minimize the computing and communication delay. To do that, a group of edge nodes serving a particular learning task is determined by considering imperfect channel state information, straggling processors, and interference. Although achieving good performance, these works and others in the literature require complete environment knowledge in advance, which may not be feasible in practice. Specifically, straggling problems at both edge nodes and wireless links are uncertain due to several unpredictable factors such as random hardware errors, maintenance activities, interference from surrounding devices, and random obstacles on wireless links. Without taking these factors into account, existing solutions may not be able to obtain good training time for distributed learning over wireless edge networks. Moreover, current works usually ignore the heterogeneity of edge nodes and wireless links when optimizing coding mechanisms, and thus limiting the performance of the system.

To address the aforementioned problems, this paper proposes a jointly optimal coding and scheduling framework that can intelligently obtain the optimal code as well as the best set of edge nodes to process each learning task, given the current state of the whole system. Specifically, we first develop a Markov decision process (MDP) framework to account for the dynamics and uncertainty of the system such as wireless channel states, straggling problems at different edge nodes, and diverse learning tasks and computing resources. To obtain the optimal coding (i.e., optimal value of n and k) and scheduling (i.e., best edge nodes to serve a particular learning task) policy under the proposed MDP framework, Q-learning algorithm can be adopted. Nevertheless, conventional Q-learning algorithms usually require very long learning time to obtain the optimal policy, especially with high-dimensional state and action spaces considered in this paper. To tackle this, we propose a highly effective deep reinforcement learning algorithm, called deep dueling, utilizing the advanced deep dueling neural network architecture [6] to significantly improve the learning process of the system. Extensive simulation results demonstrate that our proposed solution can reduce the average

latency for learning tasks up to 66% by jointly obtaining the optimal coding and scheduling policy for each learning task.

II. System Model

In this work, we consider a distributed learning over wireless edge network that includes a mobile edge computing (MEC) server and N edge nodes denoted by $\mathbf{E} = \{E_1, \ldots, E_j, \ldots, E_N\}$. Edge node E_j connects with the MEC server through wireless link C_j as illustrated in Fig. 1. The MEC server is equipped with a task queue to store learning tasks arriving at the system. The maximum size of the task queue is defined by M. Without loss of generality, we assume that time is slotted. In each time slot, a learning task arrives at the task queue with probability μ. We denote $\mathcal{D}^{(t)}$ as the learning task arrives at the system at time slot t. The data size of $\mathcal{D}^{(t)}$ is denoted by $f(\mathcal{D}^{(t)})$. We assume that learning tasks in the task queue are served in a first-come-first-served manner. In particular, a learning task in the task queue is considered to serve if the computing resources at edge nodes are available and this learning task comes earliest in the queue but not yet served by any edge nodes (e.g., $\mathcal{D}^{(2)}$ as illustrated in Fig. 1). After being served, a learning task still remains in the task queue until the MEC server successfully decodes computed results from assigned edge nodes to derive its final result. We

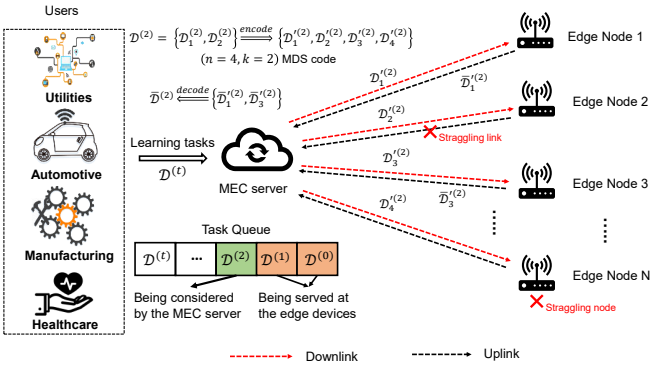


Fig. 1: System model for coded distributed learning over wireless edge network. Here, we illustrate the case when learning task $\mathcal{D}^{(2)}$ is processed with $(n = 4, k = 2)$ MDS code. The sub-learning tasks are sent to edge nodes $1, 2, 3,$ and N to process. Then, when edge node 2 is disconnected and edge node N is straggling, learning task $\mathcal{D}^{(2)}$ still can be completed by using computed results from edge nodes 1 and 3.

assume that each edge node serves a single learning task at a time. The reason is that in wireless edge networks, edge nodes usually have limited resources such as IoT gateways and mobile phones. As such, they may not be able to process multiple learning tasks at the same time while ensuring good computation latency. We denote e_j as the status of edge node E_j. In particular, $e_j = 1$ if there is no learning task executing at edge node E_j. $e_j = 0$ if edge node E_j is currently serving a learning task. We then denote $\mathbf{E}_{\text{av}} \overset{\text{def}}{=} \{E_j : \forall E_j \in \mathbf{E}$ and $e_j = 1\}$ as the set of available edge nodes.

A. Coded Computing for Distributed Learning over Wireless Edge Networks

In this work, we adopt the maximum distance separable (MDS) code [1] to encode learning tasks. In particular, with (n, k) MDS code $(1 \leq k \leq n)$, the MEC server first divides a learning task $\mathcal{D}^{(t)}$ into k equal-sized sub-learning tasks denoted by $\{\mathcal{D}_1^{(t)}, \mathcal{D}_2^{(t)}, \ldots, \mathcal{D}_k^{(t)}\}$. After that, these sub-learning tasks are encoded to n encoded sub-learning tasks denoted by $\{\mathcal{D}_1^{\prime(t)}, \mathcal{D}_2^{\prime(t)}, \ldots, \mathcal{D}_n^{\prime(t)}\}$. The MEC server then offloads these encoded sub-learning tasks to n edge nodes for processing. As soon as receiving k computed results from k edge nodes, the MEC server can decode them to obtain the final computed result for learning task $\mathcal{D}^{(t)}$. Finally, learning task $\mathcal{D}^{(t)}$ is removed from the task queue and the MEC server notifies all edge nodes to stop processing their assigned sub-learning tasks for $\mathcal{D}^{(t)}$.

B. Communication and Computation Models

Assuming that sub-learning task $\mathcal{D}_i^{\prime(t)}$ $(1 \leq i \leq n)$ is offloaded to edge node $E_j \in \mathbf{E}$ to execute, the total serving time of this sub-learning task can be formulated as follows:

$$T_{\text{serve}}^{(t,i)} = T_{\text{se}}^{(t,i)} + T_{\text{cmp}}^{(t,i)} + T_{\text{es}}^{(t,i)}, \tag{1}$$

where $T_{\text{serve}}^{(t,i)}$ is the total serving time of sub-learning task $\mathcal{D}_i^{\prime(t)}$, $T_{\text{se}}^{(t,i)}$ is the communication time for sending $\mathcal{D}_i^{\prime(t)}$ from the MEC server to edge node E_j through wireless link C_j. $T_{\text{es}}^{(t,i)}$ is the communication time for sending computed result from edge node E_j to the MEC server through wireless link C_j. $T_{\text{cmp}}^{(t,i)}$ is the computation time that edge node E_j needs to finish processing sub-learning task $\mathcal{D}_i^{\prime(t)}$. We assume that each sub-learning task and its computed result can be transmitted within one time slot as the connection from edge node to the server is usually a high-speed connection (e.g., via mmWave). p_j is defined as the disconnection probability of wireless link C_j. Then, we denote $\mathbf{p} = \{p_1, \ldots, p_j, \ldots, p_N\}$ as the set of disconnection probabilities of wireless links $\{C_1, \ldots, C_j, \ldots, C_N\}$.

At each time slot, if wireless link C_j is disconnected, transmitted data needs to be resent in the next time slot. Thus, we can formulate the communication time of the MEC server and edge node E_j through link C_j as follows:

$$T_{\text{se}}^{(t,i)} = T_{\text{es}}^{(t,i)} = H_j \xi, \tag{2}$$

where ξ denotes the duration of a time slot and H_j presents the number of time slots required to successfully transmit data over link C_j. H_j follows the *Geometric* distribution and identically and independently distributed with successful probability $p_{\text{success}} = 1 - p_j$. We then can formulate the probability function of H_j as follows [2]:

$$Pr(H_j = x) = p_j^{x-1}(1 - p_j), x = 1, 2, 3, \ldots \tag{3}$$

From (3), we can observe that with a high disconnection probability, the number of time slots required to successfully transmit data over link C_j (i.e., H_j) is also high.

It is clear that the computation time of sub-learning task $\mathcal{D}_i^{'(t)}$ $(1 \leq i \leq n)$ at edge node E_j is the sum of the deterministic time for processing data and the stochastic time that depends on unpredictable factors at the edge node. Denote $T_{\text{cmp}}^{(t,i)}$ as the total time that edge node E_j requires to process sub-learning task $\mathcal{D}_i^{'(t)}$, we have

$$T_{\text{cmp}}^{(t,i)} = \underbrace{g(\lambda_j)}_{\text{stochastic time}} + \underbrace{\eta_j f(\mathcal{D}_i^{'(t)})}_{\text{deterministic time}}, \tag{4}$$

where η_j is the processing power of edge node E_j. $g(\lambda_j)$ denotes the stochastic time caused by the straggling problem at the edge node, following an exponential distribution with rate λ_j [2], [7], i.e., $p_{g(\lambda_j)}(x) = \lambda_j e^{-\lambda_j x}, x \geq 0$. The set of rate parameters at edge nodes is then denoted as $\boldsymbol{\lambda} = \{\lambda_1, \ldots, \lambda_j, \ldots, \lambda_N\}$.

Substituting (2) and (4) into (1), the total serving time of $\mathcal{D}_i^{'(t)}$ at an edge node can be written as follows:

$$\begin{aligned} T_{\text{serve}}^{(t,i)} =& \Big(2H_j\xi + g(\lambda_j) + \eta_j|\mathcal{D}_i^{'(t)}|\Big)c_{i,j}, \\ & \forall i \in \{1, \ldots, n\}, \forall j \in \{1, 2, \ldots, N\} \end{aligned} \tag{5}$$

where $c_{i,j}$ is a scheduling binary decision. $c_{i,j} = 1$ if $\mathcal{D}_i^{'(t)}$ is served at edge node E_j, and $c_{i,j} = 0$, otherwise. Recall that each sub-learning task is only processed at one edge node, thus we have $\sum_{j=1}^{N} c_{i,j} = 1, \forall i \in \{1, \ldots, n\}$.

With (n, k) MDS code, the MEC server needs k computed results from k edge nodes to decode the final result. As such, the serving time of learning task $\mathcal{D}^{(t)}$ can be determined by the serving time of k-th completed sub-learning task. Denote $T_{\text{serve}}^{(t)}$ as the serving time of learning task $\mathcal{D}^{(t)}$, we have

$$T_{\text{serve}}^{(t)} = \min_{k-th}\left(\left\{T_{\text{serve}}^{(t,i)} : \forall i \in \{1, 2, \ldots, n\}\right\}\right), \tag{6}$$

where $\min_{k-th}(.)$ returns the k-th minimum value of a set. For example, $\min_{3-th}\Big(\{1, 5, 10, 4, 6\}\Big) = 5$.

C. Serving Time Minimization Problem

Given the above, the serving time minimization problem for each learning task can be formulated as follows:

$$\begin{aligned} &\min_{n,k,\{c_{i,j}\}} T_{\text{serve}}^{(t)}, \\ &\text{s.t.} \quad 1 \leq k \leq n, \forall n \in \{1, 2, \ldots, |\mathbf{E}_{\text{av}}|\}, \\ &\quad c_{i,j} \in \{0, 1\}, \forall i \in \{1, 2, \ldots, n\} \text{and } \forall j \in \{1, 2, \ldots, N\}, \\ &\quad \sum_{j=1}^{N} c_{i,j} = 1, \forall i \in \{1, \ldots, n\} \text{ and } \forall j \in \{1, \ldots, N\}, \\ &\quad c_{i,j} = 1, \text{ if } e_j = 1, \forall j \in \{1, \ldots, N\}. \end{aligned} \tag{7}$$

THEOREM 1. *The joint coding and scheduling optimization problem (7) is NP-hard.*

Proof. The proof can be found in [9]. □

In Theorem 1, we show that minimizing the serving time for each learning task is NP-hard, even if the environment factors such as p_j, λ_j, and η_j are available in advance. Nevertheless, these factors may not be available in advance in practice. The reason is that the straggling problems at both wireless links and edge nodes as well as arriving learning tasks are unpredictable. In addition, in this paper, we aim to minimize the average delay for all learning tasks, which is intractable for current optimization tools [1], [2]. To address these issues, we develop a Markov decision process framework to account for the dynamics and uncertainty of the system. Then, a highly effective deep reinforcement learning algorithm is proposed to learn all the environment factors and obtain the jointly optimal coding and scheduling policy for the system.

III. Coded Computing for Distributed Learning Formulation

To capture the dynamics and uncertainty of the considered system, we adopt the Markov decision process (MDP) framework to reformulate the system delay minimization problem in (7). Specifically, the MDP consists of a state space $\mathcal{S}$, an action space $\mathcal{A}$, and an immediate reward function r.

A. State Space

As discussed, the MEC server serves learning tasks in the task queue in a first-come-first-served manner. Therefore, the queue size, the data size of the considered learning task, and the available edge nodes in the system are critical parameters that should be taken into account in the system state. For that, the state space can be defined as follows:

$$\begin{aligned} \mathcal{S} \triangleq \Big\{&(m, f, \{e_1, \ldots, e_j, \ldots, e_N\}) : m \in \{0, \ldots, M\}; \\ &f \geq 0; e_j \in \{0, 1\}, \forall j \in \{1, \ldots, N\}\Big\}, \end{aligned} \tag{8}$$

where m is the queue size (i.e., number of learning tasks currently waiting in the queue), f presents the data size of the considered learning task, and e_j is the state of edge node E_j. It is worth noting that f equals 0 if the task queue is empty or all learning tasks in the queue are being served by the system.

B. Action Space

In this work, we aim to find not only the optimal code but also the best set of edge nodes to serve learning tasks based on the current system state s. Thus, the action space $\mathcal{A}$ can be formulated as follows:

$$\begin{aligned} \mathcal{A} \triangleq \{a_s\} =& \{(0, 0, \emptyset), (n, k, \mathbf{E}_{\text{b}})\}, \forall n \in \{1, \ldots, |\mathbf{E}_{\text{av}}|\}, \\ & \forall k \in \{1, \ldots, n\}, \forall \mathbf{E}_{\text{b}} \in \binom{\mathbf{E}_{\text{av}}}{n}, \end{aligned} \tag{9}$$

where a_s is the action taken at state s and $\mathbf{E}_{\text{b}}$ presents the best set of edge nodes to process the considered learning task, and $\binom{\mathbf{E}_{\text{av}}}{n}$ returns all size-n subsets of $\mathbf{E}_{\text{av}}$. Given the above, $a_s = (n, k, \mathbf{E}_{\text{b}})$ when the MEC server uses (n, k) MDS code to encode the considered learning task and edge nodes in $\mathbf{E}_{\text{b}}$ to serve the encoded sub-learning tasks at state s. If the MEC server stays idle, $a_s = (0, 0, \emptyset)$ (i.e., not select any code and edge nodes to execute the considered task or the task queue is empty).

C. Immediate Reward

The aim of this research is to minimize the average long-term delay of learning tasks. As mentioned, after serving by the MEC server, a learning task still remains in the queue until all necessary computed results are sent back from assigned edge nodes to decode the final result for this learning task. Hence, the average delay of a learning task is determined from the time it enters the system until the MEC server decodes its result successfully. However, due to random straggling problems in both the edge nodes and wireless links, the computation time and communication time of a learning task cannot be calculated correctly in advance. Consequently, after taking action a_t at state s_t to serve a learning task, the MEC server cannot know when this learning task is completed to obtain immediate reward r_t. To address this problem, r_t can be determined by the number of learning tasks currently waiting in the queue. This is because the size of the task queue can implicitly capture the delay of all learning tasks according to the Little theorem. Thus, the immediate reward function is expressed as follows:

$$r_t(s_t, a_t) = -m, \tag{10}$$

where $m \in \{0, \ldots, M\}$ represents the instantaneous size of the queue after performing action a_t at state s_t.

D. Long-Term Delay Minimization Formulation

We aim to obtain the optimal coding and scheduling policy which is a mapping from a state s_t to the optimal action a_t to maximize the long-term average reward. In this way, the average number of learning tasks waiting in the queue is minimized, resulting in a minimal average delay for the system. Therefore, the optimization problem can be formulated as follows:

$$\max_{\pi} \qquad \mathcal{R}(\pi) = \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}\left(r_t(s_t, \pi(s_t))\right), \tag{11}$$

where $\mathcal{R}(\pi)$ is the average long-term reward of the system under policy π and $r_t(s_t, \pi(s_t))$ is the immediate reward after performing an action given policy π at time step t.

IV. Optimal Coded Edge Computing with Deep Dueling Algorithm

To obtain the optimal coding and scheduling policy, the Q-learning algorithm can be used. However, this algorithm faces a slow-convergence problem, especially with large state and action spaces in our considered system. To tackle this issue, we develop a highly effective algorithm, namely deep dueling, utilizing both the deep reinforcement learning and the deep dueling neural network architecture [6]. The principle of this algorithm is to train the deep dueling neural network instead of using the Q-table as in the conventional Q-learning algorithm to find the optimal coding and scheduling policy. Specifically, the algorithm deploys a reply memory $\mathbf{D}$ to store transitions (s_t, a_t, r_t, s_{t+1}) during the training process. Then, in each training iteration, a number of samples from $\mathbf{D}$ are fed into the deep dueling neural network for training.

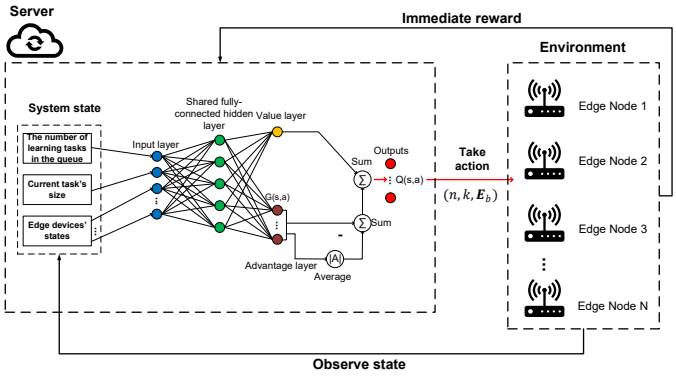


Fig. 2: Deep dueling network architecture for coded computing over wireless edge networks.

The deep dueling neural network consists of two streams of layers to separately and simultaneously estimate value and advantage functions instead of estimating the Q-value function only as that of conventional deep Q-learning algorithm as shown in Fig. 2. The reason is that several actions may have less effects on the system than others. In particular, with policy π, the value of state-action pair (s, a) is denoted as $\mathcal{Q}^{\pi}(s, a) = \mathbb{E}\left[r_t | s_t = s, a_t = a, \pi\right]$. Then, we have $\mathcal{Q}^{\pi}(s, a) = \mathcal{V}^{\pi}(s) + \mathcal{G}^{\pi}(s, a)$, in which $\mathcal{V}^{\pi}(s) = \mathbb{E}_{a\sim\pi(s)}\left[\mathcal{Q}^{\pi}(s, a)\right]$ is the value function that is used to estimate "how good it is" when the system is at state s and $\mathcal{G}^{\pi}(s, a)$ is the advantage function that represents the importance of action a compared to other actions. These two functions are separately estimated by two streams of layers in the deep dueling neural network. These two streams are then combined at the output layer by using the following function:

$$\mathcal{Q}(s, a; \boldsymbol{\alpha}, \boldsymbol{\beta}) = \mathcal{V}(s; \boldsymbol{\beta}) + \left(\mathcal{G}(s, a; \boldsymbol{\alpha}) - \frac{1}{|\mathcal{A}|} \sum_{a} \mathcal{G}(s, a; \boldsymbol{\alpha})\right). \tag{12}$$

Algorithm 1 Optimal Coding and Scheduling with Deep Dueling Neural Network Architecture

1: Construct replay memory $\mathbf{D}$ with a capacity of $\mathcal{D}$.
2: Construct the $\mathcal{Q}$ network consisting of two streams with random weights $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$.
3: Construct the target network $\hat{\mathcal{Q}}$ with weights $\boldsymbol{\alpha}^{-} = \boldsymbol{\alpha}$ and $\boldsymbol{\beta}^{-} = \boldsymbol{\beta}$.
4: **for** *iteration=1 to T* **do**
5: Performing action a_t based on ϵ-greedy policy.
6: Observe immediate reward r_t and next state s_{t+1}.
7: Add experiences (s_t, a_t, r_t, s_{t+1}) to memory $\mathbf{D}$.
8: Randomly select transitions (s_j, a_j, r_j, s_{j+1}) from $\mathbf{D}$.
9: Minimize the loss function in (13).
10: Reset $\hat{\mathcal{Q}} = \mathcal{Q}$ after every C steps.
11: **end for**

It is worth noting that, the estimated Q-value of each state-action pair may be changed during the training process. The reason is that the algorithm constantly updates the deep

neural network with new experiences. This may make the algorithm unstable as studied in [8]. To tackle this issue, we use the *quasi-static target network* method that implements a target Q-network with network parameters $(\boldsymbol{\alpha}^-, \boldsymbol{\beta}^-)$. These parameters are constantly but slowly updated with the primary Q-network parameters $(\boldsymbol{\alpha}, \boldsymbol{\beta})$. The target Q-network is used to calculate the target value y_j during the training process, i.e., $y_j = r_j + \gamma \max_{a_{j+1}} \mathcal{Q}(s_{j+1}, a_{j+1}; \boldsymbol{\alpha}^-, \boldsymbol{\beta}^-)$. Then, the loss function can be expressed as follows:

$$L_j(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \mathbb{E}_{(s_j, a_j, r_j, s_{j+1}) \sim U(\mathbf{D})} \left[\left(y_j - \mathcal{Q}(s_j, a_j; \boldsymbol{\alpha}, \boldsymbol{\beta}) \right)^2 \right], \tag{13}$$

where γ is the discount factor. By minimizing the loss function, the parameters of the deep dueling network are updated. After a number of iterations, the algorithm can converge to the optimal coding and scheduling policy. The main steps of our proposed algorithm are provided in Algorithm 1.

V. Performance Analysis and Simulation Results

A. Parameter Setting

We consider that the MEC server's task queue can store up to 10 learning tasks at a time. There are five edge nodes to execute learning tasks. Unless otherwise stated, at each time slot, a learning task arrives at the system with probability $\mu = 0.7$. For all the edge nodes, the processing time of one data point is set at five milliseconds [7]. Each learning task's size (i.e., number of data points) is generated randomly from the set of $\{100, 200, 300\}$. We set $\mathbf{p} = \{0.1, 0.5, 0.2, 0.3, 0.9\}$ and $\boldsymbol{\lambda} = \{0.1, 1, 0.5, 0.2, 2\}$. These parameters will be varied to evaluate our proposed solution in different settings. The deep neural network of the traditional deep Q-learning algorithm consists of two fully-connected hidden layers. Differently, the deep dueling neural network has two streams to estimate the value and advantage functions. These two streams are connected to a shared fully connected hidden layer as shown in Fig. 2. The hidden layer's size is set at 16. The mini-batch size is 16. The maximum size of $\mathbf{D}$ is set at $10,000$ experiences. The target Q-network is updated after every $1,000$ iterations. The ϵ-greedy scheme is used for the exploration and exploitation processes. $\epsilon = 1$ at the first iteration and is gradually decayed to 0.01 with a decay factor of 0.9999. For the deep dueling and deep Q-learning algorithms, the learning rate is set at 0.0001, and the discount factor is 0.99. For the Q-learning algorithm, these two values are set at 0.1 and 0.9, respectively.

In this work, we compare our proposed solution with three other approaches: (i) *Greedy*, (ii) *OneNode*, and (iii) *Static Optimal Code*. Under the *Greedy* policy, the MEC server uses all available edge nodes to execute a learning task. This policy is used to investigate the effect of straggling nodes and unstable wireless links on the system performance. Under the *OneNode* policy, an available edge node is randomly selected to serve a learning task. This scheme is used to evaluate the conventional uncoded and non-distributed learning methods. Finally, the *Static Optimal Code* policy is based on the optimal MDS code proposed in [1]. We use this policy to show the performance of static optimal codes that do not take the heterogeneity of edge nodes and wireless links into account.

B. Simulation Results

1) Convergence of Learning Algorithms: First, we evaluate the learning processes of the Q-learning, deep Q-learning, and deep dueling algorithms in Fig. 3. Clearly, the conventional Q-learning algorithm converges at a much slower rate than the deep Q-learning and deep dueling algorithms. The reason is that the Q-learning algorithm usually suffers from the slow-convergence problem, especially in complex systems like the considered coded computing over wireless networks. Note that our proposed deep dueling algorithm can achieve the fastest convergence rate by using the novel deep dueling neural network architecture. Specifically, it can obtain the optimal coding and scheduling policy within $10,000$ iterations, while the deep Q-learning algorithm requires more than $15,000$ iterations to converge to the optimal policy. In the following, all simulation results of the deep dueling algorithm are obtained at 4×10^4 training steps, while the Q-learning algorithm's results are obtained at 10^6 training steps. It is worth noting that the conventional Q-learning algorithm is a benchmark used to show the effectiveness of our proposed deep dueling algorithm.

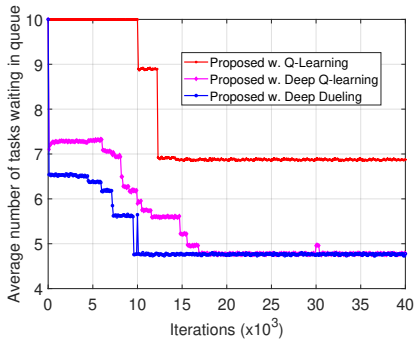
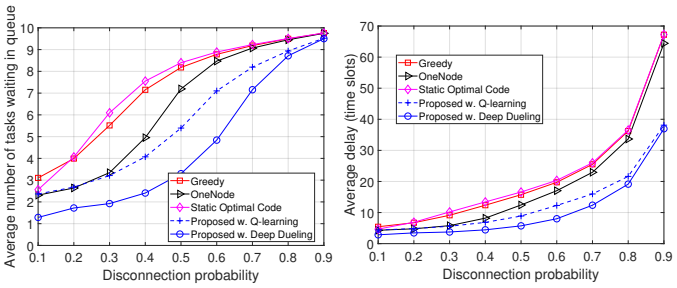


Fig. 3: Convergence rates of learning algorithms.

2) System Performance: In this section, we perform simulations to evaluate the proposed solution's performance in terms of the number of tasks waiting in the queue and the average delay of learning tasks in the system in various scenarios. First, we vary the disconnection probability of wireless links as shown in Fig. 4. It can be observed that when the disconnection probability increases, the system performances of all approaches significantly decrease. This is due to the fact that when the wireless links are likely to be unstable, the MEC server and edge nodes may resend their data more frequently. As such, the average serving time of learning tasks increases, resulting in a high average delay for learning tasks. It is worth noting that when the disconnection probability increases from 0.1 to 0.6, the performance under the *OneNode* policy is superior to that of the *Greedy* policy. This is because under the *OneNode* policy, each learning task is executed by only one edge node. Therefore, the frequency of resending data is much lower than that of the *Greedy* policy. Nevertheless, in

cases with high disconnection probabilities, the performance gap between these two policies is not significant as all wireless links are likely to be disconnected. Note that, in all scenarios, our proposed solution can always achieve the best performance compared to those of the *Greedy* and *OneNode* approaches. This is stemmed from the fact that our proposed algorithm can learn from the environment to avoid highly-straggling wireless links when serving learning tasks. The *Static Optimal Code* achieves the worst performance because this approach does not consider the dynamics and uncertainty of wireless links when determining the optimal MDS code for each learning task.



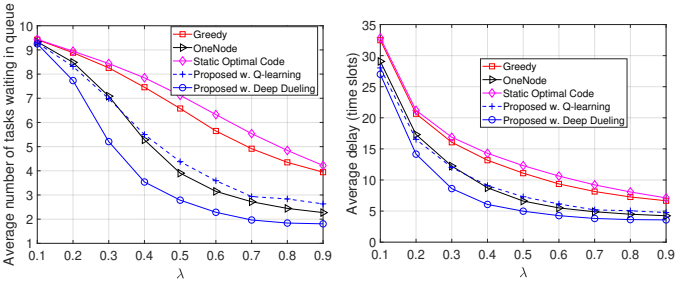
(a) (b)

Fig. 4: (a) Average number of tasks waiting in the queue and (b) average delay of learning tasks in the system vs. disconnection probability of links.

In Fig. 5, we vary the rate parameter λ (in the exponential distribution determining the stochastic computing time of edge nodes) and observe the system performance. As mentioned in Section II-B, a lower value of λ leads to a longer time for processing learning tasks at edge nodes. Consequently, when λ increases, the system performance obtained by all policies will drop. It is worth noting that the performance gap of all policies is small when λ is small. However, this gap is bigger when λ increases. This is because with a lower value of λ, edge nodes may take longer time to process learning tasks, and thus the system resources are likely to be fully utilized. Consequently, the MEC server has fewer options to serve learning tasks, resulting in a small performance gap between solutions. Again, in all scenarios, our proposed solution achieves the best performance by avoiding edge nodes in which straggling problems are likely to happen.

VI. Conclusion

In this paper, we have proposed a novel coding framework to mitigate the straggling problems on both edge nodes and wireless links for distributed learning in wireless edge networks. In particular, we have first developed a Markov decision process framework to jointly optimize the coding and scheduling policy under the dynamics and uncertainty of the system. Then, the conventional Q-learning algorithm is adopted to obtain the optimal policy for the system. However, the Q-learning algorithm requires a very long time to converge to the optimal policy. To address this problem, we have proposed



(a) (b)

Fig. 5: (a) Average number of tasks waiting in the queue and (b) average delay of learning tasks in the system vs. λ.

a highly effective deep reinforcement learning algorithm, namely deep dueling, leveraging the recent advance of the deep dueling neural network architecture. The simulation results have demonstrated that our proposed solution can greatly improve the system performance by not only choosing the optimal MDS code but also finding the best set of edge nodes to execute each learning task.

References

[1] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding Up Distributed Machine Learning Using Codes," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514-1529, Mar. 2018.

[2] S. Prakash, S. Dhakal, M. R. Akdeniz, Y. Yona, S. Talwar, S. Avestimehr, and N. Himayat, "Coded Computing for Low-Latency Federated Learning Over Wireless Edge Networks," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 1, pp. 233-250, Nov. 2020.

[3] C. Karakus, Y. Sun, S. N. Diggavi, and W. Yin, "Redundancy Techniques for Straggler Mitigation in Distributed Optimization and Learning," *Journal of Machine Learning Research*, vol. 20, no. 72, pp. 1-47, Apr. 2019.

[4] F. Wu and L. Chen, "Latency Optimization for Coded Computation Straggled by Wireless Transmission," *IEEE Wireless Communications Letters*, vol. 9, no. 7, pp. 1124-1128, Jul. 2020.

[5] S. Ha, J. Zhang, O. Simeone, and J. Kang, "Coded federated computing in wireless networks with straggling devices and imperfect CSI," *IEEE International Symposium on Information Theory (ISIT)*, Paris, France, 7-12 Jul. 2019.

[6] Z. Wang, T. Schaul, M. Hessel, H. V. Hasselt, M. Lanctot, and N. D. Freitas, "Dueling network architectures for deep reinforcement learning," *ICML*, New York, New York, USA, 20-22 Jun. 2016.

[7] J. Zhang and O. Simeone, "On model coding for distributed inference and transmission in mobile edge computing systems," *IEEE Communications Letters*, vol. 23, no. 6, pp. 1065-1068, Apr. 2019.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529-533, Feb. 2015.

[9] N. V. Huynh, D. T. Hoang, D. N. Nguyen, and E. Dutkiewicz, "Joint Coding and Scheduling Optimization for Distributed Learning over Wireless Edge Networks," [Online]. Available: arXiv:2103.04303.