Resource Optimized Split Federated Learning: A Reinforcement Learning and Optimization Approach

Maher Guizani‡, Latif U. Khan†, Waseem Ullah†, Mohammad A. Islam‡
[mahergzani@gmail.com, latif.khan2@gmail.com, mislam@uta.edu]
‡ University of Texas at Arlington, USA.
† Mohamed Bin Zayed University of Artificial Intelligence, United Arab Emirates.

***Abstract*—Federated learning (FL) offers many benefits, such as better privacy preservation and less communication overhead for scenarios with frequent data generation. In FL, local models are trained on end-devices and then migrated to the network edge or cloud for global aggregation. This aggregated model is shared back with end-devices to further improve their local models. This iterative process continues until convergence is achieved. Although FL has many merits, it has many challenges. The prominent one is computing resource constraints. End-devices typically have fewer computing resources and are unable to learn well local models. Therefore, split FL (SFL) was introduced to address this problem. However, enabling SFL is also challenging due to wireless resource constraints and uncertainties. We formulate a joint end-devices computing resources optimization, task-offloading, and resource allocation problem for SFL at the network edge. Our problem formulation has a mixed-integer non-linear programming problem nature and hard to solve due to the presence of both binary and continuous variables. We propose a double deep Q-network (DDDQN) and optimization-based solution. Finally, we validate the proposed method using extensive simulation results.**

***Index Terms*—Federated learning, split federated learning, double deep Q-network.**

I. Introduction

Recently federated learning (FL) has been extensively investigated to realize better privacy-aware machine learning for various applications [1]. In wireless systems, FL is important mainly for two reasons. These reasons include better privacy preservation and efficient communication learning for scenarios with frequent data generation (e.g., autonomous cars generate $40,000$ Giga octet of data every day). Therefore, it is very challenging to send frequently generated data to the cloud in such scenarios. FL is a suitable solution for such scenarios. Although FL enables many benefits, it also has challenges. It is challenging to train local FL models at end devices due to computing resource constraints. Additionally, sharing learning updates over a wireless channel is challenging and involves many impairments. These impairments will degrade the quality of the FL learning process. To address the challenge of the computing resource constraint, split FL (SFL) was presented in [2]. In SFL, a partial local model is learned at end-devices and partial at the network edge. Few works considered SFL [3]–[6]. The work in [3] evaluated the performance of SFL and FL over wireless networks. Another work [4] proposed a hybrid architecture of SFL and FL. They modeled a wireless communication for their framework and performed significant analysis using simulation results. The work in [5] proposed a novel federated split learning architecture and presented results. Otoum *et al.* [6] presented the use of federated learning, split learning, and transfer learning for intelligent transportation system applications. Different from the works in [3]–[6], we propose a novel dueling-based double deep Q-network for efficient task-offloading and resource allocation in SFL over wireless networks. Our contributions are as follows:

- A joint task-offloading and resource allocation problem for SFL is formulated. Moreover, we consider the latency constraint as a QoS metric for SFL.
- Due to the mixed-integer non-programming (MINLP) nature of the formulated problem, we decompose the main problem into sub-problems: (a) devices computing resource optimization, and (b) joint task-offloading and resource allocation problem. For devices computing resource allocation, we use convex optimization, whereas for the joint task-offloading and resource allocation problem, we use a double deep Q-network (DDQN) due to its combinatorial nature.
- Finally, extensive simulations are carried out for validation of the proposal.

II. System Model and Problem Formulation

Consider a system of resource-constrained Internet of Things (IoT) devices, as shown in Fig. 1. There is a set $\mathcal{S}$ of S resource-constrained IoT devices. Every device has a local data $\mathcal{B}_s, \forall s \in \mathcal{S}$ with each having B_s data points. These devices learn are supposed to learn local models. Additionally, there will be a set $\mathcal{L}$ of L edge servers. However, there are computing resource (i.e., CPU-cycles/sec) limitations. Therefore, there is a need for additional computing resources. To address this challenge, the concept of split federated learning (SFL) (i.e., as shown in Fig. 1) is presented in [2]. Although SFL enables many benefits, there is a need for efficient allocation of wireless and computing resources at end-devices and edge servers. Additionally, the devices engaged in SFL needs to

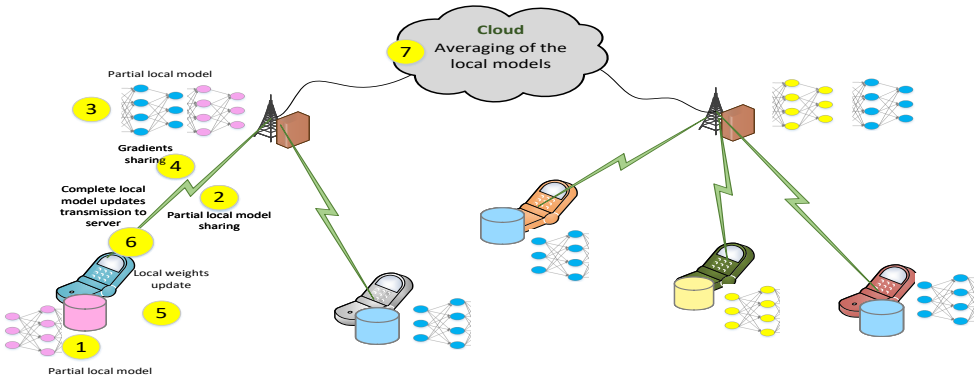


Fig. 1: Split federated learning system model.

offload their tasks to the edge servers efficiently. First, we discuss the wireless SFL model.

A. Wireless SFL Model

A set $\mathcal{S}^{'}$ of $S^{'}$ resource constrained consumer electronic devices want to participate in learning process of a global model. Among $\mathcal{S}^{'}$ of $S^{'}$, the set $\mathcal{S}$ of S will participate in the learning process due communication resources constraints. Every device learns a partial local model due to computing resources constraints. Let the learning task is given by $\mathcal{T}_s(c_s, b_s, t_s) \forall s \in \mathcal{S}$, where ($c_s$ and b_s denote the computing resources needed for learning a particular local model for a single data point and total local data set points, respectively. For a fixed local model architecture, the local latency in computing a partial model is given by:

$$\Xi_s^{\text{local}}(\boldsymbol{\wp}) = \frac{c_s b_s}{\wp_s}, \forall s \in \mathcal{S}, \tag{1}$$

where $\wp_s$ denotes the computing resource assigned to device s. From (1), it is clear that an increase in the local computing resource will minimize the latency but at the cost of high local energy consumption (i.e., energy is proportional to the square of the operating frequency of a device).

$$\ell_s^{\text{local}}(\boldsymbol{\wp}) = \alpha_s c_s b_s \wp_s^2, \forall s \in \mathcal{S}, \tag{2}$$

where α_s denotes the effective capacitance coefficient of device s. On the other hand, the local device computing resources should follow the following limits.

$$\wp_{min} \leq \wp_s \leq \wp_{max}, \forall s \in \mathcal{S}, \tag{3}$$

where $\wp_{min}$ and $\wp_{max}$ denote the minimum and maximum limits, respectively. On the other hand, the summation of computing resources of all devices should not exceed the maximum limit, i.e, $\wp_{MAX}$.

$$\sum_{s=1}^{S} \wp_s \leq \wp_{MAX}. \tag{4}$$

Next to computing the partial local model, it is shared with the edge server where the remaining computation of the local model takes place. At the network edge, there are two ways to compute the partial remaining local model. One way is to provide labels at the network edge and the other way is to share back the data to end-devices for computing loss function using labels. In this work, we assume the output labels are available at the network edge. After the edge servers receive a partial local model, further computation takes place at the network edge. Next to computing at the edge, the gradients will be shared back with the devices. On the other hand, for communication between end-devices and edge servers, we consider an orthogonal frequency division multiple access (OFDMA). As the communication resources have constraints in terms of availability, therefore, it is a good idea to reuse the existing occupied resources by cellular users. For computing the transmission latency, we can write as:

$$\ell_{\text{trans}}(\boldsymbol{\kappa}, \boldsymbol{\upsilon}) = \left(\frac{\kappa_{(s,l)} \upsilon_{(s,r)} O_s}{T_r \left(\log_2 \left(1 + \frac{p_s h_{s,l}}{\sum_{c \in \mathcal{C}} p_c h_{c,b} + N_o^2} \right) \right)} \right), \tag{5}$$

where O_s and T_r denote the learning data overhead of the device s and bandwidth of the resource block r, respectively. $\sum_{c \in \mathcal{C}} p_c h_{c,b}$ is the inference on resource block r. N_o^2 is the noise power. $\kappa_{(s,l)}$ and $\upsilon_{(s,r)}$ are task offloading and resource allocation variables, respectively. The task offloading variable has a value $\kappa_{(s,l)} = 1$ if device s offloads its task to edge server l and $\kappa_{(s,l)} = 0$, otherwise. Similarly, the variable $\upsilon_{(s,r)} = 1$ if the device s is allocated the resource block r and $\upsilon_{(s,r)} = 0$, otherwise. Every edge server has a certain capacity to serve end-devices by performing partial learning task computation for them. Therefore, there is a need for a limit.

$$\sum_{s=1}^{S} \kappa_{s,l} \leq \kappa_{\text{s}}^{max}, \forall l \in \mathcal{L}, \tag{6}$$

where κ_{s}^{max} denotes the maximum limit in serving end-

devices. On the other hand, a single device should offload its learning task to a single edge server.

$$\sum_{l=1}^{L} \kappa_{s,l} \leq 1, \forall s \in \mathcal{S}, \tag{7}$$

Similar to task-offloading, there is a need for constraints on resource allocation, i.e., a single resource block should be allocated to a single device.

$$\sum_{r=1}^{R} \upsilon_{s,r} \leq 1, \forall s \in \mathcal{S}. \tag{8}$$

For transferring learning updates to the edge servers, there is a need for wireless resource blocks. The number of resource blocks needed for transferring of learning updates depends on the learning data size, which in turn depends on the complexity of end-device model architecture. We use a single resource block for an end-device similar to the work in [7].

$$\sum_{s=1}^{S} \upsilon_{s,r} \leq 1, \forall r \in \mathcal{R}. \tag{9}$$

On the other hand, the transmission energy can be given by:

$$\Xi_{\text{trans}}(\boldsymbol{\kappa}, \boldsymbol{\upsilon}) = \left(\frac{\kappa_{(s,l)} \upsilon_{(s,r)} O_s}{T_r \left(\log_2 \left(1 + \frac{p_s h_{s,l}}{\sum_{c \in \mathcal{C}} p_c h_{c,b} + N_o^2} \right) \right)} \right) p_s, \quad \forall s \in \mathcal{S}, \tag{10}$$

After local model is computed, the local model is shared with the remote cloud via small cell base station. The number of rounds between the end-devices and the edge servers can be given by:

$$I_g(\varepsilon, \boldsymbol{\theta}) = \frac{\Gamma \log(1/\varepsilon)}{1 - \theta}, \tag{11}$$

where Γ is the local dataset dependent constant. The term θ is the relative local accuracy. θ has a different interpretation compared to the local accuracy. Lower values of θ are desirable compared to higher values and vice versa. ε is the global accuracy. It is evident from (11) that the global rounds depend on θ, i.e., lower values of θ will generally result in more global rounds and vice versa. Additionally, the latency due to transmission of learning updates should not exceed the maximum limit.

$$\left(\frac{\kappa_{(s,l)} \upsilon_{(s,r)} O_s}{T_r \left(\log_2 \left(1 + \frac{p_s h_{s,l}}{\sum_{c \in \mathcal{C}} p_c h_{c,b} + N_o^2} \right) \right)} \right) \leq \phi_{max}, \quad \forall s \in \int, r \in \mathcal{R}. \tag{12}$$

Next, we write our formulated problem.

B. Problem Formulation

For a problem formulation, first, we should define an objective function. Our objective function is the cost of training SFL model. In our case, we consider both latency and energy while writing the objective function. To do so, first, we write energy consumption as in [1]:

$$\Xi_{\text{total}}(\boldsymbol{\kappa}, \boldsymbol{\upsilon}, \boldsymbol{\wp}, \boldsymbol{\theta}) = (1 + \theta) \Xi_{\text{trans}}(\boldsymbol{\kappa}, \boldsymbol{\upsilon}) + \Xi_s^{\text{local}}(\boldsymbol{\wp}), \tag{13}$$

In (13), the notion of θ is to reflect the effect of the relative local accuracy on energy cost. For higher values of θ, the cost will be high and vice versa. Now, we write the total latency as:

$$\ell_{\text{total}}(\boldsymbol{\kappa}, \boldsymbol{\upsilon}, \boldsymbol{\wp}, \boldsymbol{\theta}) = \ell_{\text{trans}}(\boldsymbol{\kappa}, \boldsymbol{\upsilon}) + \ell_s^{\text{local}}(\boldsymbol{\wp}) \tag{14}$$

Similar to (13), one can rewrite (14) as:

$$\ell_{\text{total}}(\boldsymbol{\kappa}, \boldsymbol{\upsilon}, \boldsymbol{\wp}, \boldsymbol{\theta}) = (1 + \theta) \ell_{\text{trans}}(\boldsymbol{\kappa}, \boldsymbol{\upsilon}) + \ell_s^{\text{local}}(\boldsymbol{\wp}) \tag{15}$$

Next, we write the overall cost as:

$$\mathcal{C}(\boldsymbol{\kappa}, \boldsymbol{\upsilon}, \boldsymbol{\wp}, \boldsymbol{\chi}, \boldsymbol{\theta}) = \beta \ell_{\text{total}} + (1 - \beta) \Xi_{\text{total}}, \tag{16}$$

where β denotes the scaling constant and one can use it to scale the proportion of energy and latency while computing the cost for learning SFL model. Now, we write our formulated problem whose goal is to minimize the overall cost of learning the SFL model.

$$\begin{aligned} \mathbf{P} - \mathbf{M}: \quad & \underset{\kappa, \upsilon, \wp, \theta}{\text{minimize}} \quad \mathcal{C}(\boldsymbol{\kappa}, \boldsymbol{\upsilon}, \boldsymbol{\wp}, \boldsymbol{\theta}) \\ & \text{subject to:} \\ & (3), (4), (6) - (9), (12). \end{aligned} \tag{17}$$

The formulated problem is a MINLP problem. It is difficult to solve the formulated problem directly. Additionally, one can not use conventional optimization schemes. Therefore, we will use a decomposition-based scheme as explained in the next section.

III. Proposed Optimization and Multi-Agent Reinforcement Learning Solution

Our formulated problem is MINLP and has a challenging nature to solve. Therefore, we propose a scheme based on decomposition. Our approach is based on division of the main problem into two sub-problems. These sub-problems are devices computing resource allocation and joint task-offloading as well as resource allocation.

A. Devices Computing Resource Allocation Problem

We write devices computing resource allocation problem as:

$$\mathbf{P-DR}: \underset{\wp}{\text{minimize}} \ \mathcal{C}(\boldsymbol{\wp}) \tag{18}$$

subject to:

$$\wp_{min} \leq \wp_s \leq \wp_{max}, \forall s \in \mathcal{S}, \tag{18a}$$

$$\sum_{s=1}^{S} \wp_s \leq \wp_{MAX}. \tag{18b}$$

Problem **P-DR** is the minimize the end-devices latency in computing partial local models. Problem **P-DR** is a convex optimization problem whose convexity can be proved by checking the objective function and constraints. For fixed task-offloading variable, resource allocation, edge servers computing resource allocation, the objective function depends only on $\wp$. To check the convexity, we first take the convexity by taking the derivative twice with respect to $\wp$. For feasible values of $\wp$, the objective function results in positive values. Based on this, we can say that the objective function is convex. Additionally, the constraints are linear inequality constraints. Therefore, the problem **P-DR** is a convex optimization problem.

B. Joint Task-Offloading and Resource Allocation Problem

We write a joint task-offloading and resource allocation problem as:

$$\mathbf{P-TR}: \underset{\kappa,\upsilon}{\text{minimize}} \ \mathcal{C}(\boldsymbol{\kappa}, \boldsymbol{\upsilon}) \tag{19}$$

subject to:

$$\sum_{s=1}^{S} \kappa_{s,l} \leq \kappa_{\mathrm{s}}^{max}, \forall l \in \mathcal{L}, \tag{19a}$$

$$\sum_{l=1}^{L} \kappa_{s,l} \leq 1, \forall s \in \mathcal{S}, \tag{19b}$$

$$\sum_{r=1}^{R} \upsilon_{s,r} \leq 1, \forall s \in \mathcal{S}, \tag{19c}$$

$$\sum_{s=1}^{S} \upsilon_{s,r} \leq 1, \forall r \in \mathcal{R}, \tag{19d}$$

$$\Omega_{s,r} \leq \phi_{max}, \forall s \in \int, r \in \mathcal{R}, \tag{19e}$$

$$\kappa_{s,l} \in \{0,1\}, \upsilon \in \{0,1\}, \tag{19f}$$

Problem **P-TR** is combinatorial problem and can not be solved using convex optimization schemes. To solve this, one can use a relaxation and decomposition based scheme [8]. This approach has an inherent limitation of approximation errors in conversion from binary task-offloading and resource allocation variables into continuous and then, continuous back to binary variables. To overcome this limitation, another way is to use an exhaustive search scheme [9]. However, the exhaustive search scheme suffers from high computing complexity. Based on aforementioned discussion, we propose a solution based on DDQN.

For DDQN, first, we formulate our scenario as a stochastic game. All users in the proposed system are selfish and want to maximize their long-term rewards by choosing edge servers and resource blocks. For any time t, the reward of a particular device/user depends on the environment current state as well as other devices actions. For the stochastic game formulation, our optimization problem can be give by $\langle \mathcal{S}, \mathcal{G}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ [10].

- $\mathcal{S}$ is the set of S devices.
- $\mathcal{G}$ is the set of all possible states.
- $\mathcal{A}_s$ denotes the action of device s. Moreover, the vector of the joint action can be given by $\overrightarrow{a} = (a_1, a_2, a_3, ..., a_S)\mathrm{T} \in \ \times_s \mathcal{A}_s$
- $\mathcal{P}$ is the state transition probability.
- $\mathcal{Q}_s$ is the reward function of the s device.

In our system, all users will choose an edge server and resource block, therefore, the action space can be given by.

$$a_{ls}^r(t) = \{l_s^l(t), r_s^r(t)\}. \tag{20}$$

where $l_s^l(t) \in \{0,1\}$, $l_s^l(t) \in \{l_s^0(t), l_s^2(t), ..., l_s^{L-1}(t)\}$ and $r_s^r(t) \in \{0,1\}, r_s^r(t) \in \{r_s^0(t), r_s^1(t), ..., r_s^{R-1}(t)\}$. In our system, the total number of states can be given by the product of edge servers-based BSs and the number of resource blocks LR. For a given time t, the action of other end-devices can be given by $\mathcal{A}_{-1}(t) = \{a_{l1}^r(t), ..., a_{ls-1}^r(t), a_{ls+1}^r(t), ..., a_{ls}^r(t)\}$. Note here that the immediate reward $\mathcal{R}_s(t) = \mathcal{R}(g, \mathcal{A}_s, \mathcal{A}_{-s})$ of any device s depends on its current action $\mathcal{A}_s$ and other devices actions $\mathcal{A}_{-s}$. On the other hand, to determine the stable and mutually optimal strategies in stochastic games, one can use the concept of Nash Equilibrium. The stochastic game is said to achieve the Nash Equilibrium if the following holds true.

$$\mathcal{Q}_s(t) = \mathcal{Q}(g, \mathcal{A}_s^*, \mathcal{A}_{-s}^*) \geq \mathcal{Q}_s(t) = \mathcal{Q}(g, \mathcal{A}_s, \mathcal{A}_{-s}^*), \quad \forall \mathcal{A}_i \in \mathcal{A} \tag{21}$$

Now, we use finite-state Markov decision process (MDP) to model the stochastic game. The reward of the devices depends on the current state and actions. Therefore, the process follows the Markov property. The MDP can be given by $\langle \mathcal{S}, \mathcal{G}, \mathcal{A}, \mathcal{P}, \mathcal{Q} \rangle$. $\mathcal{Q}$ in our system is equal to the $\frac{1}{\mathcal{C}}$. The optimal policy π_i^* is obtained by maximizing the value-state function $V_i(s, \pi_i, \pi_{-i})$ of each UE in each state:

$$V_i(g, \pi_i, \pi_{-i}) = \mathbb{E}_{\tau \sim (\pi_i, \pi_{-i})} \left[\sum_{t=0}^{T-1} \gamma^t \mathcal{Q}_i(g_t, a_{i,t}, a_{-i,t}) \,\middle|\, g_0 = g \right], \tag{22}$$

where $\mathbb{E}_{\tau \sim (\pi_i, \pi_{-i})}$ is the expectation taken over all possible trajectories τ generated by the joint policies π_i and π_{-i}, where π_{-i} denotes the strategies of other agents. γ is the discount

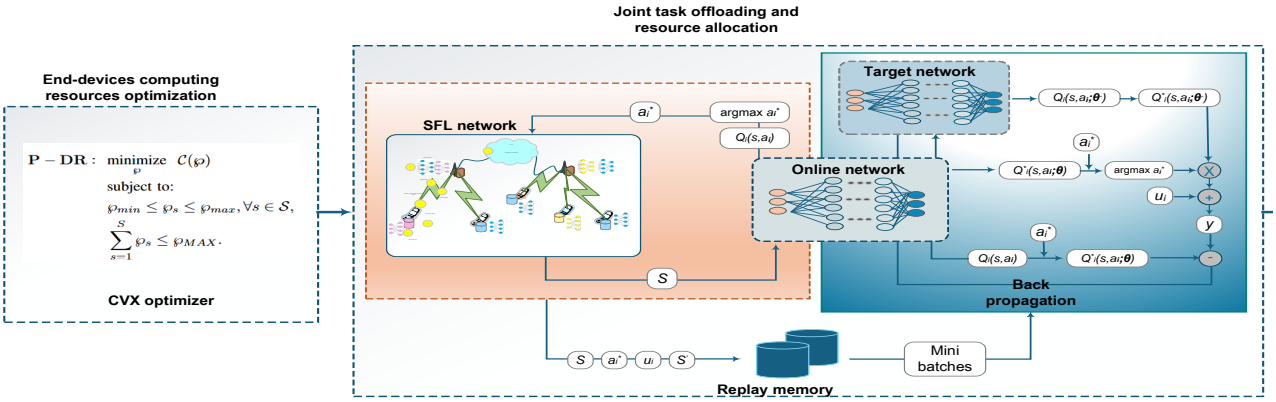


Fig. 2: Proposed solution approach.

factor. If we consider the Markov property, then we can write the value function as:

$$V_i(g, \pi_i, \pi_{-i}) = u_i(g_t, \pi_i, \pi_{-i}) + \gamma \sum_{g' \in \mathcal{G}} P_{gg'}(\pi_i, \pi_{-i}) V_i(g', \pi_i, \pi_{-i}). \tag{23}$$

where $u_i(g_t, \pi_i, \pi_{-i}) = \mathbb{E}\left[\mathcal{Q}_i(g, \pi_i, \pi_{-i})\right]$, and $P_{gg'}(\pi_i, \pi_{-i})$ is the state transition probability. A NE exists if for each π_i, if the following exists:

$$V_i(g, \pi_i^*, \pi_{-i}^*) \geq V_i(g, \pi_i, \pi_{-i}^*), \quad \forall g \in \mathcal{G}. \tag{24}$$

To solve this MDP, we use Q-learning. The optimal Q-value function $Q_i^*(g, a_i)$ is calculated through:

$$Q_i^*(g, a_i) = u_i(g_t, a_i, \pi_{-i}^*) + \gamma \sum_{g' \in \mathcal{G}} P_{gg'}(\pi_i^*, \pi_{-i}^*) V_i(g', \pi_i^*, \pi_{-i}^*), \tag{25}$$

where $V_i(g', \pi_i^*, \pi_{-i}^*) = max_{a_i \in \mathcal{A}_i} Q_i^*(g, a_i)$. Now, write (25) as follows:

$$V_i(g, \pi_i, \pi_{-i}) = u_i(g_t, \pi_i, \pi_{-i}) + \gamma \sum_{g' \in \mathcal{S}} P_{gg'}(\pi_i, \pi_{-i}) \max_{a_i \in \mathcal{A}_i} Q_i^*(g, a_i). \tag{26}$$

It is very difficult to obtain information on the transition probability. Therefore, for Q-learning, $Q_i^*(g, a_i)$ is updated by:

$$Q_i(g, a_i) = Q_i(g, a_i) + \delta\{\mathbb{E}[u_i(g_t, a_i, \pi_{-i}) + \gamma \max_{a_i' \in \mathcal{A}_i} Q_i(g', a_i') - Q_i(g, a_i)\}], \tag{27}$$

where δ denotes the learning rate which helps in determining the Q value. The value of δ should be selected appropriately in order for fast convergence. Additionally, for the action selection, there is a need for tradeoff in action selection. We use ϵ greedy strategy for selecting action. The random action is selected with a probability ϵ and the action is selected with maximum reward with probability $1 - \epsilon$. The aforementioned Q-learning method performs better for small state and action spaces. For large action spaces, Q-learning do not perform well. Therefore, there is a need for a better approach. To address this limitations, we can use deep Q-network (DQN) that uses a neural network to represent action and state space. DQN uses two networks: (a) online network and (b) target network. The use of target network is to stabilize the overall performance. In Q-network, the following loss function is used to stabilize the performance.

$$L_i(\theta) = \mathbb{E}_{g, a_i, u_{g,a_i}, g'}[(y_i^{DQN} - Q_i(g, a_i; \theta))^2], \tag{28}$$

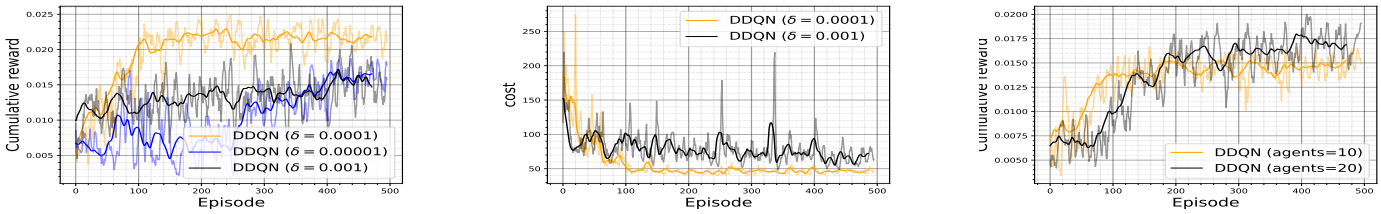
where $y_i^{DQN} = u_i(g, a_i) + \gamma \max_{a_i' \in \mathcal{A}_i} Q_i(g^{'} a_i^{'}; \theta^-$. Here, the term θ^- is the target network weights. To chose the action a_i, one can use online network $Q_i(g, a_i; \theta)$. On the other hand, the weights of the target network are kep fixed for many rounds to provide more stability. Further, experience replay strategy further improves the stability. The current transition is stored in a reply memory. Instead of using only the current transition, we previous transition are also considered for better results. Although DQN offers some merits, it still has a challenges of over optimistic estimation. To address this issue, we can use a double DQN (DDQN) that is based on using a different expression for y_i^{DQN}. For DDQN, y_i^{DDQN} can be given by:

$$y_i^{DDQN} = u_i(g, a_i) + \gamma Q_i\left(s^{'}, \max_{a_i' \in \mathcal{A}_i} Q_i(g^{'} a_i^{'}; \theta^-\right). \tag{29}$$

Fig. 2 shows the steps of DDQN. DDQN uses both online and target network to compute optimal value $Q_i(g, a_i; \theta)$. Then, with the discount factor and current reward, the y_i^{DDQN} is computed. Next, we present simulation results.

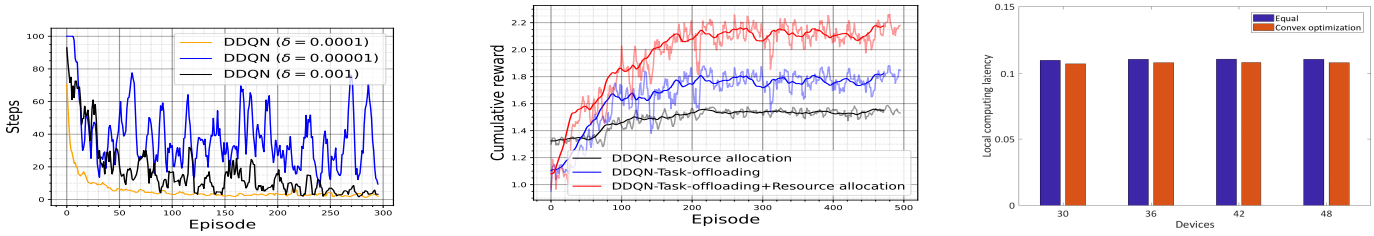
IV. Performance Evaluation

In our simulation environment, we consider 20 agents and 5 edge server-based BSs. An area of 1000×1000 is considered for simulations. All the devices are generated randomly and agents are deployed on them. Meanwhile, the BS locations are keep fixed. We use various learning rates, i.e., $\delta = 0.0001, 0.001, 0.00001$. The number of resource blocks are considered equal to the number of agents. Every agent will get one resource block. For agent, we use a fully connected network architecture that consists of an input, two hidden layers (i.e., $nn.Linear(64,\ 32)$ and $nn.Linear(32,\ Num.\ of\ BSs\ *\ Resource\ blocks)$), and an output. Furthermore, a ReLU activation function is used. For action, we use ϵ-greedy policy. The value of ϵ is chosen linearly between 0 and 0.9. Fig. 3a shows the performance of DDQN for various values of learning rate. For the learning rate 0.001, the performance is worst and does not improve with an increase in number of episodes. For the learning rates 0.0001 and 0.00001, the performance improves with an increase in number of episodes. For the learning rate 0.0001, the performance is the best. Another Fig. 3b shows the cost vs. episodes for DDQN using various learning rates. On the other hand, we study the performance of various schemes (i.e., proposed and baselines) in Fig. 3b. It is evident that the proposed scheme outperforms both



(a) (b) (c)

Fig. 3: (a) Cumulative reward vs. episodes for proposed DDQN using various values of learning rates, (b) cost vs. episodes for proposed DDQN using various values of learning rates, and (c) Cumulative reward vs. episodes for proposed DDQN using various number of agents.



(a) (b) (c)

Fig. 4: (a) Steps needed to meet QoS vs. episodes for proposed DDQN using various values of learning rates, (b) steps needed to meet QoS vs. episodes for proposed DDQN using various number of agents, and (c) cost for proposed convex optimization scheme vs. random scheme.

baselines which shows its superior performance. Fig. 3c shows the reward vs. episodes for various numbers of agents. Again, it is evident that our proposed scheme has a stable performance for a number of agents. Now, we study the number of steps needed within episodes for all devices to reach the QoS (i.e., latency less than a threshold). It is evident from Fig. 4a that performance of DDQN for learning rate 0.0001 is better as it needs less steps to reach the QoS by all devices. Additionally, for different number of agents, the number of steps needed to fulfill QoS for various number of devices remains stable for learning rate 0.0001. Finally, we compare the performance of convex optimization-based local computing resource optimization vs. equal computing resource assignment. The convex optimization scheme outperforms equal allocation scheme.

V. Conclusion

In this paper, we have considered SFL and model optimization problems for local computing resource optimization, task-offloading, and resource allocation. Additionally, we considered QoS constraints to ensure the latency for SFL. We used optimization theory and DDQN for optimizing computing resource allocation and joint task-offloading as well as resource allocation, respectively. Our extensive analysis showed the better and more stable nature of our proposed solution. We concluded that the framework and solution can be applied to many SFL-based practical applications.

References

[1] L. U. Khan, M. Guizani, A. Al-Fuqaha, C. S. Hong, D. Niyato, and Z. Han, "A joint communication and learning framework for hierarchical split federated learning," *IEEE Internet of Things Journal*, 2023.

[2] C. Thapa, P. C. M. Arachchige, S. Camtepe, and L. Sun, "Splitfed: When federated learning meets split learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 8, 2022, pp. 8485–8493.

[3] G. Yansong, M. KIM, S. ABUADBBA *et al.*, "End-to-end evaluation of federated learning and split learning for internet of things," in *Proceedings of 2020 International Symposium on Reliable Distributed Systems (SRDS), Shanghai, China*, vol. 4, 2020.

[4] X. Liu, Y. Deng, and T. Mahmoodi, "Wireless distributed learning: A new hybrid split and federated learning approach," *IEEE Transactions on Wireless Communications*, vol. 22, no. 4, pp. 2650–2665, 2022.

[5] V. Turina, Z. Zhang, F. Esposito, and I. Matta, "Federated or split? a performance and privacy analysis of hybrid split and federated learning architectures," in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. IEEE, 2021, pp. 250–260.

[6] S. Otoum, N. Guizani, and H. Mouftah, "On the feasibility of split learning, transfer learning and federated learning for preserving security in its systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 7, pp. 7462–7470, 2022.

[7] M. Chen, Z. Yang, W. Saad, C. Yin, H. V. Poor, and S. Cui, "A joint learning and communications framework for federated learning over wireless networks," *IEEE transactions on wireless communications*, vol. 20, no. 1, pp. 269–283, 2020.

[8] R. S. Klein, H. Luss, and U. G. Rothblum, "Relaxation-based algorithms for minimax optimization problems with resource allocation applications," *Mathematical programming*, vol. 64, pp. 337–363, 1994.

[9] J. Nievergelt, "Exhaustive search, combinatorial optimization and enumeration: Exploring the potential of raw computing power," in *International Conference on Current Trends in Theory and Practice of Computer Science*. Springer, 2000, pp. 18–35.

[10] N. Zhao, Y.-C. Liang, D. Niyato, Y. Pei, M. Wu, and Y. Jiang, "Deep reinforcement learning for user association and resource allocation in heterogeneous cellular networks," *IEEE Transactions on Wireless Communications*, vol. 18, no. 11, pp. 5141–5152, 2019.