Incentive-Driven Long-term Optimization for Edge Learning by Hierarchical Reinforcement Mechanism

Yi Liu*,[1], Leijie Wu*,[1], Yufeng Zhan†,[2], Song Guo†,[1], and Zicong Hong[1]
[1]*Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China*
[2] *School of Automation, Beijing Institute of Technology, Beijing, China*
csyiliu@comp.polyu.edu.hk, cslwu@comp.polyu.edu.hk, yu-feng.zhan@bit.edu.cn
song.guo@polyu.edu.hk, zicong.hong@connect.polyu.hk

***Abstract*—Edge Learning is an emerging distributed machine learning in mobile edge network. Limited works have designed mechanisms to incentivize edge nodes to participate in edge learning. However, their mechanisms only consider myopia optimization on resource consumption, which results in the lack of learning algorithm performance guarantee and long-term sustainability. In this paper, we propose Chiron, an incentive-driven long-term mechanism for edge learning based on hierarchical deep reinforcement learning. First, our optimization goal combines learning-algorithms metric (i.e., model accuracy) with system metric (i.e., learning time, and resource consumption), which can improve edge learning quality under a fixed training budget. Second, we present a two-layer H-DRL design with exterior and inner agents to achieve both long-term and short-term optimization for edge learning, respectively. Finally, experiments on three different real-world datasets are conducted to demonstrate the superiority of our proposed approach. In particular, compared with the state-of-the-art methods under the same budget constraint, the final global model accuracy and time efficiency can be increased by 6.5% and 39%, respectively. Our implementation is available at https://github.com/Joey61Liuyi/Chiron.**

***Keywords*-Federated Learning; Incentive Mechanism; Deep Reinforcement Learning; Mobile Edge Computing;**

I. Introduction

Currently, we have witnessed machine learning based on artificial neural networks acquires unprecedented success in various tasks [1]–[3]. In order to achieve a high-quality model, large amounts of training data are expected to be fed to the machine learning model. However, in most applications, the training data are generated by resource-constrained devices (e.g., tablets, smartphones, etc.) in network edge. Due to the limited network bandwidth and privacy concerns, it would be impractical to upload all the raw data to the cloud for centralized model training.

Thanks to the rapid development of edge computing, a new paradigm of distributed machine learning in mobile edge network named edge learning has been proposed, which allows distributed edge nodes to train machine learning models cooperatively without exposing their raw data. Similar to the distributed machine learning in the cloud, edge learning works round by round. In each training round, edge nodes download the global machine learning model from the parameter server and train the local machine learning models using their own data by multiple epochs, respectively. Then, the updated machine learning models are uploaded to the parameter server for aggregating a new global machine learning model. Since an inception by Google [4], both academia [5]–[13] and industry [14] are paying great attentions to this emerging field. A large amount of works have studied edge learning from different angles, e.g., resource-constraint edge learning algorithm optimization [7], resource optimization on heterogeneous edge nodes [9], attack resistance on malicious edge nodes [15], heterogeneous edge node selection [5], and momentum edge learning algorithm [16].

Although there are a great number of works focusing on edge learning, most of them are based on an assumption of edge nodes' voluntary participation. While participating in edge learning tasks, edge nodes will consume their own resources such as battery and computing power. In addition, data has become an important asset, edge nodes face data asset leakage by sharing their machine learning models with the parameter server. Therefore, an edge node would not be interested in voluntarily participating in edge learning, unless it receives a satisfying bonus to compensate its resource consumption and potential data asset leakage. Without adequate edge node participation, it is impossible for the parameter server to achieve a good machine learning model, since machine learning model is truly dependent on edge nodes' training data.

A few of researchers have studied incentive mechanisms for edge learning [8], [17]–[20], but they all neglect several key factors of edge learning as follows. **First**, they only consider the myopia optimization of single round, which results in the lack of long-term sustainability, such as consuming too many resources prematurely under the same budget. **Second**, they do not take the characteristics of different learning algorithms into the mechanism design, which makes the existing approaches unable to guarantee the quality of the

*These authors contributed equally to this work.
†Song Guo and Yufeng Zhan are the corresponding authors.

final machine learning model.

In this paper, we first consider to combine the performance of edge learning algorithm into the optimization goal design of incentive mechanism, and give the optimal strategy analysis for our mechanism. Since it is hard for the parameter server to get the private information of the edge nodes and mathematical model of the machine learning, then it is impossible to obtain the closed-form analytical solution for the incentive-driven edge learning mechanism. Thus, we adopt experience-driven DRL approach to learn the optimal pricing strategy without any prior knowledge. Besides, the traditional DRL with only one agent cannot adapt to the long-term and short-term dual optimization goals. We creatively propose Chiron, a hierarchical DRL approach to achieve both long-term and short-term optimization for edge learning, which can intelligently determines the pricing strategy for each edge node in each training round.

The main contributions of this paper are summarized as follows:

- To the best of our knowledge, this is the first work to design the incentive-driven edge learning, which takes the learning algorithm performance into the incentive mechanism design.
- We propose a hierarchical DRL-based incentive mechanism for the edge learning. Without knowing any prior information of the edge nodes, it can effectively learn the optimal pricing strategy of the parameter server and achieve the long-term sustainability of edge learning under a limited budget constraint.
- We implement our mechanism in an edge learning simulator developed from scratch using PyTorch, and evaluate it under a variety of edge learning tasks. Our experimental results on the MNIST, Fashion-MNIST, and CIFAR-10 datasets have further demonstrated the superiority of proposed approach.

The rest of this paper is organized as follows. The background of this work is reviewed in Section II. Section III presents the system model. The problem formulation and optimal strategy analysis is demonstrated in Section IV. In Section V, we present the design and implementation of Chiron. Section VI comprehensively evaluates the performance of Chiron compared with other two baselines. We review some related work in Section VII with the conclusion in Section VIII.

II. Background

In this section, we briefly introduce the background of edge learning and deep reinforcement learning in general.

A. Edge Learning

For a machine learning model $\boldsymbol{\omega}$, the loss function on the training sample j can be defined as $f(\boldsymbol{\omega}, \boldsymbol{x}_j, y_j)$, where $\boldsymbol{x}_j$ is the input of the machine learning model and y_j is the desired output. We abbreviate it to $f_j(\boldsymbol{\omega})$. The loss function captures the deviation of the model on each training sample, and the target of machine learning process is to minimize the average loss function based on all the training samples. Note that we use supervised learning to describe our problem in this paper.

Assume that we have a set $\mathcal{N} = \{1, 2, \cdots, N\}$ of edge nodes with their own local dataset $\mathcal{D}_1$, $\mathcal{D}_2$, $\cdots$, $\mathcal{D}_N$. For each dataset $\mathcal{D}_i$ at edge node i, the loss function on its training samples satisfies

$$F_i(\boldsymbol{\omega}) = \frac{1}{|\mathcal{D}_i|} \sum_{j \in \mathcal{D}_i} f_j(\boldsymbol{\omega}), \tag{1}$$

where $|\mathcal{D}_i|$ is the size of $\mathcal{D}_i$. For convenience, we will use D_i instead of $|\mathcal{D}_i|$ in the following, and $D = \sum_{i=1}^{N} D_i$.

Then, the global loss function on all the distributed training data can be defined as

$$F(\boldsymbol{\omega}) = \frac{\sum_{j \in \cup_i \mathcal{D}_i} f_j(\boldsymbol{\omega})}{|\cup_i \mathcal{D}_i|} = \frac{\sum_{i=1}^{N} D_i F_i(\boldsymbol{\omega})}{D}. \tag{2}$$

The learning problem is to find out

$$\boldsymbol{\omega}^* = \arg\min_{\boldsymbol{\omega}} F(\boldsymbol{\omega}). \tag{3}$$

It cannot be directed computed without sharing information among the edge nodes and the inherent complexity of most machine learning models, we cannot derive the closed-form solution to Eqn. (3). Therefore, the distributed gradient descent approach named edge learning has been applied to solve Eqn. (3) [4].

Edge learning works round by round, in the k-th training round, each edge node has its own model parameters $\boldsymbol{\omega}_{i,k}^{\iota}$, where $\iota = 1, 2, \cdots, \sigma$ denotes the local epoch index. We set the edge learning system performs σ epochs of local updates at each edge node in each training round. At the beginning of the k-th training round, all the edge nodes download the model parameters $\boldsymbol{\omega}_k$ from the parameter server, and set their local model parameters as $\boldsymbol{\omega}_{i,k}^{0} = \boldsymbol{\omega}_k$. For $\iota \in (0, \sigma]$, new values of $\boldsymbol{\omega}_{i,k}^{\iota}$ are computed according to a stochastic gradient descent update rule on the local loss function, based on the model parameters in the last epoch. For each edge node i, the local update rule is $\boldsymbol{\omega}_{i,k}^{\iota} = \boldsymbol{\omega}_{i,k}^{\iota-1} - \mu \nabla F_i(\boldsymbol{\omega}_{i,k}^{\iota-1})$.

After σ local updates, all the edge nodes upload the newly updated model parameters to the parameter server. When the parameter server has collected updates from all the edge nodes in round k, it performs the averaging algorithm to update the global model as

$$\boldsymbol{\omega}_{k+1} = \sum_{i=1}^{N} \frac{D_i}{D} \boldsymbol{\omega}_{i,k}^{\sigma}. \tag{4}$$

B. Deep Reinforcement Learning

DRL is the learning process of an agent that acts in corresponding to the environment to obtain the maximal reward. With the interaction between DRL agent and environment, at present lots of work has been successful [21]–[24]. Specifically, at each time step k, the DRL agent observes a state $\boldsymbol{s}_k$ and perform an action $\boldsymbol{a}_k$. After the action is executed, the state of the environment then transmits to the next state $\boldsymbol{s}_{k+1}$, and the DRL agent receives a reward $\boldsymbol{r}_k$. The state transitions and rewards follow a Markov Decision Process [25], which is discrete time stochastic control process, denoted by a sequence $\boldsymbol{s}_1$, $\boldsymbol{a}_1$, r_1, $\boldsymbol{s}_2$, $\cdots$, r_{k-1}, $\boldsymbol{s}_k$, $\boldsymbol{a}_k$, r_k, $\cdots$. The objective of DRL is to maximize the expected cumulative discounted return $R = \sum_k \gamma^{k-1} r_k$, where $\gamma \in [0, 1]$ is a factor discounting future rewards.

The DRL agent seeks to learn a mapping relation that points our the actions leading to maximum cumulative return under a particular state. Based on the policy $\pi(\boldsymbol{a}|\boldsymbol{s})$, the DRL agent will take an action $\boldsymbol{a}$ under the state $\boldsymbol{s}$. For most problems, learning all the possible combinations of state-action pairs is impossible, hence function approximation technique [26] is commonly used to represent the policy. A function approximator $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})$ is parameterized by $\boldsymbol{\theta}$, whose size is much smaller than the number of all possible state-action pairs. There are lots of function approximator forms, deep neural networks (DNNs) have been widely used to solve complex control problems. In this paper, the DNN is applied as the function approximators in Chiron. Then, the goal of the DRL is [27]

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \mathrm{E}_{\tau \sim p_{\boldsymbol{\theta}}(\tau)} R, \tag{5}$$

where τ is the trajectory of the interaction between DRL agent and environment. Then, the DRL agent can learn the policy by using gradient-descent to update $\boldsymbol{\theta}$ with the state $\boldsymbol{s}_k$, action $\boldsymbol{a}_k$, and the reward r_k in each step k.

III. System Model

In the system, we assume there is a parameter server which resides in the cloud and a set of edge nodes that connect to the parameter server via the network elements. The parameter server incentivizes the edge nodes to participate in the edge learning via bonus. More specifically, the system model is presented as follows.

As shown in Fig. 1, edge learning process consists of multiple rounds. In the k-th training round, the edge nodes first download the global machine learning model $\boldsymbol{\omega}_k$ from the parameter server. Since the downlink bandwidth is much larger than that of the uplink, we assume that the machine learning model downloading time is negligible as compared with the uploading time [8], [9]. Edge nodes then train the model $\boldsymbol{\omega}_k$ using stochastic gradient descent algorithms fed by their local training datasets.

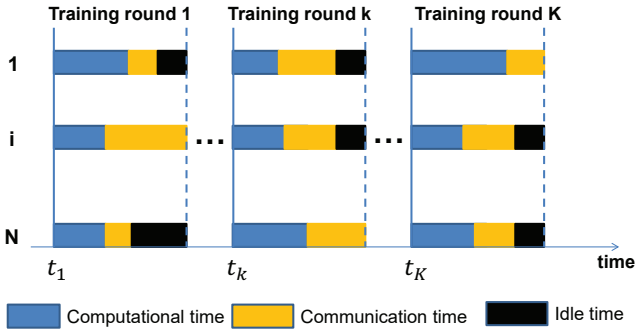


Figure 1. Illustration of edge learning operational process.

Let the number of CPU cycles for edge node i to execute one bit of training data as c_i, which can be measured offline and is known as a prior [8], [9]. We use $\zeta_{i,k} \in [\zeta_{i,k}^{min}, \zeta_{i,k}^{max}]$ to denote the CPU cycle frequency of edge node i to execute the edge learning task in the k-the training round, where ζ_i^{min} and ζ_i^{max} are the minimal and maximal CPU cycle frequency of edge node i, respectively. Therefore, the computational time of edge node i in the k-th round can be calculated by

$$T_{i,k}^{cmp} = \frac{\sigma c_i d_i}{\zeta_{i,k}}, \tag{6}$$

where d_i is the number of bits to execute the local training in one epoch, and σ is the number of local epochs in each training round.

When edge node i completes the local training, it starts to upload the newly updated model to the parameter server. Here, we use $T_{i,k}^{com}$ to denote the model uploading time of edge node i in the k-the training round, and it can be calculated by

$$T_{i,k}^{com} = \frac{\xi}{B_{i,k}}, \tag{7}$$

where ξ is the size of machine learning model and $B_{i,k}$ is the network bandwidth of edge node i in the k-th training round. Then the total time of edge node i to execute the local training in the k-th round is $T_{i,k} = T_{i,k}^{cmp} + T_{i,k}^{com}$. As shown in Fig. 1, the learning time in the k-th round satisfies $T_k = \max_i T_{i,k}$.

According to the widely used energy model [6], [8], [28], the energy consumption of edge node i in the k-th training round is mainly composed of two parts: computing and communication energy consumption. The computing energy consumption is $E_{i,k}^{cmp} = \sigma \alpha_i c_i d_i {\zeta_{i,k}}^2$, where α_i is the effective capacitance coefficient of edge node i's computing chipset. And the communication energy consumption is $E_{i,k}^{com} = \epsilon_i T_{i,k}^{com}$, where ϵ_i is the unit energy consumption for uploading the local updated model, which is independent with $\zeta_{i,k}$. Then the energy consumption is $E_{i,k} = E_{i,k}^{cmp} + E_{i,k}^{com}$.

Let $p_{i,k}$ denote the price of parameter server given to edge

node i for the unit CPU cycle frequency contribution in the k-th training round. Given $p_{i,k}$, the utility of edge node i in the k-th round satisfies

$$u_{i,k} = p_{i,k}\zeta_{i,k} - E_{i,k}. \tag{8}$$

In order to drive the edge nodes to execute the edge learning task, the parameter server should pay enough bonuses to compensate them. Based on the edge node's utility Eqn. (8), in each training round, the pricing strategy will directly affect the edge node's computational resource allocation strategy, and then the training speed of edge learning. For edge learning tasks, it aims to get the best machine learning model with the fastest speed. Then, the utility for edge learning is

$$u = \lambda \cdot A(\boldsymbol{\omega}_K) - \sum_{k=1}^{K} T_k, \tag{9}$$

where $A(\boldsymbol{\omega}_K)$ is the accuracy of global machine learning model after K rounds training. As different edge learning tasks have different preferences on learning time, and model performance, λ can be used to customize the preference.

To the best of our knowledge, we are the first to indeed combine the performance of the edge learning algorithm with the optimization goal of the incentive mechanism. Such a performance-aware incentive mechanism can not only effectively encourage edge nodes' participation, but also ensure the quality of the final machine model. Although part of the related works [29] has also noticed the importance of this point, they have used simulation methods to approximately estimate model accuracy. However, due to the neural network's complexity and the edge learning system's collaborative training paradigm, their approach is not suitable for edge learning scenarios. In fact, only through real model training can we precisely obtain the correct model accuracy in edge learning.

IV. Problem Formulation and Analysis

A. Problem Formulation

In the edge learning, edge nodes and parameter server negotiate the corresponding pricing strategies to maximize their own utilities. Specifically, the objective for edge node i in the k-th training round can be formulated as a constrained optimization problem, i.e.,

$$\begin{aligned} \mathrm{OP_{i,k}}: \quad & \max_{\zeta_{i,k}} \ u_{i,k}, \\ & s.t. \ \ \zeta_{i,k} \in [\zeta_i^{min}, \zeta_i^{max}], \\ & \qquad u_{i,k} \geq \mu_i. \end{aligned}$$

For each edge node, in each training round, it receives the price from the parameter server, then determines the optimal computational strategy. The parameter server must pay enough bonus to the edge nodes, only by this can edge nodes participate in edge learning. In this paper, we assume that the edge node's utility $u_{i,k}$ must be greater than μ_i in each training round, where μ_i is the reserve utility of each edge node for joining edge learning.

As for the parameter server, its optimization problem can be defined as

$$\begin{aligned} \mathrm{OP_{PS}}: \quad & \max_{\mathrm{p_{i,k}}} \ u, \\ & s.t. \ \sum_{k=1}^{K}\sum_{i=1}^{N} p_{i,k}\zeta_{i,k} \leq \eta, \end{aligned}$$

where η is the total budget. The $\mathrm{OP_{PS}}$ indicates that the parameter server wants to get a perfect machine learning model while ensuring the shortest learning time under the constraint of a specified budget. We can also find that if the parameter server pays higher prices to the edge nodes in each training round, the edge nodes will perform the local training faster which results in shorter learning time. However, this leads to a rapid consumption of the budget and the edge learning will implement fewer rounds, finally, we will get a suboptimal machine learning model. On the contrary, if the parameter server pays lower prices to the edge nodes in each training round, we can get a better machine learning model, but the learning time will be longer. Therefore, we need to determine the optimal pricing strategy, which can not only get an optimal machine learning model, but also make the learning time shortest.

B. Optimal Strategy Analysis

In the following, we try to theoretically analyze the optimal strategy of both edge nodes and parameter server, so as to help us design the hierarchical deep reinforcement mechanism in the next section.

Optimal strategy for edge nodes.: as mentioned in section III, only when the parameter server pays enough bonus will the edge nodes participate in the edge learning. After receiving the price from the parameter server, each edge node decides its own optimal computational strategy to maximize its own utility. According to Eqn. (8), the first-order derivative of $u_{i,k}$ with respective to $\zeta_{i,k}$ is

$$\frac{\partial u_{i,k}}{\partial \zeta_{i,k}} = p_{i,k} - 2\sigma\alpha_i c_i d_i \zeta_{i,k}. \tag{10}$$

It is obvious that $u_{i,k}$ is a strictly convex function in $\zeta_{i,k}$. Therefore, if the feasible $p_{i,k}$ is given, it has only one optimal solution when $\frac{\partial u_{i,k}}{\partial \zeta_{i,k}} = 0$. And the optimal CPU cycle frequency $\zeta_{i,k}^*$ of edge node i in k-th round is

$$\zeta_{i,k}^* = \frac{p_{i,k}}{2\sigma\alpha_i c_i d_i}. \tag{11}$$

Based on Eqn. (6), we can deduce that the optimal computational time of edge node i in the k-th training round

is

$$t_{i,k}^{cmp,*} = \frac{2\alpha_i \sigma^2 {c_i}^2 {d_i}^2}{p_{i,k}}. \tag{12}$$

Optimal strategy for parameter server.: for the parameter server, the heterogeneous hardware information of edge nodes is unknown due to privacy protection in edge learning scenario. Hence, its optimal price allocation strategy cannot be directly derived through mathematical analysis. However, we can still analyze the pros and cons of the strategy through the feedback of the edge nodes' training time.

As illustrated in Fig. 1, the training time of each round is determined by the slowest edge node, resulting the wasted idle time (black part) for other edge nodes. Besides, according to (12), each edge node's training time is essentially determined by the price $p_{i,k}$ they received. Therefore, the sum of all edge nodes' idle time can reflect the superiority of current price allocation strategy, which is demonstrated as follows

Lemma 1. *For an optimal price allocation strategy of* $\mathrm{OP_{PS}}$*, in each training round* k*, it minimizes the sum of all edge nodes' idle time.*

Proof: First, suppose that there is two edge nodes i_1 and i_2 in the system and the price allocation strategy for two edge nodes is optimal for $\mathrm{OP_{PS}}$. Assume, without loss of generality, that the training time of two edge nodes satisfies $T_{i_1,k} < T_{i_2,k}$, thus the idle time of k-th round is $|T_{i_1,k} - T_{i_2,k}|$ and the training time of k-th round is $T_k = T_{i_2,k}$. According to (17), the reserve price for each edge node μ_i lead to their training time's upper bound $T_{i,k}^{upper}$. And each node's maximum CPU cycle frequency ζ_i^{max} lead to the lower bound of their training time $T_{i,k}^{lower}$, thus the training time of each edge node belongs to a adjustable range like $T_{i,k} \in [T_{i,k}^{lower}, T_{i,k}^{upper}]$. Obviously, as long as their training time is within their adjustable range, we can reduce the price of edge node i_1 and compensate this part of the price to edge node i_2 to generate new training time and satisfy $T_{i_1,k} < T_{i_1,k}^{new} < T_{i_2,k}^{new} < T_{i_2,k}$, which leads to $|T_{i_1,k}^{new} - T_{i_2,k}^{new}| < |T_{i_1,k} - T_{i_2,k}|$ and $T_k^{new} = T_{i_2,k}^{new} < T_k$. During this period, the sum of all edge nodes' idle time is minimized, and the training time of k-th round is also reduced accordingly. The extension to N edge nodes is similar, the price of edge nodes with shorter training time is always compensated to the last edge node. This compensation can continue until the following two situations: 1) $N-1$ edge nodes have reached the boundary of their adjustable ranges. 2) The training time for all edge nodes is the same.

In the new pricing strategy, the parameter server will receive the same models from all edge nodes with shorter training time (less idle time) under the same total price. This indicates that under the optimal pricing strategy, the sum of all edge nodes' idle time is minimal, which also corresponds to the shortest training time of k-th round. ■

V. Hierarchical Reinforcement Mechanism Design

Though the optimization problem for parameter server $\mathrm{OP_{PS}}$ has been created, there are two critical open challenges to solve it. **First**, in order to solve $\mathrm{OP_{PS}}$, we must plug (11)-(12) into $\mathrm{OP_{PS}}$. However, the (11)-(12) include the private information (e.g., c_i, d_i, α_i, μ_i, etc.) of edge nodes, they would not reveal their private information. **Second**, for edge learning, it is impossible to derive the mathematical model of learning performance $A(\boldsymbol{\omega}_K)$, due to the existence of neural networks. Both of these challenges make the incentive mechanism design for edge learning through mathematical analysis is extremely difficult. These are also the key factors for us to adopt the DRL-based approach, because **1)** it is a model-free approach, which is very suitable for incomplete information caused by privacy protection, **2)** it is an experience-driven approach that does not rely on any prior knowledge, thus the implicit system rules can be explored only through feedback.

However, the traditional naive DRL approach with only one agent cannot be applied to the current optimization goal $\mathrm{OP_{PS}}$ at all. As shown in Fig. 1 and illustrated in Sec.IV-B, how to determine the pricing strategy so as to minimize the wasted idle time is very important, and a good pricing strategy will make the edge nodes' training time as consistent as possible. Nevertheless, the current optimization goal $\mathrm{OP_{PS}}$ has the following new challenges due to the impact of budget constraints: **1) Long-term goal**: due to budget constraints, the sum of the price ($p_k = \sum_{i=1}^{N} p_{i,k}$) distributed by the server to all edge nodes in each round has a significant impact on the edge learning system's long-term performance. If the server consumes too much budget in the early stage, the entire training process may end prematurely with less training rounds, which also indicates an inferior edge learning performance. **2) Short-term goal**: once the current round's total price is determined, allocating it to each edge node to adjust their training time is essential. According to **Lemma 1**, a reasonable allocation strategy can make edge nodes try to complete training in the same time to minimize wasted idle time, which we call time consistency.

Thus, we propose a hierarchical deep reinforcement learning algorithm to simultaneously achieve these optimization goal in the incentive mechanism design. Different from only using one agent in traditional DRL approach, as shown in Fig. 2, we adopt a two-layer DRL agent design in the parameter server. In each round, *exterior agent* observes the state from the edge learning environment, then feds the state into its policy network and obtains an exterior action. The exterior action will assist in optimizing the long-term goal, which is also the state of the inner agent. By feeding the exterior action into the policy network of *inner agent*, we can

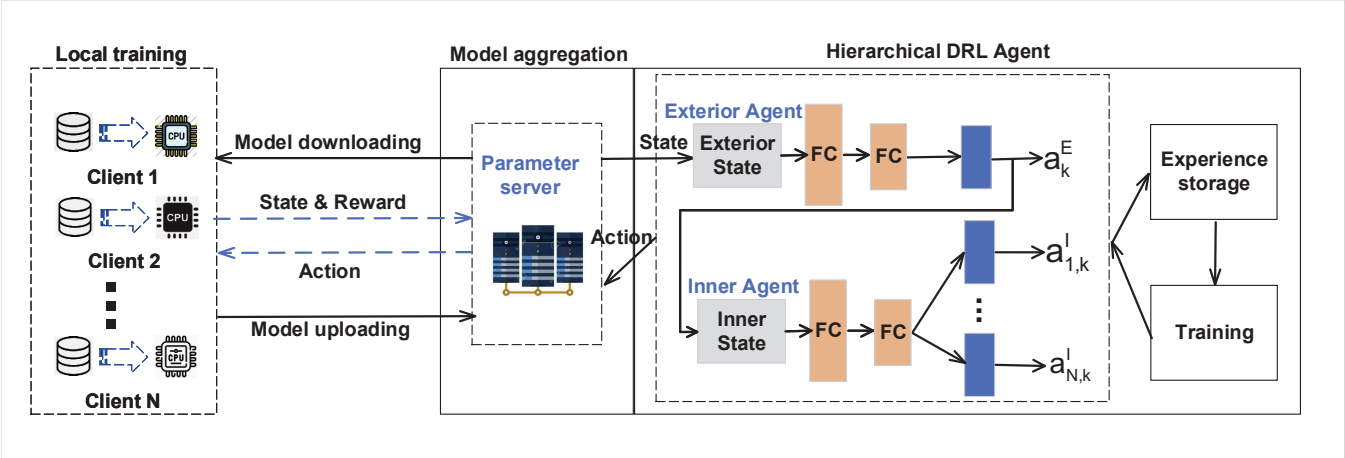


Figure 2. The architecture of Chiron.

obtain the inner action. The inner agent helps to optimize the edge nodes' training time as consistent as possible, so as to assist Chiron in making a better pricing strategy. Combining the exterior and inner actions together, the parameter server's action to the edge learning system can be determined. All the details about hierarchical model design will be presented in Sec.V-A

Furthermore, since the actions of exterior and inner agents are continuous (the price ranges are continuous), the policy-based algorithm is the most appropriate and we adopt the state-of-the-art DRL approach named Proximal Policy Optimization (PPO) as the hierarchical reinforcement learning algorithm for our Chiron.

A. Hierarchical DRL Model Design

Exterior State: the state of exterior agent in the k-th training round $\boldsymbol{s}_k^E$ consists of two parts, one is historical information of the previous L rounds including CPU cycle frequency profiles of edge nodes, pricing strategy profiles of parameter server, and the training time; the other is current information including the remaining budget and the index of current round. Formally, it is donated by $\boldsymbol{s}_k^E = \{\boldsymbol{\zeta}_{k-L}, \cdots, \boldsymbol{\zeta}_{k-1}, \boldsymbol{p}_{k-L}, \cdots, \boldsymbol{p}_{k-1}, \boldsymbol{T}_{k-L}, \cdots, \boldsymbol{T}_{k-L}, \eta, k\}$, where $\boldsymbol{\zeta}_k = \{\zeta_{1,k}, \cdots, \zeta_{N,k}\}$, $\boldsymbol{p}_k = \{p_{1,k}, \cdots, p_{N,k}\}$ and $\boldsymbol{T}_k = \{T_{1,k}, \cdots, T_{N,k}\}$. Particularly, in the initial stage when $k < L$, those nonexistent historical information is set to 0. The reason why we incorporate historical information into the state design is that we hope the agent can learn how its strategy changes affect the system performance of the edge learning.

Exterior Action: the action of exterior agent in k-th round is defined as $a_k^E = \{p_{total,k}\}$, where $p_{total,k}$ donates the total price for the whole edge nodes group. The exterior action can effectively control the amount of bonus that parameter server allocated in each round to save the budget for subsequent training, which is the long-term optimization target.

Inner State: the state of inner agent in k-th training round s_k^I is the total price strategy donated by $s_k^I = \{p_{total,k}\}$, which is also the action of exterior agent. The core idea of hierarchy is mainly embodied here, that is, the inner state is determined by the exterior action.

Inner Action: the inner action in k-th training round is the allocation proportion of the total price determined by the exterior agent among edge nodes. which is donated by $\boldsymbol{a}_k^I = \{pr_{1,k}, \cdots, pr_{N,k}\}$, where $pr_{i,k}$ is the proportion of edge node i in k-th round and $\sum_{i=1}^{N} pr_{i,k} = 1$. The inner action mainly focus on the short-term optimization target for time consistency. After taking $\boldsymbol{a}_k^I$, every edge node will receive its own price and the whole pricing strategy is the joint actions $\{a_k^E, \boldsymbol{a}_k^I\}$,

$$p_{i,k} = a_k^E * a_{i,k}^I. \tag{13}$$

Once edge nodes receive the pricing strategy, they will decide their computational strategies based on their own optimal strategy in Section IV to participate in edge learning. After all edge nodes have completed this training round and upload their model parameters to the parameter server, edge nodes will receive the bonus from the parameter server. Then, parameter server will record the training information $\{\boldsymbol{\zeta}_k, \boldsymbol{p}_k, \boldsymbol{T}_k, \eta, k+1\}$ of k-th round to update the exterior state $\boldsymbol{s}_{k+1}^E$ of next round. Especially, when the updated remaining budget is less than zero, all the training information in this round will not be recorded and the edge learning must be immediately stopped. After completing the machine learning model training in each round, parameter server will obtain the reward from edge learning environment, which consists of the exterior reward and inner reward, respectively.

Exterior Reward: according to Eqn. (9), the exterior reward

Algorithm 1 The hierarchical DRL agent training workflow

1: Initialize $\gamma, \boldsymbol{\theta}_a^E, \boldsymbol{\theta}_a^I, \boldsymbol{\theta}_c^E, \boldsymbol{\theta}_c^I, \boldsymbol{s}_1^E$
2: $\boldsymbol{\theta}_{a,old}^E = \boldsymbol{\theta}_a^E$, $\boldsymbol{\theta}_{a,old}^I = \boldsymbol{\theta}_a^I$
3: **for** episode in $1, 2, \cdots, Max\_Episode$ **do**
4: **for** k in $1, 2, \cdots,$ **do**
5: Derive action a_k^E by feeding $\boldsymbol{s}_k^E$ into the policy network $\boldsymbol{\theta}_{a,old}^E(a_k^E|\boldsymbol{s}_k^E)$
6: $s_k^I = a_k^E$
7: Derive action $\boldsymbol{a}_k^I$ by feeding s_k^I into the policy network $\boldsymbol{\theta}_{a,old}^I(\boldsymbol{a}_k^I|s_k^I)$
8: Calculate the pricing strategy based on Eqn. (13)
9: Edge nodes train the local machine learning models with the specified CPU cycle frequencies based on the pricing strategy
10: After the parameter server receiving all the new machine learning models from the edge nodes, it calculates the reward r_k^E and r_k^I by Eqn. (14) and (15), respectively
11: $\eta = \eta - \sum_{i=1}^{N} p_{i,k}\zeta_{i,k}$
12: Update the state of edge learning from $\boldsymbol{s}_k^E$ to $\boldsymbol{s}_{k+1}^E$
13: **if** $k > 1$ **then**
14: Update the inner state from s_{k-1}^I to s_k^I
15: Store transition $(s_{k-1}^I, \boldsymbol{a}_{k-1}^I, r_k^I, s_k^I)$ into $\mathcal{D}^I$
16: **end if**
17: **if** $\eta <= 0$ **then**
18: **for** $m = 1, 2, \cdots, M$ **do**
19: Update the inner critic network using SGD by minimizing loss $\sum (r_l^I + \gamma V_{\boldsymbol{\theta}_c^I}(s_{l+1}^I) - V_{\boldsymbol{\theta}_c^I}(s_l^I))^2$ with the experience data $\mathcal{D}^I$
20: Update the exterior critic network using SGD by minimizing loss $\sum (r_l^E + \gamma V_{\boldsymbol{\theta}_c^E}(\boldsymbol{s}_{l+1}^E) - V_{\boldsymbol{\theta}_c^E}(\boldsymbol{s}_l^E))^2$ with the experience data $\mathcal{D}^E$
21: Update the inner actor network by using SGD with the experience data $\mathcal{D}^I$
22: Update the exterior actor network by using SGD with the experience data $\mathcal{D}^E$
23: **end for**
24: $\boldsymbol{\theta}_{a,old}^E = \boldsymbol{\theta}_a^E$, $\boldsymbol{\theta}_{a,old}^I = \boldsymbol{\theta}_a^I$
25: Clear the buffer $\mathcal{D}^E$ and $\mathcal{D}^I$
26: Break
27: **end if**
28: Store transition $(\boldsymbol{s}_k^E, a_k^E, r_k^E, \boldsymbol{s}_{k+1}^E)$ into $\mathcal{D}^E$
29: **end for**
30: **end for**

of the k-th training round is defined as

$$r_k^E = \lambda(A(\boldsymbol{\omega}_k) - A(\boldsymbol{\omega}_{k-1})) - \lambda T_k, \tag{14}$$

which is the utility of parameter server in k-th round. This design allows the exterior agent to give actions based on the overall optimization goal, including learning metric $A(\boldsymbol{\omega})$ and system metric T_k .

Inner Reward: We set the inner reward as

$$r_k^I = -\sum_{i=1}^{N}(T_k - T_{i,k}), \tag{15}$$

which is the sum of idle time of all edge nodes. Maximizing this reward is equivalent to minimizing the sum of idle time. It can help the inner agent realize the training time consistency according to **Lemma 1**.

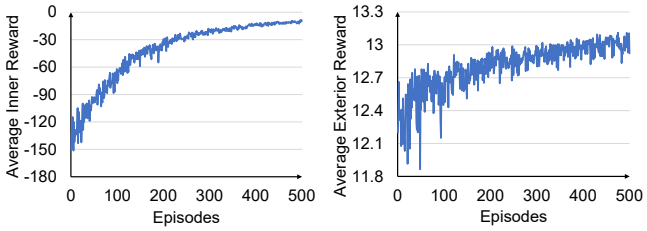
B. Training Workflow of Hierarchical DRL Agent

We train the hierarchical DRL agents by using the actor-critic approach, which well matches our context and has been successfully applied in many other fields. PPO is used to optimize the actor network. To leverage the PPO algorithm for DRL model training, we store all the transitions within an episode in experience replay buffer.

The two agents of hierarchical DRL both maintain their own experience replay buffer $\mathcal{D}^E$ and $\mathcal{D}^I$, their policy $\pi_{\boldsymbol{\theta}_a^E}(\boldsymbol{s}^E)$ and $\pi_{\boldsymbol{\theta}_a^I}(s^I)$ and their value function estimators $V_{\boldsymbol{\theta}_c^E}(\boldsymbol{s}^E)$ and $V_{\boldsymbol{\theta}_c^I}(s^I)$, where $\boldsymbol{\theta}_a^E$ and $\boldsymbol{\theta}_a^I$ are the actor network parameters, $\boldsymbol{\theta}_c^E$ and $\boldsymbol{\theta}_c^I$ are the critic network parameters. Through continuous interaction with the environment, agents gradually explore the optimal strategies for their respective learning objectives to maximize their respective accumulative rewards. The overall training workflow of the proposed hierarchical DRL agent is illustrated in **Algorithm 1**. The hierarchical DRL agent training workflow starts with randomly initialized parameters (Line 1-2). At the beginning of k-th training round, the parameter first decide the total price a_k^E by feeding $\boldsymbol{s}_k^E$ into the exterior policy network (Line 5). Then, the inner agent receives its state s_k^I and takes it as input of inner actor network $\pi_{\theta_a^I}$ to determine price allocation proportion $\boldsymbol{a}_k^I$ (Line 6). Then the pricing strategy of k-th training round is determined. The edge nodes will execute the local training based on the parameter server's pricing strategy. After receiving the updated machine learning models from the edge nodes, the parameter server calculate the inner reward and exterior reward, respectively (Line 10). At the same time, the parameter server needs to pay the bonus to the edge nodes. Note that if the parameter server's budget is negative, it will stop the edge learning task and update the hierarchical DRL agents with M times based on the experience data (Line 17-27).

VI. Performance Evaluation

To evaluate the performance of the proposed algorithm, we compare it against the existing state-of-the-art approaches. We first describe the experimental settings, followed by the experimental results and analysis.



(a) Inner DRL agent. (b) Exterior DRL agent.

Figure 3. Convergence of Chiron under MNIST.

A. Experimental Settings

According to the experimental settings in [9], [30], image classification is considered as the edge learning task. Thus, we adopt the edge learning experiment framework [1] proposed by Google [4] and similarly conducted extensive experiments on real-world datasets (MNIST, Fashion-MNIST, and CIFAR-10). Different neural networks and parameter settings are applied to the image classification task of each dataset as following.

- MNIST & Fashion-MNIST: We train a CNN model with 5×5 convolution layers. The first layer has 10 output channels and the second has 20, and each layer followed by 2×2 max pooling. The model has a total of 21,840 trainable parameters [4]. On each edge node, the batch size is 10 and the epoch number is 5.
- CIFAR-10: We train a CNN model named LeNet with 5×5 convolution layers. The first layer has 6 output channels and the second has 16, and each layer followed by 2×2 max pooling. The model has a total of 62,006 trainable parameters [4]. On each edge node, the batch size is 10 and the epoch number is 5.

For the local computational model, we set the number of CPU cycles to execute one bit c_i is 20 cycles/bit, the maximal CPU cycle frequency f_i^{max} for each edge node is randomly distributed within $1.0 \sim 2.0$ GHz, the communication time of each edge node is randomly distributed within $10 \sim 20$ seconds, the effective capacitance coefficient is 2×10^{-28}.

For the Hierarchical DRL related parameters of Chiron, since DRL is an experience-driven learning approach, we set the episode number $E = 500$. The step number of each episode is dynamic due to the budget constraint, once the budget is exhausted, the current episode will stop immediately. Furthermore, the learning rate of actor and critic network is $lr_a = lr_c = 0.00003$, which will decays by 95% every 20 episodes. The update batch of agent is equal to the step number of each episode. In reward design, the reward discount factor is $\gamma = 0.95$, and the preference adjustment coefficient is $\lambda = 2000$.

[1]https://github.com/AshwinRJ/edge-Learning-PyTorch.git

We compare our proposed approach with one state-of-the-art approach and one baseline.

- DRL-based [8]: In this work, the authors design the incentive mechanism based on edge nodes' energy consumption and learning time. They adopt the standard PPO algorithms as DRL approach, but only derive the optimal solution of single round.
- Greedy: At the beginning, the agent randomly generates a series of actions to form the replay buffer. Then it will greedily choose the action with maximum reward from the replay buffer with a high probability, or explore new actions with a small probability.

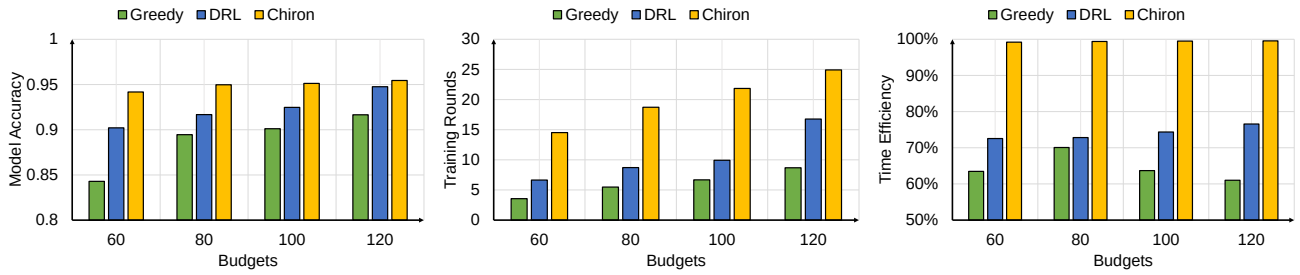
B. Experimental Performance Analysis

We first conduct a small scale experiment with five edge nodes, and the training data is randomly distributed among the edge nodes. Fig. 3 plots the training progress of the DRL agent on MNIST. An episode starts at the initialization of a edge learning task and ends when the budget η of the parameter server has run out. We observe that the average reward of each episode increases over time, it indicates that Chiron is learning a better and better policy that can output the near-optimal pricing strategy. This is because the interior agent and exterior agent cooperate with each other and gradually learn how to allocate the best pricing strategy.

Fig. 4 shows performance under MNIST dataset when varying the budget constraints. Fig. 4(a) shows the final global model accuracy of edge learning under different budget constraints. Since the combination of performance with optimization goal, Chiron can guarantee better model quality and outperform all the other baselines. Furthermore, we can observe that as the budget increases, the accuracy gap between Chiron and other baselines is gradually decreasing. This is because the accuracy improvement of MNIST has a marginal effect. As the number of training rounds increases, the accuracy improvement becomes smaller each round. Therefore, when the budget is sufficient, the performance of the other two will gradually approach Chiron.

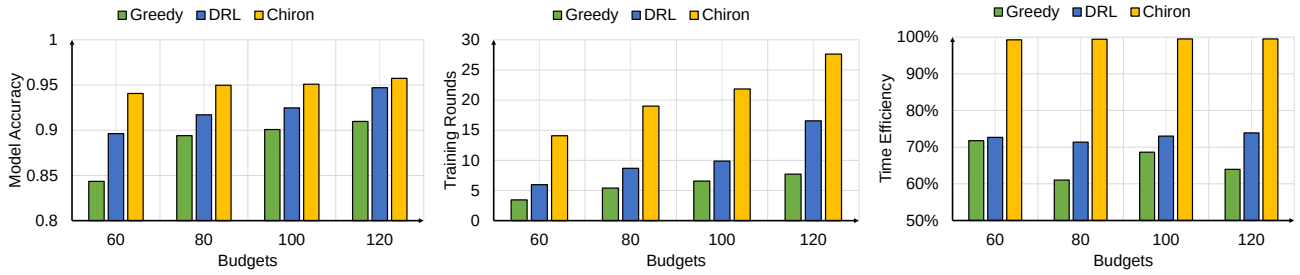
Fig. 4(b) shows the superiority of Chiron in the long-term optimization goal. with the same budget, Chiron can train the machine learning model with more rounds. For example, Chiron can train the machine learning model with 21 rounds when the budget is 100, while greedy and DRL-based can only train the machine learning model with only 6 and 9, respectively. This is because Chiron can determine the near-optimal pricing strategy in each round, thus the parameter server only needs to pay less bonus to the edge nodes in each training round, which allows Chiron to save budget for subsequent training in each round.

Fig. 4(c) illustrates the superiority of Chiron in short-term optimization goals. Recall the definition of idle time in Fig.1 and the optimization goal of time consistency in Sec.V, we



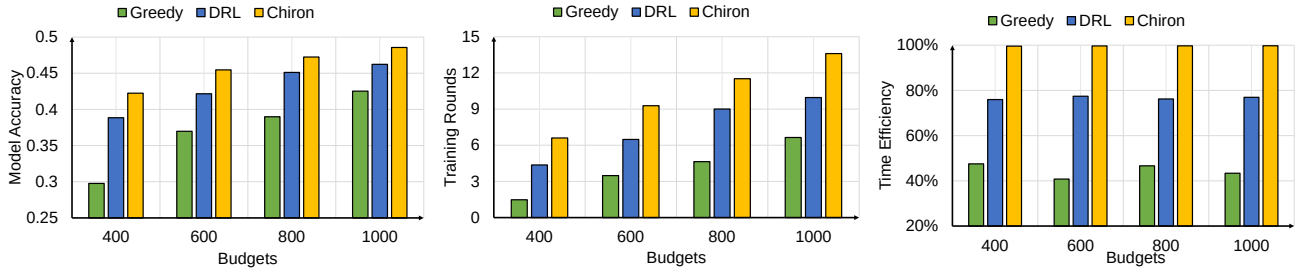
(a) Global model accuracy. (b) Training rounds. (c) Time efficiency.

Figure 4. Performance under MNIST when varying the total budgets.



(a) Global model accuracy. (b) Training rounds. (c) Time efficiency.

Figure 5. Performance under Fashion-MNIST when varying the total budgets.



(a) Global model accuracy. (b) Training rounds. (c) Time efficiency.

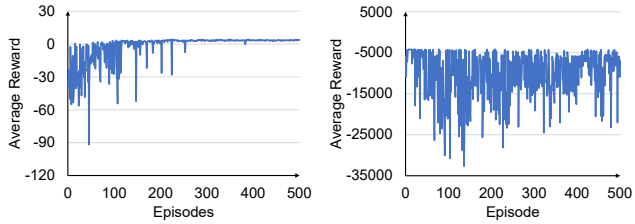
Figure 6. Performance under CIFAR-10 when varying the total budgets.

propose a metric of time consistency as,

$$\text{Time efficiency} = \frac{\sum_{i=1}^{N} T_{i,k}}{N \cdot T_k}, \tag{16}$$

which means that time efficiency represents the percentage of edge node's effective training time in total time of k-th round. In other words, the less idle time wasted in the current round, the higher the time efficiency. From Fig. 4(c), we can find that Chiron has achieved nearly 100% time efficiency under all budgets. It means that all edge nodes have completed training simultaneously and reached the time consistency required by the short-term optimization goal.

Fig. 5 and Fig. 6 show the performance under Fashion-MNIST and CIFAR10 datasets. Though the edge learning tasks are different, yet we can still find that our proposed approach Chiron obtains the best performance as compared with the other two approaches. By the way, due to the complexity of the CIFAR-10, processing the same number of samples requires more computing resources, which leads



(a) Convergence of Chiron's exterior agent. (b) Convergence of DRL-based approach.

Figure 7. Convergence of Chiron and DRL-based under MNIST.

Table I
PERFORMANCE OF CHIRON UNDER MNIST WITH 100 EDGE NODES

η	Accuracy	Rounds	Time Efficiency
140	0.916	16	71.3%
220	0.929	23	72.2%
300	0.938	31	72.7%
380	0.943	34	73.4%

to different budget constraints in the experiment.

In order to evaluate the scalability of the proposed Chiron approach. We conduct experiments with 100 edge nodes under MNIST. In Fig. 7(a), we can find that Chiron can learn the near-optimal pricing strategy after about 300 episodes, which is the same as the small scale edge learning system. This indicates that Chiron can work under a relatively large-scale scenario. In Fig. 7(b), we can find that the DRL-based approach cannot converge. It indicates that the DRL-based approach is difficult to generalize to large-scale edge learning scenarios. Table I shows the performance of Chiron with 100 edge nodes under MNIST. We can find that when the budget is 300, the time efficiency is 72.7%, which means our mechanism can still achieve relatively high optimization level under large-scale edge nodes environment.

VII. RELATED WORK

Edge Learning allows machine learning models to be trained on mobile devices in a distributed fashion without revealing user privacy. Exiting work on edge learning mainly focuses on improve the learning efficiency. Only a few work focuses on design the incentive mechanism.

There is a few work focuses on incentive mechanism design for edge learning. Weng et al. [18] present a distributed deep learning framework called DeepChain, which combines with a value-driven incentive mechanism to force edge nodes to participate in training in good faith. Pandey et al. [17] construct a utility maximization problem under a novel crowdsourcing framework to promote communication efficiency during parameters exchange, and the Stackelberg game is used to formulate the incentive mechanism. Zeng et al. [19] propose an auction-based incentive mechanism Fmore for edge learning system, which can encourage the participation of more high-quality edge nodes with low cost. They consider the edge node resource constraint, including local data, computation capability, bandwidth, CPU cycle, etc., and use game theory to derive optimal strategies for the edge participators. In [8], [31], Zhan et al. design an incentive mechanism based on DRL approach to motivate the edge nodes to participate in the edge learning which can achieve an efficient edge learning.

However, these work neglects the critical property of edge learning algorithm. 1) They only consider the myopia optimization of single round, which results in the lack of long-term optimization. 2) They do not take the characteristics of different learning algorithms performance into the mechanism design, which cannot guarantee the quality of the final learning model. Based on this observation, we are the first to design the incentive mechanism for edge learning by taking long-term utility and learning algorithm performance (i.e. model accuracy) into consideration. Note that although Zhan et al. [8] also adopt the DRL-based approach for learning optimal strategy, they only apply it to a single-round optimization and ignore the long-term sustainability.

VIII. CONCLUSION

This paper is motivated by how to incentivize the edge nodes to participate in the edge learning to train the machine learning model efficiently. As the edge nodes with the high learning speed may be dragged down by the slow edge nodes, blindly increasing the bonus to the edge nodes to make them execute local training fast not only cannot accelerate the edge learning convergence speed, but also will reduce the quality of the final machine learning model. In this paper, we propose to improve the system performance of edge learning by carefully controlling the pricing strategy. Due to the complexity of edge learning algorithm, the unawareness of the edge nodes' private information, it prompts us to use the learning-based approach to design the incentive mechanism. Thus, we propose Chiron, a hierarchical deep reinforcement learning approach to design the incentive mechanism. The final experiments based on three different datasets further demonstrate the superiority of our approach as compared with the state-of-the-art solutions.

ACKNOWLEDGEMENT

This research was supported by the funding from Hong Kong RGC Research Impact Fund (RIF) with the Project No. R5060-19 and R5034-18, General Research Fund (GRF) with the Project No. 152221/19E and 15220320/20E, Collaborative Research Fund (CRF) with the Project No. C5026-18G, the National Natural Science Foundation of China (61872310), and Shenzhen Science and Technology Innovation Commission (R2020A045).

REFERENCES

[1] T. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma, "Pcanet: A simple deep learning baseline for image classification?" *IEEE Transactions on Image Process.*, vol. 24, no. 12, pp. 5017–5032, 2015.

[2] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. P. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[3] E. Gawehn, J. A. Hiss, and G. Schneider, "Deep learning in drug discovery," *Molecular informatics*, vol. 35, no. 1, pp. 3–14, 2016.

[4] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. of AISTATS*, 2017, pp. 1273–1282.

[5] H. Wang, Z. Kaplan, D. Niu, and B. Li, "Optimizing federated learning on non-iid data with reinforcement learning," in *Proc. of IEEE INFOCOM*, 2020, pp. 1698–1707.

[6] Y. Zhan, P. Li, and S. Guo, "Experimence-driven computational resource allocation of federated learning by deep reinforcement learning," in *Proc. of IEEE IPDPS*, 2020, pp. 1–10.

[7] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "When edge meets learning: Adaptive control for resource-constrained distributed machine learning," in *Proc. of IEEE INFOCOM*, 2018, pp. 63–71.

[8] Y. Zhan and J. Zhang, "An incentive mechanism for efficient edge learning by deep reinforcement learning approach," in *Proc. of IEEE INFOCOM*, 2020, pp. 1–10.

[9] N. H. Tran, W. Bao, A. Zomaya, and C. S. Hong, "Federated learning over wireless networks: Optimization model design and analysis," in *Proc. of IEEE INFOCOM*, 2019, pp. 1387–1395.

[10] H. Wang, Z. Qu, S. Guo, X. Gao, R. Li, and B. Ye, "Intermittent pulling with local compensation for communication-efficient distributed learning," *IEEE Transactions on Emerging Topics in Computing*, 2020, doi:10.1109/TETC.2020.3043300.

[11] H. Wang, S. Guo, B. Tang, R. Li, and C. Li, "Heterogeneity-aware gradient coding for straggler tolerance," in *Proc. of IEEE ICDCS*, 2019, pp. 555–564.

[12] H. Wang, S. Guo, and R. Li, "Osp: Overlapping computation and communication in parameter server for fast machine learning," in *Proc. of ACM ICPP*, 2019, pp. 1–10.

[13] Y. Zhan, P. Li, W. Leijie, and S. Guo, "L4l: Experience-driven computational resource control in federated learning," *IEEE Transactions on Computers*, 2021, doi:10.1109/TC.2021.3068219.

[14] K. Powell, "Nvidia clara federated learning to deliver ai to hospitals while protecting patient data," https://blogs.nvidia.com/blog/2019/12/01/clara-federated-learning/, accessed Dec. 1, 2019.

[15] Y. Chen, L. Su, and J. Xu, "Distributed statistical machine learning in adversarial settings: Byzantine gradient descent," in *Proc. of ACM SIGMETRICS*, vol. 1, no. 2, 2017, pp. 1–25.

[16] W. Liu, L. Chen, Y. Chen, and W. Zhang, "Accelerating federated learning via momentum gradient descent," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 8, pp. 1754–1766, 2020.

[17] S. R. Pandey, N. H. Tran, M. Bennis, Y. K. Tun, A. Manzoor, and C. S. Hong, "A crowdsourcing framework for on-device federated learning," *IEEE Transactions on Wireless Communications*, vol. 19, no. 5, pp. 3241–3256, 2020.

[18] J. Weng, J. Weng, J. Zhang, M. Li, Y. Zhang, and W. Luo, "Deepchain: Auditable and privacy-preserving deep learning with blockchain-based incentive," *IEEE Transactions on Dependable and Secure Computing*, 2019, doi:10.1109/TDSC.2019.2952332.

[19] R. Zeng, S. Zhang, J. Wang, and X. Chu, "Fmore: An incentive scheme of multi-dimensional auction for federated learning in mec," in *Proc. of IEEE ICDCS*, 2020, doi:10.1109/ICDCS47774.2020.00094.

[20] N. Ding, Z. Fang, and J. Huang, "Optimal contract design for efficient federated learning with multi-dimensional private information," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 1, pp. 186–200, 2020.

[21] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proc. of ACM SIGCOMM*, 2019, pp. 270–288.

[22] L. Chen, J. Lingys, K. Chen, and F. Liu, "Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *Proc. of ACM SIGCOMM*, 2018, pp. 191–205.

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

[24] F. Wang, C. Zhang, J. Liu, Y. Zhu, H. Pang, L. Sun *et al.*, "Intelligent edge-assisted crowdcast with deep reinforcement learning for personalized qoe," in *Proc. of IEEE INFOCOM*, 2019, pp. 910–918.

[25] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[26] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991.

[27] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. of NIPS*, 2000, pp. 1057–1063.

[28] C. T. Dinh, N. H. Tran, M. N. Nguyen, C. S. Hong, W. Bao, A. Y. Zomaya, and V. Gramoli, "Federated learning over wireless networks: convergence analysis and resource allocation," *IEEE/ACM Transactions on Networking*, 2020, doi:10.1109/TNET.2020.3035770.

[29] Y. Jiao, P. Wang, D. Niyato, B. Lin, and D. I. Kim, "Toward an automated auction framework for wireless federated learning services market," *IEEE Transactions on Mobile Computing*, 2020, doi:10.1109/TMC.2020.2994639.

[30] L. Liu, J. Zhang, S. H. Song, and K. B. Letaief, "Client-edge-cloud hierarchical federated learning," in *Proc. of IEEE ICC*, 2020, pp. 1–6.

[31] Y. Zhan, P. Li, Z. Qu, D. Zeng, and S. Guo, "A learning-based incentive mechanism for federated learning," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6360–6368, 2020.