Dynamic and Fast Convergence for Federated Learning via Optimized Hyperparameters

Xinlei Yu, Yijing Lin*, Zhipeng Gao*, Hongyang Du, Dusit Niyato, *Fellow, IEEE*

***Abstract*—Federated Learning (FL) is a privacy-preserving computing paradigm that enables participants to collaboratively train a global model without exchanging their raw personal data. Due to frequent communication and data heterogeneity of devices with unique local data distributions, FL faces a significant issue with slow convergence speed. To achieve fast convergence, existing methods adjust hyperparameters in FL to reduce the volume of model updates, the number of participating devices, and local iterations. However, most focus on only part of the hyperparameters and primarily rely on analytical optimization. A more integrated and dynamic coordination of all hyperparameters is needed. To address this issue, we first propose an efficient FL framework enabled by rand-m sparsification and stochastic quantization methods. For this framework, we conduct a rigorous theoretical analysis to explore the trade-offs among quantization level, sparsification level, device participation, and local iteration. To improve convergence speed, we also design a Deep Reinforcement Learning (DRL)-based strategy to dynamically coordinate these hyperparameters. Experimental results show that our method can improve convergence speed by at least 8% compared to the existing approaches.**

***Index Terms*—Federated learning, Quantization, Sparsification, Deep Reinforcement Learning.**

I. Introduction

Federated Learning (FL) is a popular distributed machine learning method that can protect privacy without sharing raw data [1] [2] [3] [4] [5] [6]. In general, a typical FL system consists of multiple devices connected to the central server, following a three-step workflow. Firstly, the central server selects participating devices, synchronizes their local models with the latest global model and then participating devices compute model updates using their training data. Secondly, participating devices upload their model updates to the central server. Thirdly, the central server aggregates these updates using an aggregation algorithm. FL tasks repeat these steps until the global model convergence.

Although FL has the advantage of privacy protection, its convergence speed is often limited by communication time and device heterogeneity [7]. During the FL process, the frequent and large exchange of model updates between devices and the central server results in significant communication time. Gradient quantization and sparsification reduce communication time by compressing the size of model updates [8]. Gradient quantization reduces the size of transmitted model updates by converting model gradients from high-precision to low-precision formats [9]. Gradient sparsification further reduces the communication cost by transmitting only important elements and ignoring less significant ones [10]. The quantization level and sparsification level together determine the degree of model update compression.

Due to differences in local computing and communication resources on devices, applying uniform training hyperparameters (local iterations and device participation) and compression hyperparameters (quantization and sparsification levels) can cause synchronization delays on some devices, slowing down global model convergence [11]. Moreover, these hyperparameters are closely related to the convergence accuracy of the global model. If they are not set properly, the accuracy of the model may drop significantly [12].

An instinct method that adjusts hyperparameters in FL has been devised to solve the aforementioned challenges. Specifically, it modifies quantization level or sparsification level to diminish the communication time between servers and devices [8] [12] [13], tunes the number of local iterations (i.e., the number of updates performed on local data during each communication round, which differ from epochs that represent full passes through the dataset) [14] [15] [16], or chooses a subset of devices [16] [17] [18] to reduce the impact of device heterogeneity on convergence speed. For example, G. Yan *et al.* dynamically adjusted the quantization level and sparsification level based on the norm of the gradients, communication budget, and remaining iterations, improving accuracy under communication budget constraints [8]. Y. Liu *et al.* decomposed the research problem into multiple subproblems and iteratively solved for the number of local iterations to guarantee the convergence of the global model [15]. H. Zhang *et al.* utilized a priority-weight-based optimization model to automatically adjust the number of devices and training iterations in FL, thereby reducing system overhead [16].

However, the previous studies do not solve the following issues. First, previous works focus on coordinating only part of training hyperparameters and compression hyperparameters,

Corresponding authors: Yijing Lin and Zhipeng Gao. Xinlei Yu, Yijing Lin, and Zhipeng Gao are with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China (e-mail: {yuxinlei518, yjlin, gaozhipeng}@bupt.edu.cn). Hongyang Du is with the Department of Electrical and Electronic Engineering, University of Hong Kong, Pok Fu Lam, Hong Kong SAR, China (email: duhy@eee.hku.hk). Dusit Niyato is with the College of Computing and Data Science, Nanyang Technological University, Singapore (e-mail: DNIYATO@ntu.edu.sg).

This work is supported by the General Program of National Natural Science Foundation of China (62372050) and the Beijing Natural Science Foundation of China (4232029). This work is supported by the Fellowship of China National Postdoctoral Program for Innovative Talents under Grant BX20240045. This work is supported by the National Research Foundation, Singapore, and Infocomm Media Development Authority under its Future Communications Research & Development Programme, Defence Science Organisation (DSO) National Laboratories under the AI Singapore Programme (FCP-NTU-RG-2022-010 and FCP-ASTAR-TG-2022-003), Singapore Ministry of Education (MOE) Tier 1 (RG87/22), the NTU Centre for Computational Technologies in Finance (NTU-CCTF), and Seitee Pte Ltd.

which can lead to suboptimal performance because the full range of hyperparameters affecting convergence speed and accuracy are not considered holistically. Second, there is no comprehensive theoretical analysis that simultaneously coordinates all these hyperparameters. Third, most existing methods rely on analytical optimization approaches, where optimization problems are solved through predefined algorithms that do not adapt well to changes during training. Deep Reinforcement Learning (DRL), which optimizes decisions through interaction with the environment, can adaptively adjust strategies to overcome these limitations [19]. Nevertheless, current DRL optimization methods typically focus on a subset of hyperparameters and have not yet considered the joint optimization of both all training and compression hyperparameters.

Therefore, in this paper, we propose a fast convergence method for FL by adjusting training hyperparameters simultaneously. Specifically, we first integrate rand-m sparsification and stochastic quantization in FL to cover all compression avenues, ensuring unbiased compression for stable convergence. Then, we conduct a theoretical analysis to derive a convergence bound. Based on the convergence bound, we formulate a dynamic strategy based on DRL to adjust these hyperparameters in heterogeneous scenarios. The main contributions of this paper are summarized as follows:

- We propose an efficient framework for FL enabled by the integration of rand-m sparsification and stochastic quantization methods and formulate an optimization problem to achieve fast convergence while maintaining model accuracy.
- We conduct a theoretical analysis for the proposed framework and derive a convergence bound to minimize communication time while maximizing accuracy.
- Considering the convergence bound, we utilize DRL to find optimal strategies to simultaneously adjust quantization level, sparsification level, the number of local iterations, and device participation.
- Experimental results show that our method reduces the convergence time by at least 8% compared with the state-of-the-art method Fedeco [15].

II. RELATED WORK

In this section, we first review the concepts of quantization, sparsification, and deep reinforcement learning. Then, we discuss existing work related to hyperparameter adjustment in FL.

A. Quantization and Sparsification for FL

Gradient quantization and gradient sparsification were initially applied in distributed machine learning to reduce communication costs and time. These techniques were later extended to FL and became one of the most widely used methods for reducing communication overhead. Stochastic quantization and rand-m sparsification are the most commonly used gradient quantization and sparsification methods. Stochastic quantization reduces the amount of transmitted data by probabilistically rounding model updates to lower precision, ensuring unbiased estimates while balancing model accuracy and communication efficiency [20]. Rand-m sparsification randomly selects important elements for transmission while ignoring less significant ones [21]. Concurrently, researchers also explored different aspects of these methods. For example, SU. Stich *et al.* introduced an error feedback mechanism to improve compressed gradients, ensuring that accumulated errors over multiple communication rounds are mitigated [22]. C. Li *et al.* proposed a communication-efficient FL algorithm that leverages compressed sensing to efficiently transmit compressed model updates, enabling rapid and precise model convergence [23].

B. Deep Reinforcement Learning for FL

DRL is a machine learning paradigm where an agent learns optimal strategies through interaction with their environment, using a reward-based feedback system. In FL, DRL is often used to address challenges related to the dynamic nature of training environments, such as device selection and resource allocation. For instance, F. Zheng *et al.* proposed a DRL-based client selection algorithm that balances client resources and data heterogeneity, accelerating model convergence and enhancing performance [24]. T. Zhao *et al.* developed a DRL framework to allocate communication and computational resources, significantly reducing overhead while maintaining accuracy in FL [25].

C. Hyperparameter Adjustment in FL

In FL, achieving fast convergence in the presence of device heterogeneity is a challenge. Proper hyperparameter adjustment is key to reducing communication time and adapting to heterogeneous environments, thereby improving convergence speed while ensuring accuracy. Existing methods for hyperparameter adjustment in FL can be broadly categorized into two types: adjusting training hyperparameters and compression hyperparameters.

Some studies focus on the optimization of training hyperparameters. W. Shi *et al.* used a heuristic greedy strategy to select devices minimizing model update time, balancing learning efficiency and latency [18]. B. Luo *et al.* optimized device selection and local iterations using control variable analysis to ensure convergence while minimizing costs [26]. W. Sun *et al.* adaptively adjusted local iterations based on a Lyapunov dynamic deficit queue, improving performance under resource constraints [27]. Y. Liu *et al.* adaptively set local iterations by decomposing the problem into sub-problems to ensure model convergence [15]. Q. Chen *et al.* proposed a DRL-assisted system where heterogeneous devices adjust local iterations via locally deployed DRL models [11]. M. Kundroo *et al.* dynamically adjusted training iterations by monitoring client loss values to improve convergence and performance [28]. H. Zhang *et al.* used a priority-weight optimization model to automatically adjust device selection and iterations, reducing system overhead [16].

Similarly, compression hyperparameters are also being investigated. MK. Nori *et al.* leveraged the interdependency between local updates and gradient compression to jointly and dynamically adjust local iterations and sparsification levels,

TABLE I: Key notions

Symbol	Semantics
$\mathcal{I}$	set of devices $\{1, ..., i, ..., n\}$
$\mathcal{R}_k$	devices participating in k-th communication round
r_k	number of participating devices in k-th communication round
$\mathcal{D}^{(i)}$	data distribution on device i
$\mathcal{S}^{(i)}$	set of samples on device i
ω_k	global model in k-th communication round
$\omega_k^{(i)}$	local model derived by local training ω_{k-1} on device i
$\omega^\star$	optimal model of the global objective function
$\delta_k^{(i)}$	number of local iteration in k-th communication round on device i
η	learning rate of FL
$\mathcal{B}$	batch size of FL
e	epoch size of FL
$c_u^{(i)}$	available bandwidths for uploading on device i
$c_d^{(i)}$	available bandwidths for downloading on device i
$v^{(i)}$	time required to compute a single batch on device i
$f^{(i)}(\cdot)$	local objective function of FL
$F(\cdot)$	global objective function of FL
$\ell(\cdot)$	loss function of FL
$C(\cdot)$	compression operation of FL
$H_b(\cdot)$	quantization use b bits
$H_m(\cdot)$	sparsification retains m elements
$g^{(i)}(\cdot)$	gradient on device i

accelerating the convergence of FL and improving model accuracy [12]. G. Yan *et al.* dynamically adjusted the quantization level and sparsification level based on the norm of the gradients, communication budget, and remaining iterations, improving accuracy under communication budget constraints [8].

While these methods adjust certain hyperparameters, they overlook the joint optimization of quantization level, sparsification level, the number of local iterations, and device participation, which hinders their efficiency. There is also a lack of a theoretical framework to guide the dynamic optimization of these hyperparameters, which is crucial for developing an efficient FL system.

III. System Model

This section outlines the FL framework, including the interaction between the device and the central server, the communication and computation time for FL, and the use of sparsification and quantization to reduce communication time.

We consider a FL system comprising a set of n heterogeneous devices, denoted as $\mathcal{I} = \{1, ..., i, ..., n\}$, and a central server. Each device i possesses a local data distribution $\mathcal{D}^{(i)}$, which contains individual training samples (x, y). The objective function $f^{(i)}(\omega)$ of each device i is defined by the expected loss over its data distribution as

$$f^{(i)}(\omega) = \mathbb{E}_{(x,y)\sim\mathcal{D}^{(i)}}[\ell(\omega, (x, y))], \tag{1}$$

where $\mathbb{E}[\cdot]$ denotes the expectation, ω represents the model parameters, and ℓ is the loss function. Furthermore, we can express $f^{(i)}(\omega)$ as an average loss over a set of samples $\mathcal{S}^{(i)}$ drawn from $\mathcal{D}^{(i)}$, which is denotes as

$$f^{(i)}(\omega) = \frac{1}{|\mathcal{S}^{(i)}|} \sum_{(x,y)\in\mathcal{S}^{(i)}} \ell(\omega, (x, y)), \tag{2}$$

where $|\mathcal{S}^{(i)}|$ indicates the size of samples on device i. The global objective function of the FL system, which aims to aggregate learning objectives across all devices, is formulated as

$$F(\omega) = \frac{\sum_{i=1}^{n}\sum_{(x,y)\in\mathcal{S}^{(i)}} \ell(\omega, (x, y))}{\sum_{i=1}^{n} |\mathcal{S}^{(i)}|} = \sum_{i=1}^{n} \frac{|\mathcal{S}^{(i)}|}{|\mathcal{S}|} f^{(i)}(\omega), \tag{3}$$

where $|\mathcal{S}|$ denotes the total number of samples across all devices, denoted as $|\mathcal{S}| = \sum_{i=1}^{n} |\mathcal{S}^{(i)}|$. The goal of FL is to collaboratively train an optimal global model $\omega^\star$ that minimizes the global objective function $F(\omega)$. Table I summarizes the key notations used in this paper.

A. Workflow of Federated Learning

In the k-th communication round of the FL process, the central server selects a subset of devices $\mathcal{R}_k \subseteq \mathcal{I}$, where $|\mathcal{R}_k| = r_k$, to participate in training. r_k is the number of participating devices at each communication round. Subsequently, each device downloads the latest global model ω_k from the central server. Given the Mini-Batch size $|\mathcal{B}_k^{(i)}|$ and the epoch size $e_k^{(i)}$ for local training, the total number of local iterations is,

$$\delta_k^{(i)} = \left\lceil \frac{|\mathcal{S}^{(i)}|}{|\mathcal{B}_k^{(i)}|} \right\rceil e_k^{(i)}. \tag{4}$$

All participating devices fix the number of local iterations to δ_k and each participating device updates the local model by conducting δ_k iterations. The update process at each iteration is defined as

$$\begin{cases} \omega_{k,t}^{(i)} = \omega_{k,t-1}^{(i)} - \eta g^{(i)}(\omega_{k,t-1}^{(i)}, \mathcal{B}_{k,t-1}^{(i)}), & 1 < t < \delta_k, \\ \omega_{k,1}^{(i)} = \omega_k, & t = 1, \\ \omega_{k,\delta_k}^{(i)} = \omega_{k+1}^{(i)}, & t = \delta_k, \end{cases} \tag{5}$$

where t is the current local iteration, $g^{(i)}(\omega_{k,t-1}^{(i)}, \mathcal{B}_{k,t-1}^{(i)})$ represents the gradient computed by a batch of local samples $\mathcal{B}_{k,t-1}^{(i)}$ with the learning rate η.

After all participating devices complete local training, they compress their model updates and transmit them back to the central server. The compression operation is described in Section III-B. The central server then aggregates these compressed updates to obtain the new global model for the next communication round. The aggregation process can be described as

$$\begin{aligned} \omega_{k+1} &= \omega_k + \frac{1}{r_k}\sum_{i=1}^{r_k} C(\omega_{k+1}^{(i)} - \omega_k) \\ &= \omega_k + \frac{1}{r_k}\sum_{i=1}^{r_k} C(\Delta\omega_{k+1}^{(i)}), \end{aligned} \tag{6}$$

where $C(\cdot)$ denotes the compression operation applied to the model updates, and $\frac{1}{r_k}\sum_{i=1}^{r_k} C(\Delta\omega_{k+1}^{(i)})$ represents the average of the compressed updates from r_k devices that participate in the k-th communication round.

B. Sparsification and Quantization

To reduce the volume of model update transmission in FL, our method incorporates compression techniques that efficiently compress model updates when sending them to the server. We utilize unbiased compressors due to their ability to maintain robustness. Thus, our framework integrates rand-m sparsification and stochastic quantization to compress model updates. The compression operator $C(\Delta\omega)$ in our method is formulated as $H_b(H_m(\Delta\omega))$, where $H_b(\cdot)$ represents the quantization operation that compresses model updates to b bits, and $H_m(\cdot)$ denotes the sparsification process that retains only m significant elements of the model update $\Delta\omega$.

Rand-m sparsification. $H_m(\Delta\omega)$ is utilized to the model update $\Delta\omega$ by randomly selecting $m_k^{(i)}$ elements in $\Delta\omega$, amplifying these selected elements by $\frac{d}{m_k^{(i)}}$ (where d is the total number of elements in $\Delta\omega$) times, and setting the remaining elements to zero. This amplification ensures that the expected value of the model update remains unbiased, even though only a subset of elements is transmitted. The sparsification level $m_k^{(i)}$ of device i in the k-th communication round can dynamically vary based on network conditions and resource constraints.

Stochastic quantization. After using the sparsification operation to the model updates, we obtain a sparsified vector ($H_m(\Delta\omega) = \Delta\hat{\omega}$). Subsequently, a stochastic quantization process is executed on each element of this sparsified vector. Specifically, the quantization of the j-th element of the sparsified vector is executed as follows:

$$H_b(\Delta\hat{\omega}_{(j)}) = ||\Delta\hat{\omega}||_2 \cdot \text{sgn}(\Delta\hat{\omega}_{(j)}) \cdot \zeta(\Delta\hat{\omega}_{(j)}, z), \quad (7)$$

where $||\Delta\hat{\omega}||_2$ denotes the l_2-norm of the sparsified vector $\Delta\hat{\omega}$, and $\text{sgn}(\Delta\hat{\omega}_{(j)}) = \{+1, -1\}$ indicates the sign of $\Delta\hat{\omega}_{(j)}$. The quantization level z is approximately exponential to the number of bits $b_k^{(i)}$ used for quantization. When using b bits to quantize $\Delta\hat{\omega}_{(j)}$, one bit is allocated for its sign, and the remaining $b_k^{(i)} - 1$ bits are used to represent the function $\zeta(\Delta\hat{\omega}_{(j)}, z)$, resulting in a quantization level of $z = 2^{b_k^{(i)}-1} - 1$. $\zeta(\Delta\hat{\omega}_{(j)}, z)$ indicates an unbiased stochastic quantizer, mapping the scalar $\frac{|\Delta\hat{\omega}_{(j)}|}{||\Delta\hat{\omega}_{(j)}||_2}$ to one of the values in the set $\{0, \frac{1}{z}, \frac{2}{z}, \ldots, \frac{z}{z}\}$. If $\frac{|\Delta\hat{\omega}_{(j)}|}{||\Delta\hat{\omega}_{(j)}||_2}$ falls within the interval $[\frac{l}{z}, \frac{l+1}{z}]$, then $\zeta(\Delta\hat{\omega}_{(j)}, z)$ is probabilistically determined as

$$\zeta(\Delta\hat{\omega}_{(j)}, z) = \begin{cases} \frac{l}{z}, & \text{with probability } 1 - pr, \\ \frac{l+1}{z}, & \text{with probability } pr, \end{cases} \quad (8)$$

where $pr = z\frac{|\Delta\hat{\omega}_{(j)}|}{||\Delta\hat{\omega}_{(j)}||_2} - l$.

This stochastic quantization process allows for a trade-off between the precision of the model updates and the convergence speed. The more bits used for quantization, the finer the precision of each update. The quantization level z is only affected by quantization bit-width $b_k^{(i)}$. Adjusting $b_k^{(i)}$ is to adjust z. Hence, quantization bit-width $b_k^{(i)}$ of device i in the k-th communication round can be adjusted dynamically depending on network conditions and resource constraints.

C. Communication and Computation Time Model of FL

The training time per communication round within this FL system is influenced by the uploading time, downloading time, and local computation time of participating devices. Firstly, based on the sparsification and quantization transmission, the uploading time $\lambda_{u,k}^{(i)}$ and downloading time $\lambda_{d,k}^{(i)}$ of participating device i in k-th communication round can be described as [15] [29] [30],

$$\lambda_{u,k}^{(i)} = \frac{m_k^{(i)}(b_k^{(i)} + \log_2 d) + u}{c_{u,k}^{(i)}}, \quad (9)$$

$$\lambda_{d,k}^{(i)} = \frac{du}{c_{d,k}^{(i)}}, \quad (10)$$

where the term $m_k^{(i)}(b_k^{(i)} + \log_2 d) + u$ encapsulates the total number of bits required for a model update. $\log_2 d$ indicates the number of bits required to uniquely identify each non-zero element's index in the sparsified model update. The variable u denotes the size of a parameter unit in a single-precision floating-point format [8] [13]. This is used to represent the magnitude of elements in the model update as well as the quantization norm. The terms $c_{u,k}^{(i)}$ and $c_{d,k}^{(i)}$ denote the available bandwidths for uploading and downloading for device i in k-th communication round, respectively. These bandwidths are not static and can fluctuate during the FL process due to varying network conditions.

Secondly, the computation time $\lambda_{c,k}^{(i)}$ for device i during the k-th communication round is determined by the local processing of model updates. It is directly influenced by the number of local iterations δ_k and local computing capability, which can be denoted as,

$$\lambda_{c,k}^{(i)} = v_k^{(i)}\delta_k^{(i)}, \quad (11)$$

where $v_k^{(i)}$ represents the time required to compute a single batch of samples on device i. $v_k^{(i)}$ is not static and can fluctuate during the FL process due to varying computing capability.

The training time per communication round is ultimately determined by the longest training time taken among all participants. Thus, the total training time $\lambda_{total,k}$ for the k-th communication round is given by

$$\lambda_{total,k} = \max_{i \in \mathcal{R}_k}(\lambda_{u,k}^{(i)} + \lambda_{d,k}^{(i)} + \lambda_{c,k}^{(i)}). \quad (12)$$

where $\lambda_{u,k}^{(i)} + \lambda_{d,k}^{(i)} + \lambda_{c,k}^{(i)}$ represents the training time of device i in the k-th communication round.

D. Problem Formulation

In this paper, we focus on improving convergence speed while maintaining model accuracy. To achieve this objective, we need to design an optimized FL process that dynamically selects participating devices for each communication round $\mathcal{R}_k$ and coordinates hyperparameters for each selected device, including epoch size $e_k^{(i)}$ (Note: We adjust the local epochs to modify the local iterations), sparsification level $m_k^{(i)}$, and quantization bit-width $d_k^{(i)}$. Therefore, the objective can be

defined as the minimization of the global objective function $F(\omega)$ and the total training time of all communication rounds $\sum_{k=1}^{K}\lambda_{total,k}$, which can be given by

$$\min F(\omega)+\sum_{k=1}^{K}\lambda_{total,k}, \tag{13a}$$

subject to

$$\mathcal{R}_k \subseteq \mathcal{I}, \tag{13b}$$
$$e_k^{(i)} \in \mathbb{Z}^+, \forall k, \tag{13c}$$
$$m_k^{(i)} \in \mathbb{Z}^+, \forall i \in \mathcal{R}_k, \forall k, \tag{13d}$$
$$b_k^{(i)} \in \mathbb{Z}^+, \forall i \in \mathcal{R}_k, \forall k, \tag{13e}$$
$$v_{\min}^{(i)} \le v_k^{(i)} \le v_{\max}^{(i)}, \tag{13f}$$
$$c_{u,\min}^{(i)} \le c_{u,k}^{(i)} \le c_{u,\max}^{(i)}, \tag{13g}$$
$$c_{d,\min}^{(i)} \le c_{d,k}^{(i)} \le c_{d,\max}^{(i)}. \tag{13h}$$

Constraints (13f), (13g), and (13h) impose the minimum and maximum limits for the computing capability, and bandwidths for uploading and downloading, respectively. We note that since this optimization problem is NP-hard and non-convex, it is difficult to find an optimal solution.

IV. THEORETICAL ANALYSIS

Since $F(\omega)$ can only be obtained after FL is completed, it cannot be used to guide the FL process. Furthermore, $\sum_{k=1}^{K}\lambda_{total,k}$ can only be determined after the communication round is completed. Therefore, we transform our optimization problem, which rewrites the expression in (13) towards two goals. That is, we use the theoretical convergence bound instead of $F(\omega)$ and use the difference in training time between adjacent communication rounds instead of $\sum_{k=1}^{K}\lambda_{total,k}$.

Theoretical Convergence Bound: The convergence bound acts as a direct and measurable objective for optimization, streamlining the complexity of minimizing $F(\omega)$. Inspired by [8] and [31], we proceed to derive our theoretical convergence bound based on the following assumptions.

Assumption 1. The local objective function $f^{(i)}(\cdot)$ are L-smooth with respect to ω, i.e., for any $\omega,\omega' \in \mathbb{R}_d$, $\left\|\nabla f^{(i)}(\omega)-\nabla f^{(i)}(\omega')\right\| \le L\left\|\omega-\omega'\right\|$.

Assumption 2. Gradients $g^{(i)}(\omega)$ are unbiased and variance bounded, i.e., $\mathbb{E}[g^{(i)}(\omega)] = \nabla f^{(i)}(\omega)$ and $\mathbb{E}[\left\|g^{(i)}(\omega)-\nabla f^{(i)}(\omega)\right\|^2] \le \sigma^2$.

According to Section III-B, the compression operator $C(\cdot)$ is unbiased, which can be denoted as $\mathbb{E}[C(\omega)|\omega]=\omega$, and its variance bounded by the squared l_2-norm of its input, which can be given by $\mathbb{E}[||C(\omega)-\omega||^2|\omega] \le (\frac{d-m}{m}+\frac{d}{4^b})||\omega||^2$. Given that machine learning objectives are not necessarily convex, our convergence bound is derived without assuming μ-strong convexity [31].

Theorem 1. For ease of analysis and computation, we consider a simplified derivation approach by fixing r, δ, m, and d. The theoretical convergence bound is given by

$$\frac{1}{K\delta}\sum_{k=1}^{K}\sum_{t=1}^{\delta}\mathbb{E}\left[\left\|\nabla F(\bar{\omega}_{k,t})\right\|^2\right] \le \frac{2L(F(\omega^0)-F^*)}{\sqrt{K\delta}}+\frac{N_1}{\sqrt{K\delta}}+N_2\frac{\delta-1}{K\delta}, \tag{14}$$

where N_1, N_2 and $\bar{\omega}_{k,t}$ are defined as

$$N_1 = \frac{(1+(\frac{d-m}{m}+\frac{d}{4^b}))\sigma^2}{n}\left(1+\frac{n(n-r)}{r(n-1)}\right),$$
$$N_2 = \frac{\sigma^2}{n}(n+1),$$
$$\bar{\omega}_{k,t} = \frac{1}{n}\sum_{i\in\mathcal{I}}\omega_{k,t}^{(i)}.$$

Proof. Considering the sequence of $\bar{\omega}_{k,t} = \frac{1}{n}\sum_{i\in\mathcal{I}}\omega_{k,t}^{(i)}$ under the conditions where Assumptions 1 and 2 are satisfied, according to [31], the convergence bound can be given by

$$\frac{1}{K\delta}\sum_{k=1}^{K}\sum_{t=1}^{\delta}\mathbb{E}\left\|\nabla F(\bar{\omega}_{k,t})\right\|^2 \le \frac{2L(F(\omega^0)-F^*)}{\sqrt{K\delta}}+\frac{N_1}{\sqrt{K\delta}}+N_2\frac{\delta-1}{K\delta}, \tag{15}$$

where N_1 and N_2 are defined as

$$N_1 = \frac{(1+q)\sigma^2}{n}\left(1+\frac{n(n-r)}{r(n-1)}\right),$$
$$N_2 = \frac{\sigma^2}{n}(n+1).$$

The constant X equals $\sqrt{\frac{n}{4(n-r)}}(1+q)$, where q represents the variance of the compression error relative to the squared l_2-norm of its input. The stepsize is defined as $\eta=\frac{1}{L\sqrt{K\delta}}$. The total number of iterations $K\delta$ and the period length δ satisfy

$$K\delta \ge 2 \quad \text{and} \quad \tau \le \frac{\sqrt{X}}{2+0.8-\frac{X}{8}}\sqrt{K\delta}. \tag{16}$$

According to [31], if a vector undergoes Rand-m sparsification followed by stochastic quantization, q equals to $(1+(\frac{d-m}{m}+\frac{d}{4^b}))$ [8]. By substituting it into (15), Theorem 1 is derived to present the theoretical convergence bound of the proposed method.

Given our optimization goal, we aim to achieve rapid convergence by adjusting hyperparameters r, e, m, and d while maintaining high accuracy. According to the convergence bound derived from Theorem 1, the term $\frac{N_1}{\sqrt{K\delta}}+N_2\frac{\delta-1}{K\delta}$ effectively provides immediate feedback on the impact of these hyperparameter adjustments. Consequently, we utilize $\frac{N_1}{\sqrt{K\delta}}+N_2\frac{\delta-1}{K\delta}$ as an approximation to replace the $F(\omega)$ in the optimization objective. For computational convenience, it is generally assumed that K is a large value [29]. Therefore, N_1 is adopted as our final metric to substitute for $F(\omega)$ [32].

Difference in Training Time: Due to the unpredictability of the total time of all communication rounds before global model convergence, we use the difference in training time between adjacent communication rounds, i.e., $\lambda_{total,k}-\lambda_{total,k-1}$, as

a metric. This approach allows us to measure communication efficiency improvements between consecutive rounds. This differential evaluation encourages the system to accelerate convergence.

Replacing the global objective $F(\omega)$ with the convergence bound and the total training time $\sum_{k=1}^{K}\lambda_{total,k}$ with the difference in training time between adjacent communication rounds, we can reformulate our optimization objective as

$$\begin{aligned}
\min \quad & N_1 + \lambda_{total,k} - \lambda_{total,k-1}, \\
\text{s.t.} \quad & \mathcal{R}_k \subseteq \mathcal{I}, \\
& e_k^{(i)} \in \mathbb{Z}^+, \forall k, \\
& m_k^{(i)} \in \mathbb{Z}^+, \forall i \in \mathcal{R}_k, \forall k, \\
& b_k^{(i)} \in \mathbb{Z}^+, \forall i \in \mathcal{R}_k, \forall k, \\
& v_{\text{min}}^{(i)} \leq v_k^{(i)} \leq v_{\text{max}}^{(i)}, \\
& c_{u,\text{min}}^{(i)} \leq c_{u,k}^{(i)} \leq c_{u,\text{max}}^{(i)}, \\
& c_{d,\text{min}}^{(i)} \leq c_{d,k}^{(i)} \leq c_{d,\text{max}}^{(i)}.
\end{aligned} \tag{17}$$

V. Deep Reinforcement Learning for Fast Convergence

A. MDP-Based Device Delection and Hyperparameters Configuration

To tackle this NP-hard optimization problem presented in (17), we design a Markov Decision Process (MDP) formulated as $\mathbb{M} = (\mathbb{S}, \mathbb{A}, \mathbb{P}, \mathbb{R}, \gamma)$ and employ a DRL framework to navigate the action space. Within this framework, an agent iteratively selects actions based on the observed state, leading to state transitions and rewards. The goal is to enhance the cumulative reward through an optimal policy that dictates the best action for any given state. The components of this MDP are detailed as

1) **State Space ($\mathbb{S}$).** The state for communication round k is denoted as $s(k) = \{\{\lambda_{c,k}^{(i)}\}_{i\in\mathcal{I}}, \{\lambda_{u,k}^{(i)}\}_{i\in\mathcal{I}}, \{\lambda_{d,k}^{(i)}\}_{i\in\mathcal{I}}, \{e_{k-1}^{(i)}\}_{i\in\mathcal{I}}, \{m_{k-1}^{(i)}\}_{i\in\mathcal{I}}, \{b_{k-1}^{(i)}\}_{i\in\mathcal{I}}\}$, where each component represent the set of computation time, uploading time, downloading time, epoch size, sparsification level and quantization bit-width, respectively.
2) **Action Space ($\mathbb{A}$).** Actions in communication round k entail selecting devices and configuring their hyperparameters, formalized as $a(k) = \{\mathcal{R}_k, \{e_k^{(i)}\}_{i\in\mathcal{R}_k}, \{m_k^{(i)}\}_{i\in\mathcal{R}_k}, \{b_k^{(i)}\}_{i\in\mathcal{R}_k}\}, a(k) \in \mathbb{A}$.
3) **Policy (π).** The policy maps states to actions, denoted as $\pi : \mathbb{S} \to \mathbb{A}$. For a given communication round k, the executed action $a(k)$ is determined by the strategy $a(k) = \pi(s(k))$, which aligns actions with the current state $s(k)$.
4) **Reward (ρ).** At the communication round k, the agent formulates its action $a(k)$, informed by the current state $s(k)$. The impact of each action, specifically its influence on the convergence speed and accuracy of the global model, is quantified using the reward function $\rho(s(k), a(k))$. The reward for a given action during communication round k is represented as

$$\rho(s(k), a(k)) = \lambda_{total,k-1} - \lambda_{total,k} - N_1(k), \tag{18}$$

where $N_1(k)$ represents the result obtained from inputting action $a(k)$ into the N_1 function. The overall objective of the system is to maximize the cumulative reward over time, which is represented as

$$\Phi = \sum_{k=1}^{K} \gamma\rho(s(k), a(k))), \tag{19}$$

where $\gamma \in (0, 1]$ is the discount factor for the reward.
5) **Next State ($\mathbb{S}'$).** The transition of the system to a subsequent state, $s(k+1)$, is triggered by the implementation of action $a(k)$ within the current state $s(k)$.

The primary objective is to achieve fast convergence while maintaining accuracy. To achieve this, the DRL agent focuses on maximizing the cumulative reward. It explores various actions to optimize rewards and find the optimal action, which can be denoted as

$$a^* = \text{argmax}\mathbb{E}\left[\sum_{k=1}^{K} \gamma\rho(s(k), a(k))\right]. \tag{20}$$

B. Training and Models of DRL

In our MDP model, we encounter a scenario where both the action and state spaces are continuous and have high dimensions. To solve this issue, the Deep Deterministic Policy Gradient (DDPG) method [33] is employed. The strength of DDPG is the ability to handle complex action and state spaces with numerous variables and ensure stable decision-making, which is suitable for our problem context. As shown in Fig. 1, the DDPG framework is composed of an actor network to decide upon actions, a critic network to assess the value of these actions, and target networks to ensure learning stability. Additionally, DDPG leverages an experience replay buffer to enhance the learning efficiency and robustness, which archives state transitions including the current state, executed action, observed reward, and the subsequent state.

Actor Network. The actor network analyzes the current state $s(k)$ and selects the convergence hyperparameters and participating devices. This process utilizes a function within the actor network that maps system states to actions, which is denoted as

$$a(k) = \pi(s(k)|\theta^\pi), \tag{21}$$

where θ^π denotes the parameters of the actor network. To refine these parameters, the network employs a policy gradient approach for optimization, which is given by

$$\nabla_{\theta^\pi} J = \frac{1}{\kappa}\sum_{j=1}^{\kappa}\left[\nabla_a Q(s_j, a|\theta^Q)|_{a=\pi(s_j|\theta^\pi)} \nabla_{\theta^\pi}\pi(s_j)\right], \tag{22}$$

where κ represents the number of samples drawn from the set of experiences $(s(k), a(k), \rho(k), s(k+1))$. It is verified that the deterministic policy gradient is equivalent to the stochastic policy gradient $\nabla_\pi(a|s, \theta^\pi)$ [34]. Therefore, the deterministic strategy gradient is shown as

$$\nabla_\pi(a|s, \theta^\pi) \approx \mathbb{E}_\pi\left[\nabla_a Q(s, a|\theta^Q)|_{a=\pi(s_j|\theta^\pi)} \nabla_{\theta^\pi}\pi(s)\right]. \tag{23}$$

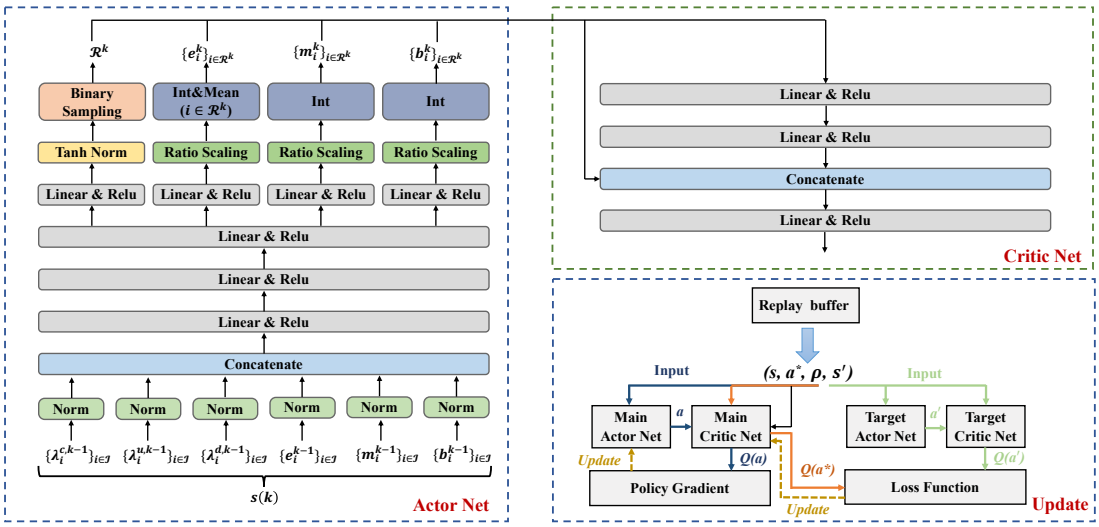

Fig. 1: Model Architecture of Our Proposed Dynamic Hyperparameter Configuration Method based on DDPG (DHC-DDPG).

In each DRL training iteration, a mini-batch of experiences is randomly chosen from the replay buffer to adjust the parameters of the actor network, which is denoted as

$$\theta^{\pi} = \theta^{\pi} + \eta^{\pi}\mathbb{E}\left[\nabla_a Q(s_j, a|\theta^Q)|_{a=\pi(s_j|\theta^{\pi})}\nabla_{\theta^{\pi}}\pi(s_j)\right], \quad (24)$$

where η^{π} is the learning rate of the actor network.

Critic Network. The critic network assesses the performance of the policy $\pi(s(k)|\theta^{\pi})$ and estimate the value of state-action pairs via the Q-function $Q^{\pi}(s(k), a(k)|\theta^Q)$. This function predicts the total expected rewards of selecting action $a(k)$ in state $s(k)$, following the Bellman equation for Q-values as

$$\begin{aligned} &Q^{\pi}(s(k), a(k)|\theta^Q) = \\ &\mathbb{E}\left[\rho(s(k), a(k)) + \gamma Q(s(k+1), \pi(s(k+1)|\theta^Q))\right]. \end{aligned} \quad (25)$$

Besides, the critic network compares predicted action outcomes with target predictions derived from the target network, facilitating precise adjustments to the training parameters θ^Q. The loss function can be defined as

$$\mathcal{L}(\theta^Q) = \mathbb{E}\left[\left(Q^{\pi}(s(k), a(k)|\theta^Q) - Y_{\text{target}}\right)^2\right], \quad (26)$$

where Y_{target} is determined by immediate and projected future rewards as

$$\begin{aligned} Y_{\text{target}}(k) = &\rho(s(k), a(k)) + \\ &\gamma Q'\left(s(k+1), \pi'\left(s(k+1)|\theta^{\pi'}\right)|\theta^{Q'}\right), \end{aligned} \quad (27)$$

where $\theta^{\pi'}$ and $\theta^{Q'}$ denote the parameters of the target networks, which are periodically updated to reflect the learning of the primary networks but are kept constant to stabilize training in between updates. Training of the critic network is executed through the processing of mini-batches of experience tuples $(s(k), a(k), \rho(k), s(k+1))$ randomly sampled from the replay buffer. This iterative training mechanism is guided by an update rule designed to minimize the defined loss function, which can be given by

$$\begin{aligned} \theta^Q = &\theta^Q + \\ &\eta^Q\mathbb{E}\left[2\left(Y_{\text{target},j} - Q(s_j, a_j|\theta^Q)\right)\nabla Q(s_j, a_j)\right]. \end{aligned} \quad (28)$$

Target Network and Experience Replay. To ensure the stability of our training process, target networks for the critic Q' and actor π' are utilized to serve as slower-moving versions of the main networks and smooth out the training updates. The parameters of these target networks are softly updated at intervals, which can be given by

$$\begin{aligned} \theta^{Q'} &= \tau\theta^Q + (1-\tau)\theta^{Q'}, \\ \theta^{\pi'} &= \tau\theta^{\pi} + (1-\tau)\theta^{\pi'}, \end{aligned} \quad (29)$$

where $\tau \in (0, 1]$ constrains the change of the target value.

C. Workflow of Our Method

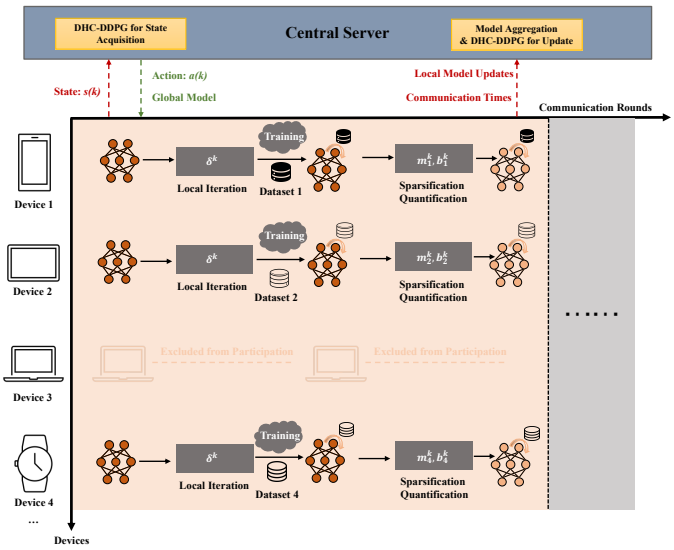

Fig. 2: Integration of FL Framework and DHC-DDPG.

Fig. 1 presents the model architecture of our proposed dynamic hyperparameter configuration method based on DDPG (DHC-DDPG) and Fig. 2 shows the integrated framework of DHC-DDPG and FL to dynamically set hyperparameters

Algorithm 1 FL based on DHC-DDPG running on the server

Input: Initial model parameters ω^1, DHC-DDPG parameters θ^π, θ^Q
$\theta^{\pi'} \leftarrow \theta^\pi$, $\theta^{Q'} \leftarrow \theta^Q$
Initialize replay buffer and initialize state $s(1)$
for each communication round k = 1,. . .,K **do**
  Calculate $a(k)$ by putting $s(k)$ into actor network
  $\{\delta_k^{(i)}\}_{i\in\mathcal{R}_k} \leftarrow \{\left\lceil \frac{|\mathcal{S}^{(i)}|}{|\mathcal{B}_k^{(i)}|} \right\rceil e_k^{(i)}\}_{i\in\mathcal{R}_k}$
  $\delta_k \leftarrow$ int(mean($\{\delta_k^{(i)}\}_{i\in\mathcal{R}_k}$))
  for each device $i \in \mathcal{R}_k$ **in parallel do**
    $C(\Delta\omega_{k+1}^{(i)}), \lambda_{u,k+1}^{(i)}, \lambda_{d,k+1}^{(i)}, \lambda_{c,k+1}^{(i)}$ $\leftarrow$ DEVICE$(i, \omega_k, \delta_k, m_k^{(i)}, b_k^{(i)})$
  end for
  Update the global model ω_{k+1} by using (6)
  Calculate reward $\rho(s(k), a(k))$ by using (18)
  Obtain $s(k+1)$ based on FL communication round k
  If the replay buffer is full, remove the oldest entry
  Store $(s(k), a(k), \rho(k), s(k+1))$ in replay buffer
  Sample κ experiences $(s(j), a(j), \rho(j), s(j+1))$
  Calculate the target value $Y_{\text{target},j}$ by using (27)
  Update the actor network $\pi(s|\theta^\pi)$ by using (24)
  Update the critic network $Q(s, a|\theta^Q)$ by using (28)
  Update $\theta^{\pi'}$ and $\theta^{Q'}$ by using (29)
end for
Output: The global model ω_{K+1}

Algorithm 2 FL based on DHC-DDPG running on the device

Input: Device index i, global model ω_k, local iteration size δ_k, sparsification level $m_k^{(i)}$, quantization bit-width $b_k^{(i)}$
for $t = 1, ..., \delta_k$ **do**
  Update the local model $\omega_{k+1}^{(i)}$ by using (5)
end for
$\Delta\omega_{k+1}^{(i)} \leftarrow \omega_{k+1}^{(i)} - \omega_k$
Calculate $\lambda_{u,k+1}^{(i)}$ by using (9)
Calculate $\lambda_{d,k+1}^{(i)}$ by using (10)
Calculate $\lambda_{c,k+1}^{(i)}$ by using (11)
Calculate $C(\Delta\omega_{k+1}^{(i)})$ by using (7)
Output: Compressed model update $C(\Delta\omega_{k+1}^{(i)})$, uploading time $\lambda_{u,k+1}^{(i)}$, downloading time $\lambda_{d,k+1}^{(i)}$, computation time $\lambda_{c,k+1}^{(i)}$

and device selection in FL. Algorithms 1 and 2 illustrate the workflow and integration of FL and DHC-DDPG. Next, we present the implementation details of DHC-DDPG and its integration into FL by providing a description of Algorithms 1 and 2.

The framework first initializes global model parameters ω^1. The DHC-DDPG parameters θ^π, θ^Q, and their respective target networks also need to be initialized. Additionally, a replay buffer is created to store experience samples and the initial state $s(1)$ is established. During communication round k, the actor network of DHC-DDPG takes the state $s(k)$ as input to calculate the action $a(k)$. Fig. 1 shows the architecture of the actor network in DHC-DDPG. Specifically, all parts in $s(k)$ are concatenated and then pass through three fully connected layers to produce the action $a(k)$. Following this, the number of local iteration δ_k is calculated by $\{\delta_k^{(i)}\}_{i\in\mathcal{R}_k}$ in $a(k)$ and the central server selects the participating devices based on $\mathcal{R}_k$ from $a(k)$. As shown in Algorithm 2, the selected devices update their local models according to δ_k, $m_k^{(i)}$, and $b_k^{(i)}$ provided by $a(k)$ and subsequently upload their compressed model updates to the central server. In addition to model updates, each participating device also needs to report the uploading time, downloading time, and computation time of the device in this communication round to the central server, which can be calculated using Equations (9), (10), and (11) given in Section III-C. Once the central server receives all updates, it performs model aggregation and obtains the new global model.

Next, the reward $\rho(s(k), a(k))$ is calculated. The next state $s(k+1)$ is updated by replacing old $\lambda_{c,k}, \lambda_{u,k}, \lambda_{d,k}, e, m$, and b of participating devices in s(k) with new ones. The central server checks whether the replay buffer is full. If it is full, replay buffer removes the oldest entry. Then it stores the new transition in the replay buffer. Finally, the central server samples κ experiences for batch processing, computes target values to update both the actor and critic networks, and proceeds to update their target networks accordingly.

Having outlined the overall process, we now perform a complexity analysis of the DHC-DDPG process based on [35]. Given the structure of the actor and critic networks in the DDPG algorithm, where the actor net has T_{la} layers each with T_{na} neurons, and the critic network has T_{lc} layers each with T_{nc} neurons, we perform an analysis on the time complexity of the algorithm. For the actor net, the complexity of forward propagation through one layer involves operations for all T_{na} neurons, approximated as $\mathcal{O}(T_{na}^2)$. Thus, the total complexity for the actor network's forward propagation across all T_{la} layers is $\mathcal{O}(T_{la} \times T_{na}^2)$. Similarly, for the critic network with T_{lc} layers and T_{nc} neurons per layer, the forward propagation complexity is $\mathcal{O}(T_{lc} \times T_{nc}^2)$. The complexity of backward propagation is analogous to that of forward propagation, given the need to calculate gradients for each neuron across all layers. The complexity of interacting with the environment is denoted as $\mathcal{O}(E)$, independent of the network architecture. With B experiences in the replay buffer, the complexity involved in sampling and updating operations is approximated as $\mathcal{O}(B)$. Combining these components, the overall time complexity of the DDPG algorithm, considering one update step that includes both networks' operations, environment interaction, and experience replay, can be represented as, $\mathcal{O}(T_{la} \times T_{na}^2 + T_{lc} \times T_{nc}^2 + E + B)$.

VI. Evaluation

In this section, we evaluate the performance of our method using various experiments. We assess its efficiency, scalability, computational demands, and convergence behavior through a series of benchmarks and comparisons with baseline methods.

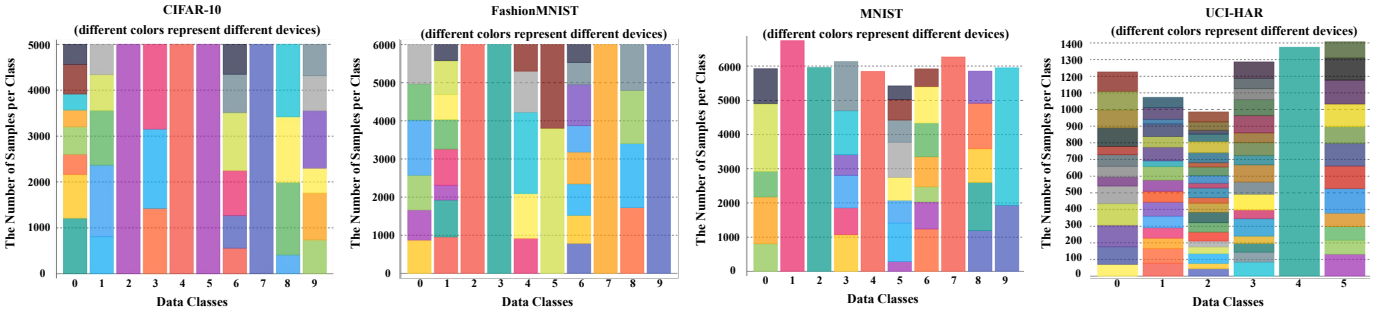

Fig. 3: Distribution of Samples Across Devices.

We provide a detailed analysis of key aspects of DHC-DDPG such as actor network loss, policy network loss, and reward trends. We also provide an analysis of the adaptability of the DHC-DDPG decision-making process.

A. Experimental Setup

Dataset and Preprocessing: In our experiments, we use the MNIST, FashionMNIST, CIFAR-10, and UCI-HAR datasets to evaluate the performance of our method. MNIST and FashionMNIST consist of grayscale images of handwritten digits and fashion items, respectively, while CIFAR-10 contains different color images, and UCI-HAR involves human activity recognition data. The MNIST, FashionMNIST, and CIFAR-10 datasets are distributed across 20 devices, while the UCI-HAR dataset is distributed across 40 devices. Similar to the approach described in [36], Fig. 3 illustrates how we distribute the data across devices and different colors represent different devices. To simulate non-independent and identically distributed (non-IID) and unbalanced scenarios, we control the sampling frequency of each class in the dataset. Specifically, the majority of classes can be sampled multiple times, while certain classes are sampled only once (e.g., in a 10-class dataset, 4 classes are sampled once, while in a 6-class dataset, 1 class is sampled once). Each device samples twice from the available classes. The data is then distributed to the devices based on the sampling results. In this way, each device contains samples from at least one class and no more than two classes.

Local Model Setting: We tailor the model for each dataset, and the model architectures are shown in Table II. Additionally, Table III shows the default FL hyperparameters used in our experiments. Since our method dynamically adjusts hyperparameters, the default hyperparameters in this table are mainly used for the baseline settings.

DHC-DDPG Configuration: In configuring the DHC-DDPG algorithm for our experiments, we use the Adam optimizer for both the actor and critic networks. The key hyperparameters, including learning rates, weight decay, and the reward parameter σ, are detailed in Table IV.

Baseline Methods: We compare the proposed method with several state-of-the-art FL methods, including 1) FedAvg [37]: The foundational algorithm of FL. 2) FedPAQ [31]: This method optimizes FL through periodic averaging, quantization, and selective participation but maintains static hyperparameter settings. 3) DTEI [30]: This method leverages digital twin and DRL to optimize FL for quicker convergence and improved accuracy through strategic device selection. 4) Qsparse [21]: This method lowers communication costs and speeds up convergence by blending sparsification, quantization, and error compensation. 5) Fedeco [15]: This method adaptively sets local iterations by decomposing the problem into sub-problems to ensure model convergence.

Dynamic Scenario: Our dynamic scenario simulations are similar to the approach outlined in [38]. To simulate computing capabilities (i.e., the time to compute a batch), we randomly assign devices into three categories: high, medium, and low. The average computation times for these categories are 0.5, 0.7, and 1 second per batch, respectively. Each device is randomly assigned to one category. During the FL process, the processing speed of each device fluctuates within a standard deviation of 0.02 seconds around its assigned average. To simulate network conditions (i.e., the bandwidths for uploading and downloading), we randomly divide devices into high-bandwidth and low-bandwidth types. For high-bandwidth devices, average download and upload bandwidths are 30 Mbps and 8 Mbps, respectively. For low-bandwidth devices, they are 5 Mbps and 0.5 Mbps. Each device is randomly assigned to one category. During the FL process, network speeds fluctuate. In the high-bandwidth group, download bandwidths vary by 5 Mbps and upload bandwidths by 2 Mbps. In the low-bandwidth group, download bandwidths vary by 1 Mbps and upload bandwidths by 0.2 Mbps.

B. Performance Comparison

In this section, we conduct a performance comparison between our method and the baselines, focusing on the accuracy of the global model over communication rounds. Fig. 4 illustrates the convergence performance of the different methods on the MNIST, FashionMNIST, CIFAR-10, and UCI-HAR datasets.

On the CIFAR-10 dataset, baselines show slightly better accuracy in the first 30 rounds. However, after about 30 rounds, our method starts to outperform them, maintaining higher accuracy and stabilizing at around 0.8 by 60 rounds. In contrast, the accuracies of the methods such as DTEI, FedAvg, and Qsparse remain close to 0.75, with small improvement over time. For the FashionMNIST dataset, all methods reach an accuracy of around 0.97 by the 35 rounds. As shown in the zoomed-in section, our method converges faster and remains

TABLE II: Model Architecture for MNIST / FashionMNIST, CIFAR-10, and UCI-HAR

Dataset	Layers
MNIST / FashionMNIST	2 × (Conv2D with 32 and 64 filters (3×3), ReLU), MaxPooling (2×2), Dropout(0.25)
	Flatten, Dense layer with 128 units, ReLU, Dropout(0.5), Dense layer with class num units, Softmax
CIFAR-10	Conv2D with 32 filters (3×3), BatchNorm, ReLU
	Conv2D with 64 filters (3×3), BatchNorm, ReLU, MaxPooling (2×2)
	Conv2D with 128 filters (3×3), BatchNorm, ReLU
	Dropout(0.5), MaxPooling (2×2), Flatten, Dense layer with 512 units, ReLU, Dropout(0.5), Dense layer with class num units, Softmax
UCI-HAR	4 × Dense layer with 512, 256, 128, 64 units, ReLU, Dropout(0.5)
	Flatten, Dense layer with class num units, Softmax

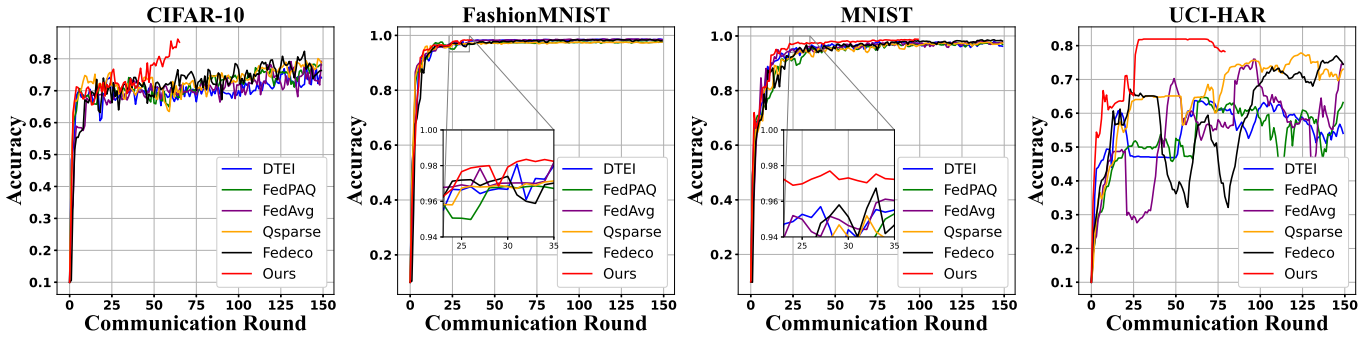

Fig. 4: Convergence Performance of Various Methods on Each Dataset.

TABLE III: Default Hyperparameters of FL

Dataset	Epoch	Batch Size	Learning Rate (Momentum)	Participant Num.	Quant. Level	Spars. Level
MNIST	5	256	0.5 (-)	10	3	80% of model param.
FashionMNIST	10	256	0.5 (-)	10	3	
CIFAR-10	10	256	0.001 (0.9)	10	3	
UCI-HAR	1	256	0.005 (0.0009)	20	3	

TABLE IV: Default Hyperparameters of Our DHC-DDPG on All Dataset

Learning Rate of Actor Network	Learning Rate of Critic Network	Decay of Critic Network	σ
0.001	0.001	0.1	0.001

more stable in accuracy. The MNIST dataset shows similar trends to FashionMNIST, with all methods reaching near-peak accuracy quickly. Our method shows a slight advantage in convergence speed, reaching a higher accuracy of about 0.97 within the first 20 rounds. On the UCI-HAR dataset, our method performs significantly better, stabilizing around 0.8 after 25 rounds. Other methods, such as FedAvg and DTEI, exhibit more fluctuations and tend to hover around 0.65 accuracy. Overall, the advantage of our method lies in its ability to adjust hyperparameters dynamically and adapt to heterogeneous and non-IID data distributions. This enables faster convergence and higher final accuracy compared to other methods.

Methods like FedPAQ and Qsparse, which apply quantization and sparsification, perform similarly to FedAvg on some datasets. This is likely because quantization and sparsification help reduce local model overfitting to local data while preserving important model updates in non-IID and unbalanced scenarios.

C. *Efficiency Analysis of Training Time in FL*

Fig. 5 compares the training time required by different methods to reach target accuracies on all datasets in our simulation scenario. On the CIFAR-10 dataset, our method reaches a target accuracy of 75% in 7.23 hours (i.e., 7 hours, 13 minutes, and 48 seconds). In comparison, FedPAQ and FedAvg require significantly more time, taking 15.59 and 32.13 hours, respectively. Qsparse and DTEI also take around 10 to 30 hours. This results in our method being 76.1% faster than DTEI and 77.5% faster than FedAvg. On the FashionMNIST dataset, our method achieves 97% accuracy in just 0.82 hours, whereas FedAvg takes over twice as long at 2.01 hours. Fedeco and DTEI require 1.73 and 1.88 hours, respectively. On the MNIST dataset, our method reaches 95% accuracy in 0.61 hours. FedAvg needs over 3 hours to achieve the same accuracy, while other methods take approximately 2 hours. On the UCI-HAR dataset, our method achieves 60% accuracy in 0.57 hours. FedAvg and FedPAQ take significantly longer, at 2.26 and 4.35 hours, respectively. Fedeco shows a smaller difference, reaching 60% accuracy in 0.62 hours, which is about 8.8% longer than our method. Our method consistently demonstrates a substantial reduction in training time to achieve target accuracy across various datasets compared to other methods. This efficiency is particularly notable on larger and more complex datasets such as CIFAR-10.

To further understand the advantage of our method, we analyze the average training time per communication round. While our method shows a significant overall time reduction, the per-round advantage is less pronounced, as shown in Fig. 6. On the CIFAR-10 dataset, our method takes 0.26

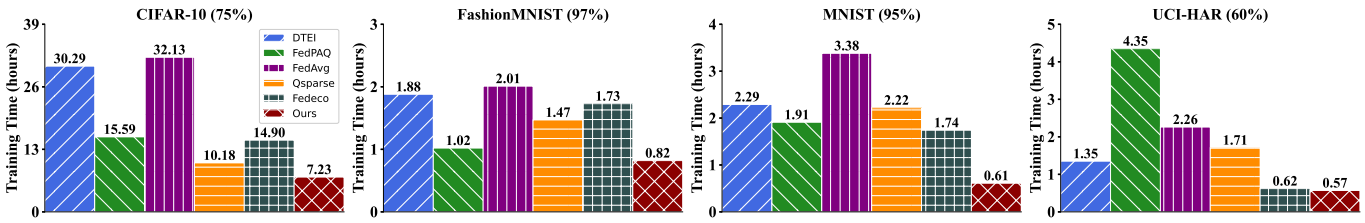

Fig. 5: Training Time to Reach Target Accuracy on Each Dataset.

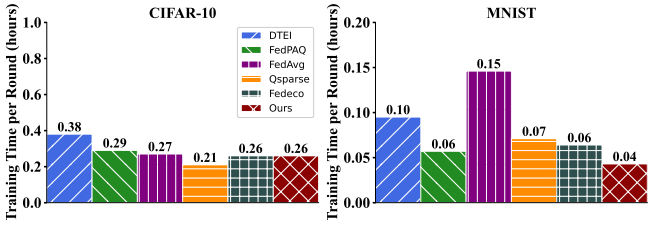

Fig. 6: Average Training Time per Communication Round on Each Dataset.

hours per communication round, which is almost the same as Fedeco, FedAvg, and FedPAQ. Likewise, for the MNIST dataset, the per-round time of our method is nearly the same as that of Qsparse and Fedeco. This suggests that the primary advantage of our method lies in reducing the total number of communication rounds, rather than optimizing the time spent in each communication round. While our method is highly efficient overall, there is still potential to reduce the per-round time for further improvements.

D. Scalability Evaluation with Increasing Devices

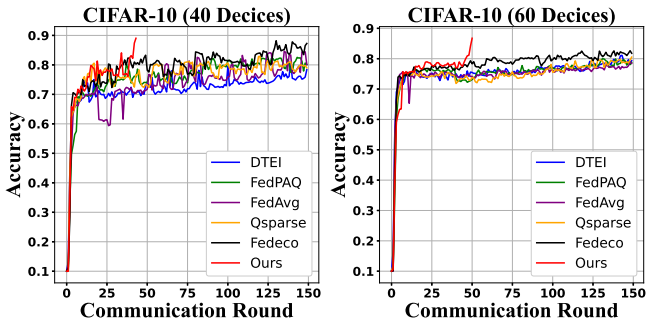

Fig. 7: Convergence Performance of Various Methods on CIFAR-10 with 40 and 60 Devices.

To evaluate the scalability of our method, we conduct experiments on the CIFAR-10 dataset with an increased number of devices. Specifically, we perform the test with 40 and 60 devices to observe how accuracy evolves over communication rounds. As shown in Fig. 7, our method maintains high accuracy with the increasing number of devices. For the experiment with 40 devices, our method reaches an accuracy of around 0.8 in about 25 communication rounds. It shows faster convergence speed than those of the other methods. Other methods including FedPAQ, FedAvg, and Qsparse take longer to reach similar accuracy levels. In the experiment with 60 devices, our method also shows better performance. It maintains an accuracy close to 0.8 with fewer communication rounds. The baseline methods converge more slowly and reach lower final accuracies. These results demonstrate the scalability of our method in FL scenarios with larger numbers of devices. Even as the number of devices increases, our method maintains consistent performance.

E. Analysis of Computational Demands

We compare the computational demands of our method with two baseline methods: DTEI and Fedeco. As shown in Fig. 8, the comparison is based on the runtime per iteration for each method on four datasets. The reason for selecting these baselines is that, similar to our method, they both utilize optimization strategies for improving the FL process. On the CIFAR-10 dataset, our method demonstrates an overhead of 0.0072 seconds per iteration. This overhead is marginally better than DTEI's 0.0097 seconds, and significantly more efficient compared to Fedeco's 0.1231 seconds. For the FashionMNIST and MNIST datasets, the computational demands of our method are 0.0064 and 0.0067 seconds, respectively. On the UCI-HAR dataset, our method incurs a runtime of 0.0179 seconds per iteration, which is marginally higher than DTEI's 0.0152 seconds but remains substantially more efficient than Fedeco's 0.1179 seconds. Although our method does not always outperform DTEI in computational overhead, our method is significantly better in terms of convergence speed and accuracy compared with DTEI. Furthermore, our computational overhead is consistently lower than that of Fedeco on all datasets.

F. Analysis of Loss and Reward Convergence in DHC-DDPG

Fig. 9 presents actor network loss, policy network loss, and reward trends of DHC-DDPG in our method over communication rounds. Each dataset requires a different number of communication rounds to reach convergence, which we annotate in Fig. 9.

It is observed that the actor network loss across all datasets decreases sharply and approaches near-zero values as the number of communication rounds increases. During the initial phase of training, the loss exhibits volatility. This early instability gives way to stabilization over time, likely due to the accumulation of more varied data in the replay buffer. The FashionMNIST dataset shows the most rapid decline in loss, dropping to below 0.02 within the first 20 rounds but

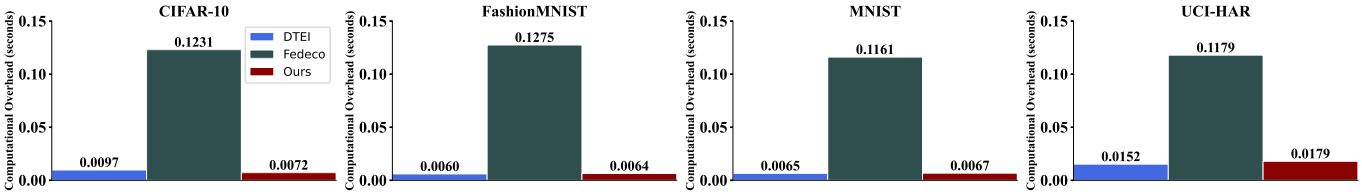

Fig. 8: Runtime per Iteration of DHC-DDPG and Baselines on Each Dataset.

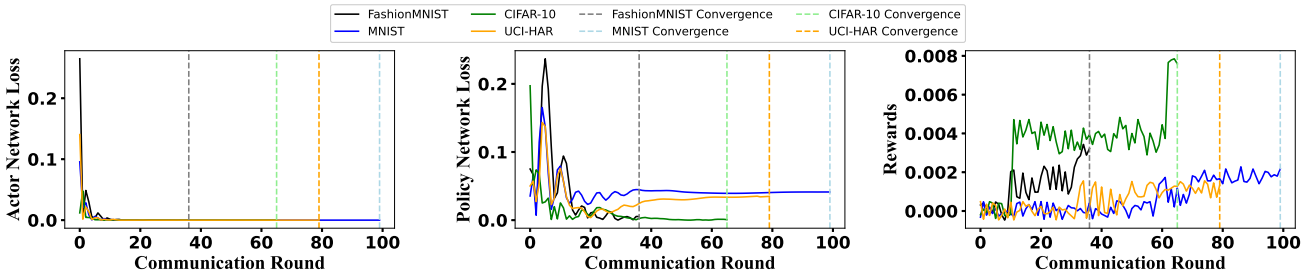

Fig. 9: Variation of Actor Model Loss, Policy Model Loss, and Reward Values over Communication Rounds.

also displays the greatest volatility. On the other hand, the CIFAR-10 dataset demonstrates minimal fluctuations, consistently maintaining a low loss under 0.1 throughout the communication rounds. This indicates that the actor network loss tends to be more stable on complex datasets like CIFAR-10.

In the initial stages, the policy network loss spikes for all datasets. This spike indicates the exploratory learning phase as the algorithm adjusts to the environment. The FashionMNIST dataset shows the most pronounced fluctuations, suggesting a complex learning process. In contrast, CIFAR-10 exhibits a smoother reduction in loss, maintaining values typically below 0.05. As training advances, the policy network loss on each dataset gradually decreases. It stabilizes as the network converges toward optimal policy formulation.

Rewards for all datasets fluctuate, reflecting the trial-and-error throughout the phase. Over time, our method refines its policies, and we see a general trend of increasing rewards. Especially in the CIFAR-10 dataset, the rewards trend higher. Meanwhile, the other datasets, while exhibiting more erratic reward patterns, eventually show a sharp increase. This sharp increase signifies that our method has adapted to the varied learning challenges presented by each dataset.

G. Adaptive Decision-Making of DHC-DDPG

To investigate how DHC-DDPG tailors these hyperparameters to train each dataset, we track and analyze the selected actions throughout training, including how many devices are selected for participation in each communication round, how quantization bit-width and sparsification level are adjusted over time, and what the number of epochs per device is set (Note: We adjust the local epochs to modify the local iterations). However, it is too complex to display all the details for every device in every communication round. To simplify the presentation, we show the average values of quantization bit-width and sparsification level for each communication round, as shown in Fig. 10.

In the CIFAR-10 dataset, our method employs an incremental adjustment strategy, where the quantization bit-width starts low but gradually rises, reaching a peak of approximately 12 by the 60-th communication round. This gradual increase is necessary given the complexity of the CIFAR-10 dataset, which has over 4 million parameters. Concurrently, a conservative adjustment to sparsification is observed, maintaining a low rate of zero-valued parameters to avoid significant information loss. The epoch also shows a significant increase, which indicates a strategic investment in computational resources to refine model accuracy. It is also apparent that CIFAR-10 requires a higher number of devices for participation, leveraging computational resources to enhance model accuracy.

For MNIST, FashionMNIST, and UCI-HAR, the quantization bit-widths remain consistently low throughout the process, indicating a strategy of aggressive compression. This method is well-suited to these simpler datasets and their smaller model sizes, allowing for significant data compression without notably impacting model performance. Despite this aggressive approach to quantization, these three datasets exhibit a conservative pattern in sparsification. This reveals a preference to leverage quantization over sparsification, potentially to maintain the essential representational capacity of the neural networks.

In terms of epochs, the FashionMNIST dataset shows a marked increase, suggesting an appreciation for the benefits of increased local iterations within computational resource constraints. Conversely, the results from MNIST and UCI-HAR maintain a stable and relatively lower count of epochs, indicating that for simpler datasets, higher epochs do not necessarily yield substantial accuracy gains.

MNIST, FashionMNIST, and UCI-HAR engage a similar number of devices per communication round. This indicates that extensive device involvement is not necessary for these

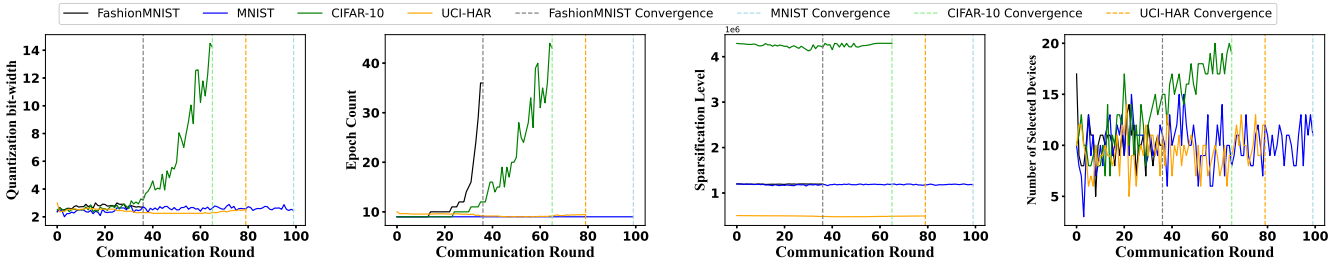

Fig. 10: Adaptive Hyperparameter Configuration on Each Dataset.

datasets. Our method adjusts the number of participating devices according to the specific challenges and objectives of each dataset. It efficiently balances computational demand against learning efficiency.

VII. Conclusion

This paper introduced a new framework designed to accelerate FL convergence while maintaining global model accuracy. By integrating adaptive quantization, sparsification, and device selection, we formulated an optimization problem to achieve FL fast convergence. We conducted a theoretical analysis for the proposed framework and derive a convergence bound. We dynamically adjusted multiple hyperparameters affecting FL convergence using DRL, based on a reward derived from this convergence bound. Experimental results confirmed that our method performs well in heterogeneous scenarios and outperforms existing FL methods.

VIII. Future Work

While our method shows substantial improvements in convergence speed, there are still areas for further enhancement, particularly in terms of the efficiency within a single communication round. Specifically, there is room to optimize average training time and reduce runtime during each communication round. Another area that needs improvement is ensuring fairness in device selection, as the current method may introduce slight imbalances. A most efficient FL system requires addressing both these aspects—optimizing the per-round performance and ensuring fair device participation. These are key areas for future research and refinement in developing a more comprehensive and balanced FL framework.

References

[1] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, Brendan McMahan, et al. Towards federated learning at scale: System design. *Proceedings of Machine Learning and Systems*, 1:374–388, 2019.

[2] Jinjin Xu, Wenli Du, Yaochu Jin, Wangli He, and Ran Cheng. Ternary compression for communication-efficient federated learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[3] Daniel Rothchild, Ashwinee Panda, Enayat Ullah, Nikita Ivkin, Ion Stoica, Vladimir Braverman, Joseph Gonzalez, and Raman Arora. Fetchsgd: Communication-efficient federated learning with sketching. In *International Conference on Machine Learning*, pages 8253–8265. PMLR, 2020.

[4] Jiacheng Wang, Hongyang Du, Dusit Niyato, Jiawen Kang, Shuguang Cui, Xuemin Sherman Shen, and Ping Zhang. Generative ai for integrated sensing and communication: Insights from the physical layer perspective. *IEEE Wireless Communications*, 2024.

[5] Jiacheng Wang, Hongyang Du, Dusit Niyato, Zehui Xiong, Jiawen Kang, Bo Ai, Zhu Han, and Dong In Kim. Generative artificial intelligence assisted wireless sensing: Human flow detection in practical communication environments. *IEEE Journal on Selected Areas in Communications*, 2024.

[6] Hongyang Pan, Yanheng Liu, Geng Sun, Junsong Fan, Shuang Liang, and Chau Yuen. Joint power and 3d trajectory optimization for uav-enabled wireless powered communication networks with obstacles. *IEEE transactions on communications*, 71(4):2364–2380, 2023.

[7] Xiangwang Hou, Jingjing Wang, Chunxiao Jiang, Zezhao Meng, Jianrui Chen, and Yong Ren. Efficient federated learning for metaverse via dynamic user selection, gradient quantization and resource allocation. *IEEE Journal on Selected Areas in Communications*, 2023.

[8] Guangfeng Yan, Tan Li, Shao-Lun Huang, Tian Lan, and Linqi Song. Ac-sgd: Adaptively compressed sgd for communication-efficient distributed learning. *IEEE Journal on Selected Areas in Communications*, 40(9):2678–2693, 2022.

[9] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*, 2017.

[10] Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. Sparse binary compression: Towards distributed deep learning with minimal communication. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.

[11] Qian Chen, Zilong Wang, Jiawei Chen, Haonan Yan, and Xiaodong Lin. Dap-fl: Federated learning flourishes by adaptive tuning and secure aggregation. *IEEE Transactions on Parallel and Distributed Systems*, 34(6):1923–1941, 2023.

[12] Milad Khademi Nori, Sangseok Yun, and Il-Min Kim. Fast federated learning by balancing communication trade-offs. *IEEE Transactions on Communications*, 69(8):5168–5182, 2021.

[13] Divyansh Jhunjhunwala, Advait Gadhikar, Gauri Joshi, and Yonina C Eldar. Adaptive quantization of model updates for communication-efficient federated learning. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3110–3114. IEEE, 2021.

[14] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K Leung, Christian Makaya, Ting He, and Kevin Chan. Adaptive federated learning in resource constrained edge computing systems. *IEEE journal on selected areas in communications*, 37(6):1205–1221, 2019.

[15] Yutao Liu, Xiaoning Zhang, Zhihao Zeng, and Shui Yu. Fedeco: Achieving energy-efficient federated learning by hyperparameter adaptive tuning. *IEEE Transactions on Cognitive Communications and Networking*, 2024.

[16] Huanle Zhang, Lei Fu, Mi Zhang, Pengfei Hu, Xiuzhen Cheng, Prasant Mohapatra, and Xin Liu. Federated learning hyperparameter tuning from a system perspective. *IEEE Internet of Things Journal*, 10(16):14102–14113, 2023.

[17] Yae Jee Cho, Jianyu Wang, and Gauri Joshi. Towards understanding biased client selection in federated learning. In *International Conference on Artificial Intelligence and Statistics*, pages 10351–10375. PMLR, 2022.

[18] Wenqi Shi, Sheng Zhou, and Zhisheng Niu. Device scheduling with fast convergence for wireless federated learning. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2020.

[19] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[20] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. *Advances in neural information processing systems*, 30, 2017.

[21] Debraj Basu, Deepesh Data, Can Karakus, and Suhas N Diggavi. Qsparse-local-sgd: Distributed sgd with quantization, sparsification, and local computations. *IEEE Journal on Selected Areas in Information Theory*, 1(1):217–226, 2020.

[22] Sebastian U Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. Sparsified sgd with memory. *Advances in neural information processing systems*, 31, 2018.

[23] Chengxi Li, Gang Li, and Pramod K Varshney. Communication-efficient federated learning based on compressed sensing. *IEEE Internet of Things Journal*, 8(20):15531–15541, 2021.

[24] Feng Zheng, Yuze Sun, and Bin Ni. Fedaeb: Deep reinforcement learning based joint client selection and resource allocation strategy for heterogeneous federated learning. *IEEE Transactions on Vehicular Technology*, 2024.

[25] Tantan Zhao, Fan Li, and Lijun He. Drl-based joint resource allocation and device orchestration for hierarchical federated learning in noma-enabled industrial iot. *IEEE Transactions on Industrial Informatics*, 19(6):7468–7479, 2022.

[26] Bing Luo, Xiang Li, Shiqiang Wang, Jianwei Huang, and Leandros Tassiulas. Cost-effective federated learning design. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2021.

[27] Wen Sun, Shiyu Lei, Lu Wang, Zhiqiang Liu, and Yan Zhang. Adaptive federated learning and digital twin for industrial internet of things. *IEEE Transactions on Industrial Informatics*, 17(8):5605–5614, 2020.

[28] Majid Kundroo and Taehong Kim. Federated learning with hyper-parameter optimization. *Journal of King Saud University-Computer and Information Sciences*, 35(9):101740, 2023.

[29] Guangfeng Yan, Shao-Lun Huang, Tian Lan, and Linqi Song. Dq-sgd: Dynamic quantization in sgd for communication-efficient distributed learning. In *2021 IEEE 18th International Conference on Mobile Ad Hoc and Smart Systems (MASS)*, pages 136–144. IEEE, 2021.

[30] Wei Yang, Wei Xiang, Yuan Yang, and Peng Cheng. Optimizing federated learning with deep reinforcement learning for digital twin empowered industrial iot. *IEEE Transactions on Industrial Informatics*, 19(2):1884–1893, 2022.

[31] Amirhossein Reisizadeh, Aryan Mokhtari, Hamed Hassani, Ali Jadbabaie, and Ramtin Pedarsani. Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization. In *International Conference on Artificial Intelligence and Statistics*, pages 2021–2031. PMLR, 2020.

[32] Xing Xu, Rongpeng Li, Zhifeng Zhao, and Honggang Zhang. The gradient convergence bound of federated multi-agent reinforcement learning with efficient communication. *IEEE Transactions on Wireless Communications*, 2023.

[33] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[34] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. Pmlr, 2014.

[35] Yijing Lin, Zhipeng Gao, Hongyang Du, Jiawen Kang, Dusit Niyato, Qian Wang, Jingqing Ruan, and Shaohua Wan. Drl-based adaptive sharding for blockchain-based federated learning. *IEEE Transactions on Communications*, 2023.

[36] Yuanli Wang, Joel Wolfrath, Nikhil Sreekumar, Dhruv Kumar, and Abhishek Chandra. Accelerated training via device similarity in federated learning. In *Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking*, pages 31–36, 2021.

[37] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[38] Young Geun Kim and Carole-Jean Wu. Autofl: Enabling heterogeneity-aware energy efficient federated learning. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 183–198, 2021.

Xinlei Yu is a Ph.D. candidate at the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications (BUPT), Beijing, China. Her interests include Federated Learning, Edge Computing, Convolutional Neural Network and Machine Learning.

Yijing Lin received the doctoral degree from the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications (BUPT), where she is working as a postdoctoral researcher. Her current research interests include blockchain and federated unlearning.

Zhipeng Gao received the Ph.D. degree from the Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 2007. He is currently a Professor with the State Key Laboratory of Networking and Switching Technology, BUPT, Beijing, China. His research interests are wireless networks, edge computing and blockchain. He presides over a series of key research projects on network and service management, including the projects supported by the National Natural Science Foundation and the National High-Tech Research and Development Program of China.

Hongyang Du is an assistant professor at the Department of Electrical and Electronic Engineering, The University of Hong Kong. He received the BEng degree from the School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing, in 2021, and the PhD degree from the Interdisciplinary Graduate Program at the College of Computing and Data Science, Energy Research Institute @ NTU, Nanyang Technological University, Singapore, in 2024. He is the Editor-in-Chief assistant of IEEE Communications Surveys & Tutorials (2022-2024). He is the recipient of the IEEE Daniel E. Noble Fellowship Award from the IEEE Vehicular Technology Society in 2022, the IEEE Signal Processing Society Scholarship from the IEEE Signal Processing Society in 2023, the Chinese Government Award for Outstanding Students Abroad in 2023, and the Singapore Data Science Consortium (SDSC) Dissertation Research Fellowship in 2023. He was recognized as an exemplary reviewer of the IEEE Transactions on Communications and IEEE Communications Letters in 2021. His research interests include edge intelligence, generative AI, semantic communications, and network management.

Dusit Niyato (M'09-SM'15-F'17) is a professor in the College of Computing and Data Science, at Nanyang Technological University, Singapore. He received B.Eng. from King Mongkuts Institute of Technology Ladkrabang (KMITL), Thailand and Ph.D. in Electrical and Computer Engineering from the University of Manitoba, Canada. His research interests are in the areas of mobile generative AI, edge intelligence, decentralized machine learning, and incentive mechanism design.