AEDFL: Efficient Asynchronous Decentralized Federated Learning with Heterogeneous Devices

Ji Liu, Tianshi Che, Yang Zhou, Ruoming Jin, Huaiyu Dai, Patrick Valduriez

AEDFL: Efficient Asynchronous Decentralized Federated Learning with Heterogeneous Devices

Ji Liu[†*‡] Tianshi Che[†§] Yang Zhou[*§] Ruoming Jin[¶] Huaiyu Dai[‖] Dejing Dou[**]
Patrick Valduriez[††]

Abstract

Federated Learning (FL) has achieved significant achievements recently, enabling collaborative model training on distributed data over edge devices. Iterative gradient or model exchanges between devices and the centralized server in the standard FL paradigm suffer from severe efficiency bottlenecks on the server. While enabling collaborative training without a central server, existing decentralized FL approaches either focus on the synchronous mechanism that deteriorates FL convergence or ignore device staleness with an asynchronous mechanism, resulting in inferior FL accuracy. In this paper, we propose an Asynchronous Efficient Decentralized FL framework, i.e., AEDFL, in heterogeneous environments with three unique contributions. First, we propose an asynchronous FL system model with an efficient model aggregation method for improving the FL convergence. Second, we propose a dynamic staleness-aware model update approach to achieve superior accuracy. Third, we propose an adaptive sparse training method to reduce communication and computation costs without significant accuracy degradation. Extensive experimentation on four public datasets and four models demonstrates the strength of AEDFL in terms of accuracy (up to 16.3% higher), efficiency (up to 92.9% faster), and computation costs (up to 42.3% lower).

Keywords **–** Federated Learning, Decentralized Machine Learning, Asynchronous Learning, Staleness-Aware Model Update, Sparse Training

1 Introduction

In recent years, a huge amount of data is generated on numerous edge devices, which contain sensitive information of end users, e.g., location information, private images, financial accounts, etc. While the implementation of diverse laws or regulations, e.g., General Data Protection Regulation, hinders the data aggregation, Federated Learning (FL) [20] emerges as an efficient approach to deal with distributed data. A typical distributed FL architecture consists of multiple devices and a centralized parameter server [37], which transfers gradients or models between devices and servers without moving raw data [41]. FL is deployed in multiple applications [15, 56] and various domains [18, 30].

The distributed FL training is typically composed of local training on each device and model aggregation on the server. The server can select available devices and broadcasts a global model to the selected devices. Then, the model is updated based on the local data within each selected device, which is coined local training. After receiving the updated models from the selected devices, the server aggregates them with the global model and generates a new global model, which is denoted as model aggregation. The training process can be either synchronous [41] or asynchronous [47]. With the synchronous FL mechanism, the model aggregation is carried out after receiving all updated models, while, the asynchronous FL enables model aggregation when parts of the models are received.

While edge devices are generally heterogeneous with diverse computation or communication capacities, the synchronous distributed FL corresponds to inferior efficiency due to stragglers, i.e., modest devices [5, 43]. The asynchronous mechanism in FL may lead to inferior accuracy or even fail to converge with non-Independent and Identically Distributed (non-IID) data [47]. In addition, the centralized FL incurs severe communication or computation workload on the server, which becomes a bottleneck and results in low efficiency and a single point of failure [50].

Decentralized FL [44] is proposed to alleviate the communication bottleneck on the central server. Decentralized FL organizes the devices with a connected topology and enables each device to communicate with its neighbors in a peer-to-peer manner. Each device aggregates its local model with the models or gradients transferred from its one-hop neighbors without relying on a central server. Decentralized FL generally inherits from decentralized learning [58], which can be either synchronous [31, 59] or asynchronous [32, 53]. The synchronous mechanism relies on a global clock to synchronize the training process on each device, which corresponds to low efficiency with heterogeneous devices. The asynchronous mechanism enables model aggregation without synchronization, which can well utilize heterogeneous devices while incurring a staleness problem and degrading the accuracy. Existing decentralized approaches generally exploit static weights within the model aggregation process with inferior accuracy.

[‡]J. Liu is with Hithink RoyalFlush Information Network Co., Ltd., Hangzhou, China. [†] equal contribution. [*] corresponding author: jiliuwork@gmail.com, yangzhou@auburn.edu.

[§]T. Che and Y. Zhou are with Auburn University, United States.

[¶]R. Jin is with Kent State University, United States.

[‖]H. Dai is with North Carolina State University, United States.

[**]D. Dou is with Boston Consulting Group, Beijing, China.

[††]P. Valduriez is with Inria, University of Montpellier, CNRS, LIRMM, Montpellier, France and LNCC, Petropolis, Rio de Janeiro, Brazil

As the computation and communication capacities of edge devices are limited, model compression methods, e.g., pruning [62] or sparse training [4] methods, can be exploited to shrink the models so as to reduce the computation and communication costs. Personalized sparse training in decentralized FL [12] is proposed to reduce communication and computation costs with heterogeneous devices. However, the pruning process either degrades the accuracy due to lossy strategies or requires a centralized server. In addition, the decentralized personalized sparse training approach only focuses on the current weights and the local data without considering the interaction with neighbors, which corresponds to inferior accuracy.

In this paper, we propose a novel Asynchronous Efficient Decentralized Federated Learning framework, i.e., AEDFL, for heterogeneous environments. To deal with heterogeneous devices, we enable asynchronous training on each device with a new dynamic model aggregation method. The dynamic model aggregation method consists of a reinforcement learning-based model selection method and a dynamic weight update strategy. In addition, we propose an original adaptive sparse training method to further reduce computation and communication costs so as to improve efficiency with a lossless method based on the consideration of the impacts on both the current loss and the whole training process. The main contributions are summarized as follows:

- We propose an original asynchronous decentralized FL system model with a novel dynamic model aggregation method for collaborative model training with heterogeneous devices. Our proposed dynamic model aggregation method consists of a reinforcement learning-based model selection approach to choose proper models and a dynamic weight update strategy to adjust the weights of each model, which can improve the accuracy.
- We propose a new adaptive sparse training method to reduce the computation and communication costs so as to improve efficiency. The adaptive sparse training method shrinks the model based on the weights of neurons in the model, the gradients, and the values in the Hessian matrix while minimizing the impact on the current loss function to improve the accuracy and considering the impact of the pruning operation in the following training process.
- We conduct extensive experiments to compare AEDFL with representative approaches based on four typical models over four real-world datasets. Experimental results demonstrate the superb advantages of AEDFL in terms of accuracy, efficiency, and computation costs.

The rest of the paper is organized as follows. In Section 2, we present the related work. In Section 3, we present the system model of AEDFL. In Section 4, we propose our dynamic model aggregation method. In Section 5, we propose our sparse training method. In Section 6, we demonstrate the experimental results. Section 7 concludes.

2 Related Work

FL [41] is proposed to train a global model with distributed non-IID data on heterogeneous devices [5]. Numerous model aggregation algorithms [41, 29, 51, 21, 62, 34, 36, 7, 8, 38, 6, 19, 26, 66] are proposed for synchronous FL. The synchronous FL mechanism is inefficient due to stragglers devices [5]. While asynchronous FL [55, 17, 35] can deal with the device heterogeneity, the staleness may degrade the efficiency or the accuracy [47]. Although staleness-based weight discount [54], feature representation adjustment [10], and learning rate adjustment [64] can adjust the training process, they do not consider the direct impact on the loss function or the difference among diverse devices, which results in inferior accuracy.

Decentralized FL [40, 44, 14] enables devices to communicate with their one-hop neighbors in a peer-to-peer manner without a central server. Many decentralized FL [58] approaches directly exploit the decentralized learning techniques [31, 61]. Synchronous [31] decentralized learning synchronizes the training process on each device, which favors homogeneous environments. Asynchronous decentralized learning [32, 53] can deal with heterogeneous resources while incurring staleness problems. Existing decentralized FL approaches [58] are generally synchronous while exploiting static weights within the model aggregation process with inefficiency or accuracy degradation. Asynchronous decentralized FL [39, 27] can deal with heterogeneous devices, while each device needs to send its models to all other devices. This mechanism generates heavy communication overhead when the number of devices is significant. To reduce the communication overhead, exponential topology [1], where each device is connected to the magnitude of $\log(n)$ with n representing the number of devices, can be exploited thanks to its excellent performance [59]. Although grid and ring topologies can be exploited as well, they correspond to low generalization capacity [68].

Pruning techniques [16] can be exploited for sparse training in the FL [2, 4], aiming to reduce computation and communication costs for devices with limited computation and communication capacities. However, existing sparse training approaches [12, 16], which only focus on the weights or ranks of neurons in the model, often lead to reduced accuracy. In addition, the pruning process is usually implemented on models that are not fully trained[2], yet existing approaches, e.g., HAP [60], assume pruning a well-trained model. Furthermore, the pruning process can lead to personalized models [49, 16, 12]. However, current methods often overlook the potential impact of the pruned sections on subsequent training, leading to reduced accuracy. Fi-

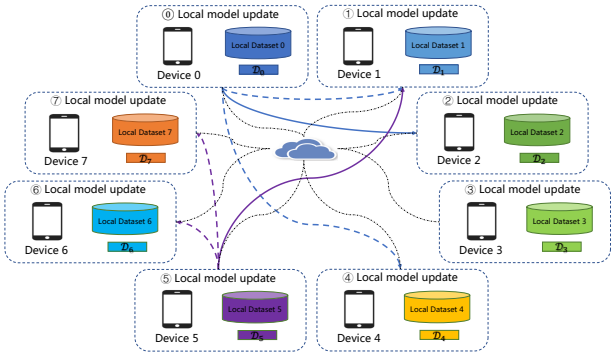

Figure 1: The system model of AEDFL. We consider 8 devices in this figure. Each device has 3 neighbor devices (represents by dashed lines). When a device updates its local model, it sends the model to a randomly sampled neighbor (represented by the solid lines). Each device is coordinated by a coordinator (represented by the dash-dotted lines).

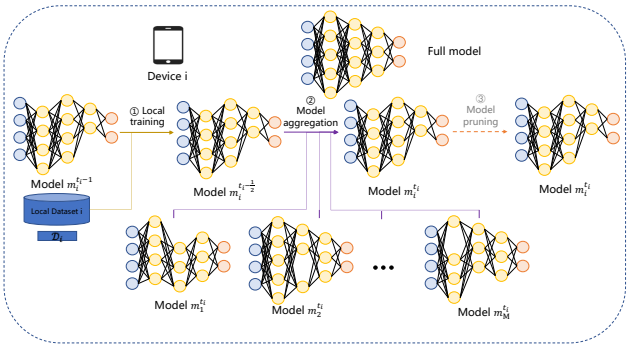

Figure 2: The local update of AEDFL. In each device, the models of all its neighbors are cached. When a neighbor sends its updated model, the corresponding cached model is atomically updated. The model pruning (marked in gray) is carried out in specific local updates. The full model structure is kept during the training process. "M" represents the number of neighbor models.

nally, contemporary sparse training methods, e.g., HAP and DisPFL [12], neglect the impact of the pruning process on the gradients, resulting in significant accuracy degradation during FL's training process.

3 System Model

In this section, we propose the system of AEDFL. First, we detail the system architecture of AEDFL, including the asynchronous communication based on the exponential topology. Then, we present the local update within each device.

As shown in Figure 1, we consider a decentralized FL environment with multiple devices and a coordinator. Please note that although there is a centralized coordinator in the system, it is quite different from a central server. Similar to that in Cassandra [23], the coordinator only manages the index and the heartbeats of each device without participating in the training process or the model aggregation process of FL. Each device is connected to the system and gets an index from the coordinator for the following training process. Each device i has a local dataset $D_i = \{x_i, y_i\}^{s_i}$ with x_i and y_i representing a sample and s_i representing the number of samples on Device i. We denote the number of all the samples by s. Then, the objective of the training process of FL is formulated as follows:

$$\min_{m} \left[\mathcal{F}(m) \triangleq \frac{1}{s} \sum_{i=1}^{n} s_i F_i(m_i) \right], \tag{3.1}$$

where m is the parameters of the whole global model (without pruning), $F_i(m_i) \triangleq \frac{1}{s_i} \sum_{\{x_i, y_i\} \in \mathcal{D}_i} f(m_i, x_i, y_i)$ is the loss function on Device k with $f(m_i, x_i, y_i)$ capturing the error of the local model m_i on the sample $\{x_k, y_k\}$.

During the training process, we exploit an exponential topology [1, 31], where each device has $\mathcal{O}(\log(n))$ neighbors with n representing the number of devices in the system. Please note that the topology is independent of communication network among devices. The topology represents the information transfer among multiple devices. We assume that the devices can communicate with each other. While the exponential topology is exploited in our framework, other topologies can be utilized as well. We define $w_{i,j}$ as the weight to transfer information from Node j to Node i, as follows:

$$w_{i,j} \begin{cases} > 0 & i = j + 2^k, k \in \mathbb{Z}, \text{ or } i = j; \\ = 0 & \text{otherwise.} \end{cases} \tag{3.2}$$

In addition, we further define the topology matrix $\mathcal{W} = [w_{i,j}]_{i,j=0}^{n-1} \in \mathbb{R}^{n \times n}$ as the matrix to represent the topology. The training process of AEDFL consists of three stages. First, each device gets its index from the coordinator. Then, each device performs its local update. Afterward, each device sends its updated model to a randomly selected neighbor. The second and third steps are repeated until satisfying predefined conditions, e.g., a predefined number of iterations on each device is achieved or the consensus distance is smaller than a predefined value. The consensus distance is defined as the average discrepancy of local models between any two nodes. Although each device has $\mathcal{O}(\log(n))$ neighbors, we take advantage of random selection for the model diffusion so as to further reduce communication overhead. Please note that the local update and the model diffusion can be executed in parallel in order to improve efficiency. Furthermore, the local update and the model diffusion are asynchronous and thus independent. The asynchronous training can well alleviate the inefficiency brought by the heterogeneity of devices.

As shown in Figure 2, the local update of AEDFL on each device consists of three steps. First, the local model is updated based on the local dataset exploiting Stochastic

Gradient Descent (SGD) [69]. Afterward, the local model is aggregated with the cached models from its neighbors in the model aggregation process (see details in Section 4). To reduce the model size, in certain local updates, we perform model pruning (see details in Section 5). Finally, an updated model is generated, which is sent to one of its neighbors. At the same time, we create another copy of the updated model, which continues to be updated from the first step. Within each device (i), there are $\mathcal{O}(\log(n))$ cached models. Each model corresponds to the local model of a device (j) such that Device i is a neighbor of Device j, i.e., $\omega_{i,j} > 0$. When Device j sends its model ($m_j^{t_j}$) to Device i, the corresponding cache j on Device i is updated. This process is atomic to ensure that the whole model is available for model aggregation. As a model may take much memory space, e.g., LLaMA [48], the cached model can be either placed on the memory of the GPU (GPU RAM) or the memory of the device (CPU RAM). When the model is placed on the memory of the device, it may take extra time to move the model to the GPU within the model aggregation process. Thus, we first choose the GPU memory by default. When the GPU memory is not enough or there are numerous devices in the system, we take extra memory space of the device for the cached models. The consensus distance can be calculated based on the cached models and the local model.

4 Dynamic Model Aggregation

In this section, we propose our dynamic model aggregation method. First, we propose a novel reinforcement learning-based model selection method. Then, we present a dynamic weight adjustment strategy. Last but not least, we present a model aggregation method to merge diverse models.

In order to achieve the objective defined in Formula 3.1, we dynamically aggregate the local model and the cached neighbor models. We formulate the problem of the local update process as a bi-level optimization problem [3] as defined in Formula 4.3:

$$\min_{m_i,\omega_i} \left[F_i(m_i,\omega_i) \triangleq \frac{1}{s_i} \sum_{(x_i,y_i)\in\mathcal{D}_i} f(agg(m_i,\omega_i),x_i,y_i) \right], \tag{4.3}$$
$$s.t. \sum \omega_i = 1,$$

where $agg(m_i,\omega_i)$ represents the model aggregation process, $\omega_i = \{\omega_{i,i}, \omega_{i,i+1}, \omega_{i,i+2}, ..., \omega_{i,i+2^m}\}$ represents a set of weights for the local model and neighbor models within the model aggregation process and the sum equals to 1.

4.1 Model Selection In this section, we propose a reinforcement learning-based method to select proper neighbor models for model aggregation. While the training process is asynchronous, the cached models within each device are of diverse versions. For instance, Device i is among the neighbor devices of Devices j and j'. Then, the cached models for Devices j and j' are $m_j^{t_j}$ and $m_{j'}^{t_{j'}}$ with t_j and $t_{j'}$ representing the number local updates executed on Device i when receiving $m_j^{t_j}$ and $m_{j'}^{t_{j'}}$, respectively. We denote the number of local updates on Device i by t_i. Then, when $t_j << t_i$, the cached model $m_j^{t_j}$ may not be beneficial within the model aggregation because of stale knowledge. When $t_{j'} \geq t_i$, it is of much possibility to improve the local model of Device i with $m_{j'}^{t_{j'}}$. In addition, some models may be already aggregated in a previous local update, which can be ignored in the following model aggregation. Thus, we construct a reinforcement learning-based model to intelligently select proper neighbor models for the model aggregation process.

The reinforcement learning-based model consists of two modules. The first module is a priority neural network composed of a Long Short-Term Memory network and two fully connected layers. The output of the priority neural network is the priority possibility to choose each neighbor model. The second module is a priority converter, which selects the neighbor models based on the priority possibility.

Within the training process on a device, we define the reward as the loss value calculated based on the local loss function, i.e., $\mathcal{F}_i(\boldsymbol{w}_i^{t_i}, \zeta_i^{t_i})$ with $\zeta_i^{t_i}$ representing the sampled data in SGD. Inspired by [52, 70], we update the priority neural network based on Formula 4.4 as follows:

$$\theta_{t_i+1} = \theta_{t_i} - \eta' \sum_{m=0}^{\log(n)} \nabla_{\theta_{t_i}} \log P(c_m | c_{(m-1):1}; \theta_t)(\mathcal{R}_t - l_i), \tag{4.4}$$

where θ_{t_i} represents the parameters in the priority neural network on Device i at t-th local training, η' refers to the learning rate, $\log(n)$ is the number of cashed neighbor models, c_m corresponds to whether Model m is selected, current local loss ($\mathcal{R}_t$) on device i is the reward, i.e., $\mathcal{R}_t = \mathcal{F}_i(\boldsymbol{w}_i^{t_i}, \zeta_i^{t_i})$, and l_i is a constant value for the bias, i.e., the average loss of the last certain times of local training on Device i. The input of the model includes whether the model is aggregated, the staleness of the model, and the loss value of the model. The model can be pre-trained with some heuristics, e.g., the model that is already aggregated should not be selected for the following model aggregation (which is synthetic model selection data), which correspond to the profiling results from real training process. Then, the model can be updated during the training process of AEDFL. To the best of our knowledge, we are among the first to propose a reinforcement learning-based approach to select the model of neighbors for model aggregation so as to improve accuracy.

4.2 Dynamic Weight Update In this section, we propose our dynamic weight update method. In order to address the bi-level optimization problem defined in Formula 4.3, we propose a dynamic weight update method for the model aggregation process. We exploit the SGD method to update the local model m_i as defined in Formula 4.5 for the model

aggregation on Device i.

$$m_i^{t_i+\frac{1}{2}} \leftarrow m_i^{t_i} - \eta^{t_i} \nabla_{m_i^{t_i}} F_i(m_i^{t_i}, \omega_i^{t_i}), \tag{4.5}$$

where $\nabla_{m_i^{t_i}} F_i(m_i^{t_i}, \omega_i^{t_i})$ represents the gradients in SGD, and η^{t_i} refers to the learning rate. Then, we dynamically update the weights of neighbor models ω_i. We use Formula 4.6 to calculate the importance of neighbor model j:

$$\omega_{i,j}^{'t_i} = \frac{s_j * \lambda_{i,j}^{t_i}}{\sqrt{\triangle t_{i,j}^{t_i}} * loss_j^{t_i}}, \tag{4.6}$$

where $\omega_{i,j}^{'t_i}$ is the importance of the cached model from Device j on Device i, $\lambda_{i,j}^{t_i}$ is a control parameter to be dynamically updated, $\triangle t_{i,j}^{t_i}$ represents the difference between the current number of local update t_i and the number of local updates when the model is updated from Device j, $loss_j^{t_i}$ refers to the loss of the model on Device j. $\triangle t_{i,j}^{t_i}$ can well represent the staleness of the neighbor model of Device j on Device i. While a doubly-stochastic weight matrix can help obtain a consensual solution [61, 45] we calculate the weight of neighbor model j with Formula 4.7.

$$\omega_{i,j}^{t_i} = \frac{\omega_{i,j}^{'t_i}}{\sum_{j=i \text{ or } j\in\mathcal{M}} \omega_{i,j}^{'t_i}}, \tag{4.7}$$

where $\omega_{i,j}^{t_i}$ is the weight of the cached model from Device j on Device i in t_i-th local update, and $\mathcal{M}$ refers to the set of neighbors of Device i. In order to minimize the loss value, we update the control parameter $\lambda_{i,j}^{t_i}$ exploiting formula 4.8.

$$\lambda_{i,j}^{t_i} = \lambda_{i,j}^{t_i-1} - \eta_\lambda \nabla_{\lambda_{i,j}^{t_i-1}} F_i(m_i^{t_i}), \tag{4.8}$$

where η_λ is the learning rate, and $m_i^{t_i}$ is calculated based on Formula 4.9:

$$m_i^{t_i} = \sum_{j=i \text{ or } j\in\mathcal{M}} \omega_{i,j}^{t_i-1} m_j^{t_i-1}. \tag{4.9}$$

The partial derivatives of the loss function on the control parameter are calculated based on Formula 4.10:

$$\begin{aligned} &\nabla_{\lambda_{i,j}^{t_i-1}} F_i(m_i^{t_i}) \\ &= \frac{\sum_{k=i \text{ or } (k\in\mathcal{M} \text{ and } k\neq j)} \omega_{i,k}^{'t_i-1}}{(\sum_{k=i \text{ or } k\in\mathcal{M}} \omega_{i,k}^{'t_i-1})^2} \frac{s_j g_i^{\mathrm{T}} m_j^{t_i-1}}{\sqrt{\triangle t_{i,j}^{t_i-1}} * loss_j^{t_i-1}}, \end{aligned} \tag{4.10}$$

where g_i represents the gradients of $m_i^{t_i}$ on Device i.

4.3 Heterogeneous Model Aggregation In this section, we present our heterogeneous model aggregation method with the updated weights. Because of the pruning process (see details in Section 5), the cached neighbor models may be heterogeneous in terms of structure. We exploit a full model structure as shown in Figure 2 and masks for each neighbor model and local model for the aggregation. A full model structure is an original model without pruning. The mask is utilized to identify which neuron is kept and which neuron is removed after pruning. Then, for each parameter (μ^p) in the full model, we can calculate its aggregated value based on Formula 4.11.

$$\mu_i^p = \frac{1}{\sum_{j=i \text{ or } j\in\mathcal{M}} \omega_{i,j}^{t_i-1} o_j^p} \sum_{j=i \text{ or } j\in\mathcal{M}} \omega_{i,j}^{t_i-1} \mu_{i,j}^p o_j^p, \tag{4.11}$$

where o_j^p is the mask of the model from neighbor Device j with 1 representing the corresponding parameter μ^p is not pruned and 0 representing that μ^p is pruned. After calculating the parameter values in the full model, the local model can be generated by exploiting the local mask.

Algorithm 1 Dynamic Model Aggregation

Input:
i: The index of the device
m_i^0: The initial model on Device i
T: The maximum number of local updates
M: The set of neighbor models

Output:
$m_i^{t_i}$: The global model at Round t_i

1: **for** t_i in $\{1, 2, ..., T\}$ **do**
2: $m_i^{t_i-\frac{1}{2}} \leftarrow$ update $m_i^{t_i-1}$ based on Formula 4.5
3: **for** $j \in \mathcal{M}$ **do**
4: Update $\lambda_{i,j}^{t_i}$ according to Formula 4.8
5: Update $\omega_{i,j}^{t_i}$ according to Formulas 4.6 and 4.7
6: **end for**
7: $m_i^{t_i} \leftarrow$ update $m_i^{t_i-\frac{1}{2}}$ with M according to Formula 4.11
8: **end for**

As shown in Algorithm 1, the dynamic model aggregation algorithm consists of multiple local updates. Within each local update, the local model is updated with the SGD method (Line 2). Then, for each model in $\mathcal{M}$, the control parameter is updated (Line 4) and the corresponding weight value is updated (Line 5). Afterward, the local model and the neighbor models are aggregated based on Formula 4.11 (Line 7). The convergence analysis can be achieved based on the existing theoretical works [61, 32], which is out of the scope of this paper.

5 Sparse Training

We propose a novel adaptive sparse training method to reduce computation and communication costs while minimizing the impact on the loss function and considering the exploration of the sensitivity of neurons, i.e., the impact of the pruned parameters in the next rounds of updates. We assume that at specific rounds of local training, e.g., t_i, we carry out the pruning operation on the whole model, i.e., the model with all the parameters. Within the pruning operation, we denote the modification of the local model m_i on Device i by Δm_i. Then, we denote the impact on the current local loss function by ΔF_i^C, which can be calculated via Taylor expansion as defined in Formula 5.12.

$$\begin{aligned}\Delta F_i^C &= F_i(m_i + \Delta m_i) - F_i(m_i) \\ &= g_i^T \Delta m_i + \frac{1}{2}\Delta m_i^T \mathcal{H} \Delta m_i + \mathcal{O}(||\Delta m_i||^3),\end{aligned} \tag{5.12}$$

where g_i is the gradient, $\mathcal{H}$ represents the Hessian matrix of the model, and $\mathcal{O}(||\Delta m_i||^3)$ corresponds to higher-order items, which can be ignored [60]. We retrieve the Hessian matrix efficiently utilizing the PyHessian library [57] with small computation and storage costs. We associate the pruned parameters (channels) to p and the remained parameters to r. Then, we have:

$$\begin{aligned}\Delta F_i^C &= g_i^T \Delta m_i + \frac{1}{2}\Delta m_i^T \mathcal{H} \Delta m_i \\ &= \begin{pmatrix} g_i^p \\ g_i^r \end{pmatrix}^T \begin{pmatrix} \Delta m_i^p \\ \Delta m_i^r \end{pmatrix} \\ &\quad + \frac{1}{2}\begin{pmatrix} \Delta m_i^p \\ \Delta m_i^r \end{pmatrix}^T \begin{pmatrix} \mathcal{H}^{p,p}, \mathcal{H}^{p,r} \\ \mathcal{H}^{r,p}, \mathcal{H}^{r,r} \end{pmatrix} \begin{pmatrix} \Delta m_i^p \\ \Delta m_i^r \end{pmatrix}.\end{aligned} \tag{5.13}$$

Please note that the pruning operation is carried out during the training process. Thus, we cannot ignore g_i, i.e., $g_i^T \neq 0$. We can have Formula 5.14 minimizing the current impact ΔF_i^C.

$$\begin{aligned}\Delta F_i^C = &\frac{1}{2}(m_i^p)^T \mathcal{H}^{p,p} m_i^p - (g_i^p)^T m_i^p \\ &- \frac{1}{2}(m_i^p)^T \mathcal{H}^{p,r} (\mathcal{H}^{r,r})^{-1} \mathcal{H}^{r,p} m_i^p \\ &- \frac{1}{2}(g_i^r)^T (\mathcal{H}^{r,r})^{-1} g_i^r \\ &+ (g_i^r)^T (\mathcal{H}^{r,r})^{-1} \mathcal{H}^{r,p} m_i^p.\end{aligned} \tag{5.14}$$

However, Formula 5.12 only considers the impact of the current loss value. We further consider the exploration of the pruned parameters by inserting the magnitude of the gradients as Defined in Formula 5.15.

$$\Delta F_i = (1 - \lambda_g)\frac{|\Delta F_i^C|}{|F_i|} + \lambda_g \frac{||\Delta g_i||_2}{||g_i||_2}, \tag{5.15}$$

where $|\cdot|$ represents the absolute value and $0 \leq \lambda_g \leq 1$ is a hyper-parameter. The added term $\lambda_g \frac{||\Delta g_i||_2}{||g_i||_2}$ represents the exploration of the training process, which should be significant at the beginning of the training process and small at the end. When the magnitude of g_i is significant, the model is not well trained and we should pay attention to the exploration. We empirically set $\lambda_g = \frac{||g_i||_2}{\mathcal{C}||g_i||_2^{max}}$, with $\mathcal{C}$ being a hyper-parameter, g_i representing the current gradients and $||g_i||_2^{max}$ is the maximum L^2 norm of the gradients in the previous training process. The setting of λ_g can help reduce the impact of pruned parameters on gradients at the beginning and that on loss at the end so as to improve the accuracy of the model, as the gradients are important to improve the model at the beginning. In practice, we fine-tune the value of $\mathcal{C}$ in the experimentation. As presented in Section 4.3, for a device, although the pruned model is transferred and updated in the local update, the pruned parameters are still updated when they remain in neighbor models. Then, during each pruning process, we consider the whole model as the original model to preserve the parameters that potentially become important.

Furthermore, as pruning rates are critical to the pruning process [63], we automatically generate a proper pruning rate for the pruning operation. With the recent success of lottery ticket for model pruning [2], we exploit a lossless method to calculate the pruning rate. On Device i, we denote the initial model by m_i^o and the difference between the current model and the initial model by $\Delta m_i = m_i^o - m_i$. We sort the eigenvalues of the Hessian matrix $\mathcal{H}(m_i)$ in ascending order, i.e., $\{h_i^p | p \in (1, d_i)\}$ with d_i referring to the rank of the Hessian matrix and p representing the index. Then, we calculate the Lipschitz constant, denoted as $\mathscr{L}_i$, of a benchmark function $F_i'(\Delta m_i) = \mathcal{H}(m_i) - \nabla F_i(\Delta m_i + m_i)$. We take the first eigenvalue h_i^p that satisfies $h_i^{p+1} - h_i^p > 4\mathscr{L}_i$ to calculate a proper pruning rate by $p_i^* = \frac{p}{d_k}$ to achieve lossless pruning [65]. Afterward, we calculate the pruning rate with Formula 5.16 with the consideration of neighbors.

$$p_i = \sum_{j \in \mathcal{M}} \omega_{i,j}^{t_i} p_j + \omega_{i,i}^{t_i} p_i^*, \tag{5.16}$$

where $\omega_{i,j}^{t_i}$ refers to the weights calculated in Section 4.2.

6 Experiments

In this section, we demonstrate evaluation results for AEDFL. First, we present the experimental setup. Then, we show the comparison of AEDFL with 14 baseline approaches. Afterward, we explain the advantages of dynamic model aggregation and the sparse training with the ablation study.

6.1 Setup We consider a decentralized FL system with 100 devices with an exponential graph topology. We exploit 3 datasets, i.e., Emnist-letters (Emnist) [11], CIFAR-10 (CIFAR) [22], Tiny-ImageNet [24], and 3 models, i.e., LeNet-5 (LeNet) [25], Resnet-8 (ResNet) [13], VGG-9 (VGG) [46], for image classification tasks. We further conduct experiments on a Natural Language Processing (NLP) task, i.e., sentiment analysis with the IMDb dataset on TextCNN [67], to demonstrate the generality of AEDFL. We utilize the Dirichlet distribution [28] to partition the data.

We take 14 existing approaches as baselines, i.e., FedAvg [41], FedProx [29], FedNova [51], SAFA [54], Sageflow [43], AD-PSGD [32], FedSA [9], ASO-Fed [10], FedBuff [42], Port [47], Hrank [33], FedAP [62], HAP [60], DisPFL [12]. In addition, we denote the version with Reinforcement Learning-based model selection by AEDFL-RL, the version with Dynamic Weight Update by AEDFL-DWU, and the version with sparse training by AEDFL-P, the version with dynamic model aggregation by AEDFL-RL-DWU, the

Table 1: The accuracy, training time, and computation costs with AEDFL and diverse baseline approaches. "Acc" represents the convergence accuracy. "Time" refers to the training time (s) to achieve a target accuracy, i.e., 0.9 for LeNet with Emnist, 0.6 for ResNet with CIFAR, and 0.17 for VGG with Tiny-ImageNet, 0.17 for ResNet with Tiny-ImageNet, and 0.65 for IMDb with TextCNN. "MFP" represents the computational costs (MFLOPs). "/" represents that training does not achieve the target accuracy.

Method	Emnist & LeNet			CIFAR & ResNet			TinyImageNet & VGG			TinyImageNet & ResNet			IMDb & TextCNN		
	Acc	Time	MFP	Acc	Time	MFP	Acc	Time	MFP	Acc	Time	MFP	Acc	Time	MFP
AEDFL	**0.9326**	**384**	**0.172**	**0.7453**	**314**	**1.29**	**0.2025**	**884**	**61.8**	**0.2696**	**1085**	**368**	**0.7976**	**517**	**0.389**
FedAvg	0.9159	817	0.283	0.6600	3133	1.74	0.1713	5360	73.6	0.2254	3776	452	0.7763	1329	0.652
FedProx	0.9174	696	0.283	0.6136	3935	1.74	0.1726	4357	73.6	0.1645	/	452	0.7625	1289	0.652
FedNova	0.9160	998	0.283	0.6493	3796	1.74	0.1700	4564	73.6	0.2312	3567	452	0.7737	1269	0.652
SAFA	0.9173	800	0.283	0.6751	2050	1.74	0.1701	1742	73.6	0.2305	10101	452	0.7488	1243	0.652
Sageflow	0.9168	767	0.283	0.6755	1614	1.74	0.1740	1742	73.6	0.2323	2863	452	0.7529	1160	0.652
FedSA	0.9158	397	0.283	0.6613	3090	1.74	0.1539	1658	73.6	0.2123	1931	452	0.7746	1105	0.652
ASO-Fed	0.9136	600	0.283	0.6652	1030	1.74	0.1529	1419	73.6	0.2229	1707	452	0.7814	1092	0.652
Port	0.9165	419	0.283	0.6596	1243	1.74	0.1477	1632	73.6	0.2155	2149	452	0.7620	1188	0.652
FedBuff	0.9177	497	0.283	0.6647	3379	1.74	0.1487	1465	73.6	0.2118	1879	452	0.7890	977	0.652
AD-PSGD	0.9129	940	0.283	0.6384	4346	1.74	0.1165	/	73.6	0.1938	1438	452	0.7724	552	0.652
DisPFL	0.9175	402	0.175	0.6473	767	1.48	0.1530	1067	62.5	0.1852	5350	384	0.7448	764	0.587
Hrank	0.9270	394	0.277	0.7121	545	1.46	0.1849	975	62.5	0.2391	2636	384	0.7675	683	0.587
FedAP	0.9280	417	0.278	0.7189	647	1.39	0.1806	936	63.5	0.2381	2991	383	0.7504	1454	0.463
HAP	0.8752	402	0.173	0.6319	572	1.49	0.1632	1010	62.6	0.1794	3947	384	0.7668	928	0.587

version with all three modules by AEDFL and the version without any module by AEDFL-0.

6.2 Evaluation of AEDFL As shown in Table 1, AEDFL corresponds to the highest accuracy and the fastest training speed among all the approaches. AEDFL significantly outperforms other baseline approaches in the whole training process with a small dataset (Emnist) and model (LeNet). The advantage of AEDFL can be up to 5.8% in terms of convergence accuracy, 61.5% in terms of the training time, and 39.1% in terms of computation costs.

When the dataset and the model become complicated, e.g., CIFAR with ResNet or VGG and Tiny-ImageNet with ResNet or VGG, AEDFL significantly outperforms other baseline approaches in terms of the convergence accuracy. At the beginning of the training, AEDFL performs the dynamic adjustment of the weights for neighbor models, which can bring high accuracy in the middle or at the end of the training process. As shown in Table 1, the advantages of AEDFL can be up to 8.5% for FedAvg, 13.2% for FedProx, 9.6% for FedNova, 10% for SAFA, 7% for Sageflow, 8.4% for FedSA, 8% for ASO-Fed, 8.6% for Port, 8.9% for FedBuff, 15.4% for AD-PSGD, 9.8% for DisPFL, 7.7% for Hrank 7.3% for FedAP, and 11.3% for HAP in terms of accuracy. Furthermore, AEDFL corresponds to a short training time. The training time of AEDFL to achieve a target accuracy can be up to 90% (compared with FedAvg), 92% (compared with FedProx), 91.7% (compared with FedNova), 84.7% (compared with SAFA), 82.2% (compared with Sageflow), 89.8% (compared with FedSA), 69.5% (compared with ASO-Fed), 84.9% (compared with Port), 90.7% (compared with FedBuff), 92.8% (compared with AD-PSGD), 71% (compared with DisPFL), 52.2% (compared with Hrank), 56.3% (compared with FedAP), and 62.8% (compared with HAP) shorter. Finally, as the sparse training can well reduce the model size, AEDFL corresponds to much smaller computation costs (up to 42.3% compared with FedAvg, FedProx, FedNova, SAFA, Sageflow, FedSA, ASO-Fed, Port, FedBuff, and AD-PSGD, 12.5% compared with DisPFL, 11.5% compared with Hrank, 6.9% compared with FedAP, and 13.6% compared with HAP), while the corresponding accuracy remains the highest with the shortest time to achieve target accuracy. Meanwhile, the minimal advantages of our approach are significant as well with the most complicated setting (Tiny-ImageNet with ResNet), i.e., 3.1% higher accuracy, 24.5% faster to achieve target accuracy, and 3.9% smaller computational cost.

We further conduct sentiment analysis experiments on the IMDb dataset with TextCNN to demonstrate the generality of AEDFL. As shown in Table 1, the advantages of AEDFL can be up to 5.3% in accuracy, 64.4% in training speed and 40.3% in computation costs. The results reveal that AEDFL can be easily adopted across various tasks.

Furthermore, we carry out the experimentation with different network bandwidth, multiple values of $\mathcal{C}$.

7 Conclusion

In this paper, we propose a novel Asynchronous Efficient Decentralized Federated Learning framework (AEDFL), with three original contributions, i.e., an asynchronous decentralized FL system model, a dynamic model aggregation method consisting of a reinforcement learning-based model selection method and a dynamic staleness-based weight update strategy, and an adaptive pruning method for sparse training. We carry out extensive experiments with four models and four public datasets to demonstrate the significant advantages of AEDFL in terms of accuracy (up to 16.3% higher), efficiency (up to 92.9% faster), and computation costs (up to 42.3% smaller).

References

[1] M. Assran, N. Loizou, N. Ballas, and M. Rabbat, *Stochastic gradient push for distributed deep learning*, in ICML, 2019, pp. 344–353.

[2] S. Babakniya, S. Kundu, S. Prakash, Y. Niu, and S. Avestimehr, *Federated sparse training: Lottery aware model compression for resource constrained edge*, in FL-NeurIPS, 2022, pp. 1–11.

[3] J. F. Bard, *Practical Bilevel Optimization: Algorithms and Applications*, Springer, New York, USA, 1998.

[4] S. Bibikar, H. Vikalo, Z. Wang, and X. Chen, *Federated dynamic sparse training: Computing less, communicating less, yet learning better*, in AAAI, 2022, pp. 6080–6088.

[5] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, et al., *Towards federated learning at scale: System design*, MLSys, (2019), pp. 374–388.

[6] T. Che, J. Liu, Y. Zhou, J. Ren, J. Zhou, V. S. Sheng, H. Dai, and D. Dou, *Federated learning of large language models with parameter-efficient prompt tuning and adaptive optimization*, in EMNLP, 2024, pp. 1–18.

[7] T. Che, Z. Zhang, Y. Zhou, X. Zhao, J. Liu, Z. Jiang, D. Yan, R. Jin, and D. Dou, *Federated fingerprint learning with heterogeneous architectures*, in ICDM, IEEE, 2022, pp. 31–40.

[8] T. Che, Y. Zhou, Z. Zhang, L. Lyu, J. Liu, D. Yan, D. Dou, and J. Huan, *Fast federated machine unlearning with nonlinear functional theory*, in ICML, PMLR, 2023, pp. 4241–4268.

[9] M. Chen, B. Mao, and T. Ma, *Fedsa: A staleness-aware asynchronous federated learning algorithm with non-iid data*, FGCS, 120 (2021), pp. 1–12.

[10] Y. Chen, Y. Ning, M. Slawski, and H. Rangwala, *Asynchronous online federated learning for edge devices with non-iid data*, in Big Data, IEEE, 2020, pp. 15–24.

[11] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik, *Emnist: Extending mnist to handwritten letters*, in IJCNN, IEEE, 2017, pp. 2921–2926.

[12] R. Dai, L. Shen, F. He, X. Tian, and D. Tao, *Dispfl: Towards communication-efficient personalized federated learning via decentralized sparse training*, in ICML, 2022, pp. 1–18.

[13] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in CVPR, 2016, pp. 770–778.

[14] I. Hegedűs, G. Danner, and M. Jelasity, *Decentralized recommendation based on matrix factorization: a comparison of gossip and federated learning*, in ECML PKDD, Würzburg, Germany, 2019, Springer, pp. 317–332.

[15] T.-M. H. Hsu, H. Qi, and M. Brown, *Federated visual classification with real-world data distribution*, in ECCV, Glasgow, UK, 2020, Springer, pp. 76–92.

[16] T. Huang, S. Liu, L. Shen, F. He, W. Lin, and D. Tao, *Achieving personalized federated learning with sparse local models*, arXiv:2201.11380, (2022), pp. 1–19.

[17] D. Huba, J. Nguyen, K. Malik, R. Zhu, M. Rabbat, A. Yousefpour, C.-J. Wu, H. Zhan, P. Ustinov, H. Srinivas, et al., *Papaya: Practical, private, and scalable federated learning*, MLSys, (2022), pp. 814–832.

[18] J. Jiang, Y. Zhou, G. Ananthanarayanan, Y. Shu, and A. A. Chien, *Networked cameras are the new big data clusters*, in HotEdgeVideo, 2019, pp. 1–7.

[19] J. Jin, J. Ren, Y. Zhou, L. Lv, J. Liu, and D. Dou, *Accelerated federated learning with decoupled adaptive optimization*, in ICML, vol. 162, 2022, pp. 10298–10322.

[20] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, and M. B. et al., *Advances and open problems in federated learning*, Foundations and Trends® in Machine Learning, (2021), pp. 1–210.

[21] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, *SCAFFOLD: Stochastic controlled averaging for federated learning*, in ICML, 2020, pp. 5132–5143.

[22] A. Krizhevsky, G. Hinton, et al., *Learning multiple layers of features from tiny images*, Technical report, University of Toronto, 1 (2009), pp. 1–60.

[23] A. Lakshman and P. Malik, *Cassandra: a decentralized structured storage system*, ACM SIGOPS operating systems review, 44 (2010), pp. 35–40.

[24] Y. Le and X. Yang, *Tiny imagenet visual recognition challenge*, CS 231N, 7 (2015), p. 3.

[25] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE, 86 (1998), pp. 2278–2324.

[26] G. Li, Y. Hu, M. Zhang, J. Liu, Q. Yin, Y. Peng, and D. Dou, *Fedhisyn: A hierarchical synchronous federated learning framework for resource and data heterogeneity*, in ICPP, 2022, pp. 1–10. To appear.

[27] J. Li, Y. Shao, K. Wei, M. Ding, C. Ma, L. Shi, Z. Han, and H. V. Poor, *Blockchain assisted decentralized federated learning (blade-fl): Performance analysis and resource allocation*, TPDS, (2021), pp. 2401–2415.

[28] Q. Li, Y. Diao, Q. Chen, and B. He, *Federated learning on non-iid data silos: An experimental study*, arXiv preprint arXiv:2102.02079, abs/2102.02079 (2021), pp. 1–20.

[29] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, *Federated optimization in heterogeneous networks*, in MLSys, 2020, pp. 429–450.

[30] W. Li, F. Milletarì, D. Xu, N. Rieke, J. Hancox, W. Zhu, M. Baust, Y. Cheng, S. Ourselin, M. J. Cardoso, et al., *Privacy-preserving federated brain tumour segmentation*, in MLMI, 2019, pp. 133–141.

[31] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, *Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent*, in NeurIPS, 2017, pp. 1–11.

[32] X. Lian, W. Zhang, C. Zhang, and J. Liu, *Asynchronous decentralized parallel stochastic gradient descent*, in ICML, 2018, pp. 3043–3052.

[33] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, and L. Shao, *Hrank: Filter pruning using high-rank feature map*, in CVPR, 2020, pp. 1529–1538.

[34] J. Liu, J. Huang, Y. Zhou, X. Li, S. Ji, H. Xiong, and D. Dou, *From distributed machine learning to federated learning: a survey*, KAIS, 64 (2022), pp. 885–917.

[35] J. Liu, J. Jia, T. Che, C. Huo, J. Ren, Y. Zhou, H. Dai, and D. Dou, *Fedasmu: Efficient asynchronous federated learning with dynamic staleness-aware model update*, in AAAI, 2024, pp. 1–18. To appear.

[36] J. Liu, J. Jia, B. Ma, C. Zhou, J. Zhou, Y. Zhou, H. Dai, and D. Dou, *Multi-job intelligent scheduling with cross-device federated learning*, TPDS, 34 (2022), pp. 535–551.

[37] J. Liu, Z. Wu, D. Feng, M. Zhang, X. Wu, X. Yao, D. Yu, Y. Ma, F. Zhao, and D. Dou, *Heterps: Distributed deep learning with reinforcement learning based scheduling in heterogeneous environments*, FGCS, (2023).

[38] J. Liu, X. Zhou, L. Mo, S. Ji, Y. Liao, Z. Li, Q. Gu, and D. Dou, *Distributed and deep vertical federated learning with big data*, CCPE, (2023), p. e7697.

[39] Q. Liu, B. Yang, Z. Wang, D. Zhu, X. Wang, K. Ma, and X. Guan, *Asynchronous decentralized federated learning for collaborative fault diagnosis of pv stations*, IEEE TNSE, 9 (2022), pp. 1680–1696.

[40] W. Liu, L. Chen, and W. Zhang, *Decentralized federated learning: Balancing communication and computing costs*, IEEE TSIPN, 8 (2022), pp. 131–143.

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

[42] D. C. Nguyen, Q.-V. Pham, P. N. Pathirana, M. Ding, A. Seneviratne, Z. Lin, O. Dobre, and W.-J. Hwang, *Federated learning for smart healthcare: A survey*, CSUR, (2022), pp. 1–37.

[43] J. Park, D.-J. Han, M. Choi, and J. Moon, *Sageflow: Robust federated learning against both stragglers and adversaries*, in NeurIPS, 2021, pp. 840–851.

[44] Y. Qu, H. Dai, Y. Zhuang, J. Chen, C. Dong, F. Wu, and S. Guo, *Decentralized federated learning for uav networks: Architecture, challenges, and opportunities*, IEEE Network, 35 (2021), pp. 156–162.

[45] W. Shi, Q. Ling, G. Wu, and W. Yin, *Extra: An exact first-order algorithm for decentralized consensus optimization*, SIAM Journal on Optimization, (2015), pp. 944–966.

[46] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, in ICLR, San Diego, CA, USA, 2015, pp. 1–14.

[47] N. Su and B. Li, *How asynchronous can federated learning be?*, in IWQoS, 2022, pp. 1–11.

[48] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, et al., *Llama: Open and efficient foundation language models*, arXiv:2302.13971, (2023), pp. 1–27.

[49] S. Vahidian, M. Morafah, and B. Lin, *Personalized federated learning by structured and unstructured pruning under data heterogeneity*, in ICDCSW, 2021, pp. 27–34.

[50] P. Vanhaesebrouck, A. Bellet, and M. Tommasi, *Decentralized collaborative learning of personalized models over networks*, in AISTATS, 2017, pp. 509–517.

[51] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, *Tackling the objective inconsistency problem in heterogeneous federated optimization*, in NeurIPS, 2020, pp. 7611–7623.

[52] R. J. Williams, *Simple statistical gradient-following algorithms for connectionist reinforcement learning*, Machine Learning, 8 (1992), pp. 229–256.

[53] T. Wu, K. Yuan, Q. Ling, W. Yin, and A. H. Sayed, *Decentralized consensus optimization with asynchrony and delays*, IEEE TSIPN, (2017), pp. 293–307.

[54] W. Wu, L. He, W. Lin, R. Mao, C. Maple, and S. Jarvis, *SAFA: A semi-asynchronous protocol for fast federated learning with low overhead*, TC, (2020), pp. 655–668.

[55] C. Xie, S. Koyejo, and I. Gupta, *Asynchronous federated optimization*, arXiv preprint, abs/1903.03934 (2019).

[56] T. Yang, G. Andrew, H. Eichner, H. Sun, W. Li, N. Kong, D. Ramage, and F. Beaufays, *Applied federated learning: Improving google keyboard query suggestions*, arXiv:1812.02903, (2018).

[57] Z. Yao, A. Gholami, K. Keutzer, and M. W. Mahoney, *Pyhessian: Neural networks through the lens of the hessian*, in Big Data, 2020, pp. 581–590.

[58] H. Ye, L. Liang, and G. Y. Li, *Decentralized federated learning with unreliable communications*, JSTSP, (2022), pp. 487–500.

[59] B. Ying, K. Yuan, Y. Chen, H. Hu, P. Pan, and W. Yin, *Exponential graph is provably efficient for decentralized deep training*, NeurIPS, (2021), pp. 13975–13987.

[60] S. Yu, Z. Yao, A. Gholami, Z. Dong, S. Kim, M. W. Mahoney, and K. Keutzer, *Hessian-aware pruning and optimal neural implant*, in CVPR, 2022, pp. 3880–3891.

[61] K. Yuan, Q. Ling, and W. Yin, *On the convergence of decentralized gradient descent*, SIAM Journal on Optimization, 26 (2016), pp. 1835–1854.

[62] H. Zhang, J. Liu, J. Jia, Y. Zhou, H. Dai, and D. Dou, *Fedduap: Federated learning with dynamic update and adaptive pruning using shared data on the server*, in IJCAI, 2022, pp. 2776–2782.

[63] T. Zhang, X. Ma, Z. Zhan, S. Zhou, C. Ding, M. Fardad, and Y. Wang, *A unified dnn weight pruning framework using reweighted optimization methods*, in DAC, ACM, 2021, pp. 493–498.

[64] W. Zhang, S. Gupta, X. Lian, and J. Liu, *Staleness-aware async-sgd for distributed deep learning*, in IJCAI, 2016, p. 2350–2356.

[65] Z. Zhang, J. Jin, Z. Zhang, Y. Zhou, X. Zhao, J. Ren, J. Liu, L. Wu, R. Jin, and D. Dou, *Validating the lottery ticket hypothesis with inertial manifold theory*, NeurIPS, 34 (2021), pp. 30196–30210.

[66] C. Zhou, J. Liu, J. Jia, J. Zhou, Y. Zhou, H. Dai, and D. Dou, *Efficient device scheduling with multi-job federated learning*, AAAI, 36 (2022), pp. 9971–9979.

[67] Y. Zhou, G. Pu, X. Ma, X. Li, and D. Wu, *Distilled one-shot federated learning*, arXiv:2009.07999, (2021), pp. 1–16.

[68] T. Zhu, F. He, L. Zhang, Z. Niu, M. Song, and D. Tao, *Topology-aware generalization of decentralized sgd*, in ICML, 2022, pp. 27479–27503.

[69] M. Zinkevich, M. Weimer, A. J. Smola, and L. Li, *Parallelized stochastic gradient descent.*, in NeurIPS, vol. 23, 2010, pp. 1–37.

[70] B. Zoph and Q. V. Le, *Neural architecture search with reinforcement learning*, in ICLR, 2017, pp. 1–16.