# FedMP: Federated Learning through Adaptive Model Pruning in Heterogeneous Edge Computing

Zhida Jiang[1,2] Yang Xu*[1,2] Hongli Xu*[1,2] Zhiyuan Wang[1,2] Chunming Qiao[3] Yangming Zhao[3]
[1]School of Computer Science and Technology, University of Science and Technology of China
[2]Suzhou Institute for Advanced Research, University of Science and Technology of China
[3]Department of Computer Science and Engineering, University at Buffalo, The State University of New York

2022 IEEE 38th International Conference on Data Engineering (ICDE) | 978-1-6654-0883-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICDE53745.2022.00062

***Abstract*—Federated learning (FL) has been widely adopted to train machine learning models over massive distributed data sources in edge computing. However, the existing FL frameworks usually suffer from the difficulties of resource limitation and edge heterogeneity. Herein, we design and implement FedMP, an efficient FL framework through adaptive model pruning. We theoretically analyze the impact of pruning ratio on model training performance, and propose to employ a Multi-Armed Bandit based online learning algorithm to adaptively determine different pruning ratios for heterogeneous edge nodes, even without any prior knowledge of their computation and communication capabilities. With adaptive model pruning, FedMP can not only reduce resource consumption but also achieve promising accuracy. To prevent the diverse structures of pruned models from affecting the training convergence, we further present a new parameter synchronization scheme, called Residual Recovery Synchronous Parallel (R2SP), and provide a theoretical convergence guarantee. Extensive experiments on the classical models and datasets demonstrate that FedMP is effective for different heterogeneous scenarios and data distributions, and can provide up to 4.1× speedup compared to the existing FL methods.**

***Index Terms*—Internet of Things, Federated Learning, Adaptive Model Pruning, Heterogeneity.**

## I. Introduction

Recently, billions of Internet of Things (IoT) devices and smartphones around the world produce a significant amount of data every second [1]. The traditional way of uploading those data to the remote cloud for processing will encounter many challenges, including privacy leak, network congestion, and high transmission delay. Since data are usually generated far from the cloud, edge computing (EC) is becoming a natural alternative [2], which performs data processing on edge nodes close to the data sources. According to Cisco's survey, most of IoT-generated data will be stored, processed and analyzed close to or at the network edge [3]. With more data and advanced applications (*e.g.*, autonomous driving, virtual reality), machine learning (ML) tasks will be a dominant workload in EC systems [4]. To alleviate the network bandwidth burden and protect privacy, federated learning (FL) becomes an efficient solution to analyze and process the distributed data on edge nodes for those ML tasks [4], [5].

Among previous FL frameworks, the dominant one is the parameter server (PS) based framework [6] which consists of two components, the PS and the workers. In EC, edge nodes usually act as workers that cooperatively train an ML model using their local data. After each node performs *local update* using stochastic gradient descent (SGD) [4], its local model will be forwarded to the PS for further processing, which is called *global aggregation*. The PS maintains a global model to solve large-scale ML problems.

However, the PS-based framework will encounter some difficulties in practice for the following reasons. 1) *Scarce Communication Bandwidth*. Since the average wide area network (WAN) bandwidth between the PS and workers (*i.e.*, edge nodes) is much (*e.g.*, 15×) lower than the local area network (LAN) bandwidth within the datacenters where the PS resides [7], the frequent communication between the PS and workers will overwhelm the scarce WAN bandwidth, and the communication may be frequently interrupted, slow or expensive. 2) *Limited On-Edge Resources*. Besides the bandwidth, the computation and storage resources of the edges are always limited in EC [5]. On the other hand, the parameter size of deep neural networks (DNNs) may reach tens or even hundreds of megabytes, which requires a huge amount of computation and storage resources for training such a large model [8]. The heavyweight architecture of DNNs makes them computationally expensive with excessive memory requirements and probably results in unbearable delay or breakdown when training DNNs on resource-constrained edges. 3) *Heterogeneous Edge Nodes*. The edge devices may range from universal gateways to specialized base stations, and their communication, computation and storage capabilities are usually heterogeneous. The workers with different capabilities will widen the performance gap between high- and low-performance nodes and deteriorate the training efficiency [9].

Some previous works have made efforts to reduce resource consumption through various solutions, such as parameter quantization [10], [11] and compression [12], [13]. Before global aggregation, these approaches either quantize each parameter with fewer bits, or only send a subset of gradient elements to the PS. Unfortunately, all these works are dedicated to alleviating the total communication cost and do not essentially reduce the model complexity. As a consequence, the computation and storage overhead is still high and may become the principal constraint of these approaches. Moreover, an important factor is ignored in previous studies of efficient FL, *i.e.*, heterogeneity. The communication bandwidth and computation capabilities of different workers may vary significantly and even dynamically in EC [9], [14]. If the PS sends the *identical* model to heterogeneous workers like

2375-026X/22/$31.00 ©2022 IEEE
DOI 10.1109/ICDE53745.2022.00062

[15], [16], the workers with poor capabilities may become the bottleneck of model training and delay the global aggregation. In light of this fact, it is worthwhile to design more effective solutions, which can reduce the resource cost and mitigate the impact of device heterogeneity on FL.

In this paper, we adopt model pruning technique which can reduce the resource demands of DNNs while maintaining the accuracy of the original models [15], [17], [18]. Compared with parameter quantization and compression, model pruning can realize both communication and computation savings. However, the existing pruning methods are proposed for centralized model training and cannot be directly applied to distributed scenarios of FL, as there is no data on the PS for model pre-training [4], [5]. Furthermore, it is challenging to determine different pruning ratios for heterogeneous workers so as to achieve the trade-off between resource efficiency and model accuracy. Especially, when the workers are reluctant to share their private information (*e.g.*, computing power) with the PS for privacy concerns, it will be more difficult to determine the pruning ratios without any prior knowledge of workers' capabilities.

To tackle the above challenges, this paper proposes a novel FL framework, termed FedMP, to improve both computation and communication efficiency in heterogeneous EC. Specifically, we develop an online learning algorithm to adaptively determine appropriate pruning ratios for heterogeneous workers. Given the pruning ratios, the PS performs distributed model pruning and sends the *personalized* sub-models to the workers for local training. In FedMP, each worker only transmits and trains a pruned model under its resource constraints, which can accelerate the training process significantly. The main contributions of this paper are as follows:

- We propose an efficient FL framework, called FedMP, which reduces computation and communication overhead through adaptive model pruning in heterogeneous edge computing. Besides, a theoretical convergence guarantee is provided for FedMP.
- FedMP adopts a Multi-Armed Bandit (MAB) based online learning algorithm to determine the pruning ratios for the workers even without knowing any prior knowledge of their capabilities. The proposed algorithm can adaptively learn the quantitative relationship among pruning ratio, resource consumption and model performance to achieve the trade-off between efficiency and accuracy.
- We also present a novel parameter synchronization scheme called Residual Recovery Synchronous Parallel (R2SP), which recovers the sub-models with diverse structures before model aggregation and ensures comprehensive model updates during the training process.
- We implement FedMP on a physical platform and extensively evaluate FedMP with typical FL tasks. The experimental results show that FedMP is effective for different heterogeneous scenarios and data distributions, and can provide up to 4.1× speedup compared to the existing FL methods.

## II. Background And Motivation

In this section, we first review some works about FL and its variants regarding resource efficiency. We then discuss the disadvantages of traditional model pruning methods for FL in heterogeneous EC, which motivates our design.

### A. Federated Learning

In FL, the workers make full use of the rich data from millions of IoT devices to train DNNs collaboratively while keeping data locally [5]. Considering a set $\mathcal{N} = \{1, 2, ..., N\}$ of workers (*i.e.*, edge nodes) with local datasets $\{D_1, D_2, ..., D_N\}$, the goal of FL is to solve the following distributed optimization problem:

$$\min_{\mathbf{x}} f(\mathbf{x}) \triangleq \frac{1}{N} \sum_{n=1}^{N} f_n(\mathbf{x}) \tag{1}$$

where $\mathbf{x}$ is the model parameter vector and $N$ is the number of workers. The local loss function $f_n(\mathbf{x})$ of worker $n$ is denoted as $f_n(\mathbf{x}) \triangleq \mathbb{E}_{\varphi_n \sim D_n}[F_n(\mathbf{x}; \varphi_n)]$, where $\varphi_n$ is a training sample in the local dataset $D_n$.

A principal advantage of this approach is the decoupling of model training from the need for direct access to the raw training data. However, such a distributed ML framework incurs huge computation and communication overhead, especially training modern DNNs with a large number of parameters [21]. Due to the limited computation and communication resources in EC, the considerable overhead becomes a bottleneck of speeding up the model training [20]. To overcome the bottleneck, some efficient FL solutions have been proposed to alleviate communication overhead by reducing the frequency of global aggregation [4], [5], gradient sparsification [10], [12], [13] and quantization [11], [20].

However, most of the previous works mainly focus on reducing the communication overhead, but ignore the computation efficiency. In fact, the computation capability of edge nodes in EC is also limited compared to that of the data center. It is notoriously expensive for DNNs to perform iterations of SGD. For example, the VGG-16 model [22], which contains 138.34 million parameters, requires 30.94 billion floating-point operations (FLOPs) to classify a single image. Consequently, local updates are often very slow or nearly impossible in modern DNNs due to the large amount of computation overhead. It is necessary to reduce both communication and computation overhead to speed up the training process.

Furthermore, the edge nodes participating in FL are heterogeneous, *i.e.*, with different computation and communication capabilities. The performance of FL will be significantly affected by straggling edge nodes which finish training much slower than others. However, most previous works on FL ignore the computation heterogeneity or communication heterogeneity [13], [19]. As summarized in Table I, compared to the existing works, FedMP can reduce overhead in multiple dimensions such as computation and communication, and provide device-specific solutions considering both computation and communication heterogeneity.

TABLE I
COMPARISON OF DIFFERENT METHODS FOR EFFICIENT FL

| FL Methods | | Efficient Computation | Efficient Communication | Hardware/Library Independence | Computation Heterogeneity | Communication Heterogeneity | Convergence Analysis |
|---|---|---|---|---|---|---|---|
| Aggregation Frequency Optimization | FedAvg [5] | | | ✓ | | | |
| | Wang *et al.* [4] | | ✓ | ✓ | | | ✓ |
| | FedProx [19] | | | ✓ | ✓ | | ✓ |
| Compression of Model Updates | Konečný *et al.* [20] | | ✓ | ✓ | | | |
| | STC [10] | | ✓ | ✓ | | | |
| | Han *et al.* [12] | | ✓ | ✓ | | | ✓ |
| | Caldas *et al.* [16] | ✓ | ✓ | ✓ | | | |
| | Li *et al.* [13] | | ✓ | ✓ | | ✓ | ✓ |
| Model Pruning | Jiang *et al.* [15] | ✓ | ✓ | | | | ✓ |
| | **FedMP** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

### B. Model Pruning

There are usually tens of millions of parameters in modern DNNs, and it could be infeasible to store and train these over-parameterized models on the resource-constrained devices [8]. To address this challenge, model pruning is proposed to reduce the model parameters and resource demands of DNNs while achieving almost similar accuracy of the original model [17], [18], [23]. In general, model pruning approaches can be classified into unstructured pruning and structured pruning. On the one hand, unstructured pruning removes the unimportant weights (*i.e.*, the connections between the neurons) in the neural networks, which results in a high level of parameter sparsity [17]. However, due to the irregularization of the resulting sparse matrix, it is very difficult to compress the parameters in memory, and some specialized hardware/libraries are required to accelerate the model training [24]. On the other hand, structured pruning [18], [25] is designed to directly remove some unimportant model structures (*e.g.*, convolutional filters) without introducing sparsity. Therefore, the resulting model can be regarded as a sub-figure or sub-model of the original neural network, which contains fewer parameters and helps reduce both communication and computation overhead.

Most of the existing model pruning methods are proposed for centralized model training. However, the data in FL is scattered across workers, and the PS does not have access to data for pre-training. Thus these methods cannot be directly applied to distributed scenarios of FL [17], [18]. The most related work [15] studies the application of unstructured model pruning in FL, which needs the support of additional hardware and libraries. Besides, it ignores the heterogeneity of workers and the PS sends the identical pruned model to all workers, thus weak workers will become the bottleneck of model training. To this end, we are inspired to design an efficient FL framework through adaptive model pruning for heterogeneous workers.

## III. PROPOSED FRAMEWORK

In this section, we propose FedMP, an efficient FL framework, to improve both computation and communication efficiency in heterogeneous EC. We first introduce the overview of FedMP, then describe two key phases in detail. Finally, we theoretically analyze the convergence of FedMP.

### A. Overview of FedMP

The model training usually involves a certain number of rounds. As shown in Fig. 1, each round in FedMP consists of the following phases:

① **Adaptive model pruning**: At the beginning of round $k \in \{0, 1, ..., K\}$, the PS adaptively determines the specific pruning ratio $\alpha_n^k$ according to the heterogeneous capabilities of worker $n$ (Section IV). Then the PS prunes the global model $\mathbf{x}^k$ into the sub-models in terms of the pruning ratios. FedMP exploits a distributed method to remove less important filters and neurons from the original model (Section III-B). After model pruning, the PS sends the corresponding sub-model to each worker for local training.

② **Local training**: Within a training round, each worker updates the sub-model over its local dataset via SGD for $\tau$ iterations. The larger the pruning ratio is (*i.e.*, more global model parameters are pruned), the fewer CPU cycles it requires to process a data sample, and the fewer parameters it transmits. When iteration $t \in \{0, 1, ..., I\}$ is a multiple of $\tau$ (*i.e.*, $t \bmod \tau = 0$), model aggregation will be activated, and each worker only uploads its trained sub-model rather than the entire model.

③ **Model aggregation**: After receiving the sub-models from all workers, the PS begins to perform global aggregation. In FedMP, we propose R2SP to recover the sub-models, then derive an updated global model by aggregating the recovered models (Section III-C).

The above procedure repeats until model convergence. Since only the pruned models are transmitted and trained, the computation and communication overhead is reduced significantly compared with training the entire model.

### B. Adaptive Model Pruning Phase

In order to prevent weak workers from blocking the learning process, the PS needs to determine different pruning ratios for heterogeneous workers, which will be elaborated in Section IV. After determining the specific pruning ratio $\alpha_n^k$ for each worker $n$ in round $k$, the PS performs distributed model pruning on the same global model $\mathbf{x}^k$ for each worker, which is different from centralized model pruning methods. FedMP adopts the structured pruning approach rather than the unstructured pruning approach. The sparse network derived by the unstructured pruning approach is difficult to accelerate

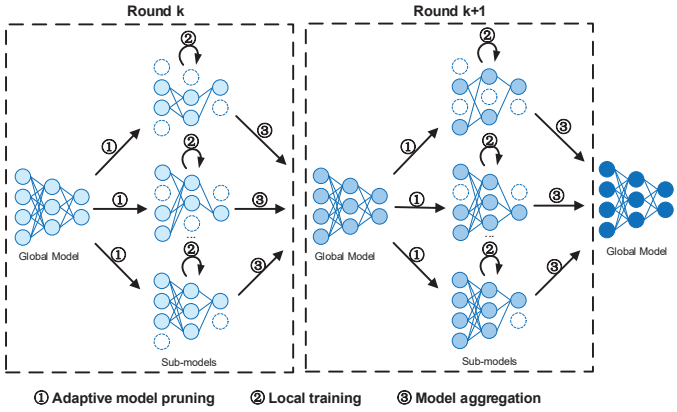



Fig. 1. FedMP overview.

due to the lack of efficient sparse libraries and specialized hardware. Pruning weights does not efficiently reduce the computation time since most removed weights are from the fully-connected layers, but the computation cost is concentrated in the convolutional layers [26].

Specifically, the pruning strategy in FedMP is as follows. To avoid introducing layer-wise hyper-parameters, every layer uses the same pruning ratio [18]. The filters/neurons in the same layer are sorted by their importance scores. Given the pruning ratio, the filters/neurons with lower importance scores will be removed from each layer of the original model. FedMP chooses to use a simple yet effective metric (*i.e.*, $l_1$-norm) to obtain the importance scores of filters/neurons [18]. For each filter in the convolutional layers, we calculate the sum of the absolute kernel weights as the filter's score. Previous studies [17] have shown that low-weight connections have a weak effect on model accuracy. Consequently, for each neuron in the fully-connected layers, we calculate the sum of the absolute weights that the neuron is connected to as the neuron's score. If the filters/neurons have relatively lower scores, they seem less important, and will be removed in terms of the pruning ratio, leading to a more compact network architecture.

When the filters with their feature maps are pruned, the corresponding channels of filters in the next layer are also removed. In addition, if a convolutional layer is pruned, the weights of the subsequent batch normalization layer are removed too. A smaller sub-model $\hat{\mathbf{x}}_n^k$ is created for worker $n$ after distributed model pruning, and the remaining parameters of the modified global model are copied into the sub-model. The sub-models are sent from the PS to workers, then each worker starts local training over its local dataset. These pruned models have much fewer parameters compared to the original global model, thus both resource consumption and memory footprint will be reduced dramatically. Note that the sub-model $\hat{\mathbf{x}}_n^k$ of each worker is related to its pruning ratio $\alpha_n^k$ and may vary with different workers.

### C. Model Aggregation Phase

After the workers complete the sub-model training over the local datasets, the local updates will be sent to the PS that synchronizes the parameters. Synchronization among workers is very critical in FL, and is a costly operation that may significantly reduce the benefits of data parallelism and model parallelism. Since the pruning ratios of different workers change dynamically, the sub-models involved in global aggregation have diverse structures in FedMP. Traditional synchronization schemes in distributed machine learning such as BSP may degrade the overall accuracy [27]. Therefore, we design a new synchronization scheme, called *Residual Recovery Synchronous Parallel (R2SP)* to guarantee the training convergence. We first introduce several key concepts in R2SP.

- **Sparse model** has the same network structure as the global model except that some parameters to be (logically) pruned are set to 0.
- **Sub-model** has a more compact structure compared to the sparse model, as some parameters are pruned physically.
- **Residual model** is obtained by the global model subtracting the sparse model.

We next introduce the workflow of R2SP. In round $k$, the relatively unimportant filters and neurons are selected for worker $n$ according to the pruning ratio $\alpha_n^k$. The kernel weights of these filters and the weights connected to these neurons will be set to zero so as to obtain the sparse model $\mathbf{x}_n^k$. Note that the sparse model $\mathbf{x}_n^k$ is different from the sub-model $\hat{\mathbf{x}}_n^k$ that will be sent to worker $n$. That is, the physically removed parameters in the sub-model are set to 0 in the sparse model. The global model and the sparse model are subtracted to obtain the residual model $\overline{\mathbf{x}}_n^k$, *i.e.*, $\overline{\mathbf{x}}_n^k = \mathbf{x}^k - \mathbf{x}_n^k$, and it works as an auxiliary variable for parameter synchronization. In practice, the indexes of the remaining parameters also need to be recorded for each worker $n$ in R2SP, and we can use a binary vector to store the indexes, resulting in a negligible overhead for the resource-rich PS [7].

After receiving the trained sub-models from all workers, the PS begins to perform global aggregation. R2SP first performs model recovery for each sub-model based on the indexes stored in the PS. The recovered models have the same network structure as the global model. R2SP adds the recovered models and the corresponding residual models, then performs parameter averaging among all workers. Although the PS needs to maintain the residual models in R2SP, their occupied memory will be released after each global aggregation. In the current round, the residual models will not incur significant memory overhead on the PS. It is known that DNNs do not need full floating-point precision for inference [28], [29]. When there are many workers, we can quantize each parameter in residual models with fewer bits to further reduce the memory overhead. The memory occupied by the residual model is only 10-20% of that by the original model, and thus leads to a small overhead for the resource-rich PS, which is usually a cloud or cloudlet in edge computing scenarios [30].

By the iterative pruning and recovery process, the PS maintains a rather complete model structure in the whole training process. With residual models, the filters and neurons pruned in the last round will be recovered during global aggregation, thus they may be trained in the next round. In this way, R2SP

ensures that each model parameter has a chance to be trained. This is the major difference from traditional model pruning methods, which *permanently* remove filters and neurons from the network. The pruned parameters in traditional methods cannot update their weights, leading to a smaller and smaller global model. However, R2SP enables a fully functional model by recovering the sub-models and prevents the removal of important parameters from affecting subsequent training.

### D. Convergence Analysis

In this section, we analyze the convergence property of proposed framework approximately [15]. Most of the existing FL studies provide convergence analysis based on the convex optimization assumption [4], [31], which is not the case for deep learning. This paper presents a provable convergence guarantee for non-convex optimization problems that are solved using distributed SGD. According to R2SP, we define the model average at iteration $t$ as

$$\mathbf{x}^k(t) \triangleq \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n^k(t) + \overline{\mathbf{x}}_n^k) \tag{2}$$

where $\mathbf{x}_n^k(t)$ is the local model of worker $n$ with $t \in [k\tau, (k+1)\tau)$. $k = \lfloor \frac{t}{\tau} \rfloor$ represents the round index where the iteration $t$ is located. When $t$ is a multiple of $\tau$ (*i.e.*, $k = \frac{t}{\tau}$), FedMP performs global aggregation to obtain the new global model $\mathbf{x}^k \triangleq \mathbf{x}^{k-1}(t-1)$, where the iteration $t-1$ is the last iteration of round $k-1$ and the iteration $t$ is the first iteration of round $k$. After obtaining new sub-models by pruning global model, each worker continues the iteration $t$ of SGD. Furthermore, each worker can locally observe unbiased independent stochastic gradients denoted by $\mathbf{G}_n^t = \nabla F_n(\mathbf{x}_n^{k'}(t-1); \varphi_n)$ with $\mathbb{E}_{\varphi_n \sim D_n}[\mathbf{G}_n^t] = \nabla f_n(\mathbf{x}_n^{k'}(t-1))$, and $k' = \lfloor \frac{t-1}{\tau} \rfloor$ [32].

To analyze the convergence of FedMP, we assume that our problem satisfies the following common assumption, which is widely used in previous works [32], [33] on the convergence analysis of distributed SGD.

*Assumption 1:* Assume that the loss function satisfies the following conditions:

1) Smoothness: Each function $f_n(\mathbf{x})$ is smooth with modulus L.

2) Bounded variances and second moments: There exist constants $\sigma > 0$ and $G > 0$ such that

$$\mathbb{E}_{\varphi_n \sim D_n} \|\nabla F_n(\mathbf{x}; \varphi_n) - \nabla f_n(\mathbf{x})\|^2 \leq \sigma^2, \forall \mathbf{x}, \forall n$$
$$\mathbb{E}_{\varphi_n \sim D_n} \|\nabla F_n(\mathbf{x}; \varphi_n)\|^2 \leq G^2, \forall \mathbf{x}, \forall n$$

Considering the largest $t_0 \leq t$ such that $\mathbf{x}^k = \mathbf{x}^{k-1}(t_0)$, we have

$$\mathbf{x}_n^k(t) = \mathbf{x}_n^k - \gamma \sum_{i=t_0+1}^{t} \mathbf{G}_n^i \tag{3}$$

where $\gamma$ is the learning rate. Note that such $t_0$ always exists and $t - t_0 \leq \tau$. Since R2SP adds the recovered sub-models and the corresponding residual models, we have

$$\mathbf{x}^k(t) = \mathbf{x}^k - \gamma \sum_{i=t_0+1}^{t} \frac{1}{N}\sum_{n=1}^{N} \mathbf{G}_n^i \tag{4}$$

The following lemma shows that the deviations between $\mathbf{x}^k(t)$ and $\mathbf{x}_n^k(t)$ can be controlled by selecting proper pruning ratios. In this paper, we use the pruning error to measure how well $\mathbf{x}_n^k$ approximates $\mathbf{x}^k$ after pruning, which is defined as $Q_n^k \triangleq \mathbb{E}[\| \mathbf{x}^k - \mathbf{x}_n^k \|^2]$.

*Lemma 1:* Under Assumption 1, the proposed synchronization scheme ensures

$$\mathbb{E}[\| \mathbf{x}^k(t) - \mathbf{x}_n^k(t) \|^2] \leq 6\gamma^2\tau^2G^2 + 3Q_n^k$$

where $G$ is the constant defined in Assumption 1.

*Theorem 1:* If $0 < \gamma < \frac{1}{L}$, the convergence bound of our proposed framework is

$$\frac{1}{I}\sum_{t=1}^{I}\mathbb{E}[\| \nabla f(\mathbf{x}^{k'}(t-1)) \|^2]$$
$$\leq \frac{2}{\gamma I}(f(\mathbf{x}^0) - f(\mathbf{x}^*)) + \frac{3L^2}{NI}\sum_{t=1}^{I}\sum_{n=1}^{N} Q_n^{k'} + \frac{L}{N}\gamma\sigma^2$$
$$+ 6\gamma^2\tau^2G^2L^2$$

where $k' = \lfloor \frac{t-1}{\tau} \rfloor$ and $\mathbf{x}^*$ denotes the optimal model which minimizes the global loss function $f$.

In Theorem 1, we settle for a weaker notion of convergence and use the expected squared gradient norm to analyze the convergence property due to the non-convex settings, as suggested in [34], [35]. We find that the convergence bound is closely related to the pruning error of each worker. The pruning error reflects the difference between the global model and sub-model after pruning. The fewer parameters the sub-model contains, the larger the pruning error is, leading to a looser convergence bound, and vice versa. Due to space limit, herein, we omit the detailed proof of Lemma 1 and Theorem 1, and the similar proof can be found in [32].

## IV. Algorithm For Pruning Ratio Decision

In this section, we propose an adaptive pruning ratio decision algorithm which is an important part in FedMP. We first formalize the problem and then identify two key challenges in designing such an algorithm. Finally, we propose an online learning algorithm to adaptively determine the pruning ratios for workers.

### A. Problem Formulation

As discussed in Section III, the completion time $T_n^k$ of worker $n$ in round $k$ includes local computation time $T_{n,comp}^k$ and transmission time $T_{n,comm}^k$, *i.e.*,

$$T_n^k = T_{n,comp}^k + T_{n,comm}^k \tag{5}$$

Both $T_{n,comp}^k$ and $T_{n,comm}^k$ are related to the pruning ratios. The larger the pruning ratios are, the more parameters will be removed from the original model, resulting in less transmission time and local computation time.

In the synchronous FL, the total latency is determined by the "slowest" worker. The total time of round $k$ in FedMP is defined as

$$T^k(\alpha_1^k, \alpha_2^k, ..., \alpha_N^k) = \max_n T_n^k \tag{6}$$

By Eq. (6), the completion time $T^k$ of round $k$ depends on the pruning ratios of all workers. Our goal is to determine the optimal pruning ratios for workers so as to minimize the completion time of the whole FL training with accuracy requirement. The optimization problem can be formulated as:

$$\min \sum_{k=0}^{K} T^k(\alpha_1^k, \alpha_2^k, ..., \alpha_N^k)$$
$$s.t. \begin{cases} f(\mathbf{x}^K) < \varepsilon \\ 0 \leq \alpha_n^k < 1, \quad \forall n, \forall k \end{cases} \tag{7}$$

The first inequality guarantees the model accuracy where $\varepsilon$ is the convergence threshold of the global loss function. The second inequality bounds the range of the pruning ratio.

### B. Key Challenges

There are two key challenges in designing an effective algorithm for adaptive pruning ratio decision.

**Challenge 1.** *How should we determine optimal model pruning ratios for workers so as to achieve the trade-off between resource efficiency and training accuracy?* The larger pruning ratio will contribute to less communication and computation overhead, and is more likely to deteriorate the model accuracy. On the contrary, the smaller pruning ratio ensures the model accuracy, but the computation and communication overhead is still high. As shown in Fig. 2, given the same time budget, the accuracy first increases and then decreases with the increase of pruning ratio. When the pruning ratio is relatively small, the pruned model can achieve better accuracy than the original one whose pruning ratio is 0. This is because model pruning with a smaller ratio does not hurt the model accuracy, and can reduce the per-round training time [17], [18]. Thus under the given time, the pruned model can be trained for more rounds than the original model, resulting in better accuracy. However, as the pruning ratio increases, some sensitive and meaningful filters/neurons are probably removed, leading to the negative impact on the test accuracy. According to the experimental results, the pruning ratio decision has a crucial impact on the trade-off between efficiency and accuracy. However, it is difficult to capture the quantitative relationship among model pruning ratio, resource consumption and convergence performance.

**Challenge 2.** *How should we adaptively determine different pruning ratios for heterogeneous workers?* Considering edge heterogeneity, the pruning ratios should be adaptive to the training process of different workers. If the PS determines a uniform pruning ratio or sends a uniform pruned model to all workers, the high-performance workers will compromise to the low-performance ones and perform local training with large pruning ratios, leading to resource waste and accuracy loss. The low-performance workers are still the bottleneck of model training. However, due to the large solution space, it is challenging to simultaneously determine different pruning ratios for heterogeneous workers. Furthermore, the edge nodes probably are unwilling to share their private information such as computing power with the PS. The pruning ratio decision would be more complicated due to the lack of prior knowledge.

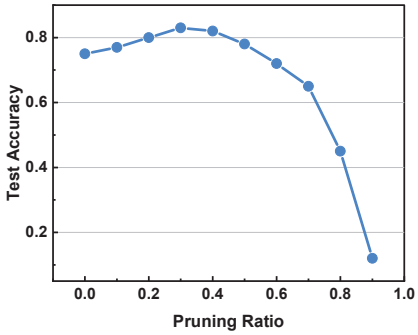


Fig. 2. Effect of different pruning ratios on test accuracy.

### C. Multi-Armed Bandit Based Algorithm

To address the challenges above, we propose a Multi-Armed Bandit (MAB) based online learning algorithm to adaptively determine the pruning ratios for heterogeneous workers without any prior knowledge of their capabilities.

The decision optimization problem can be modeled as an MAB problem, where the PS and the pruning ratios can be regarded as the player and the arms, respectively. In each round, the PS makes the decision about which arm of the bandit is pulled, and a reward in Eq. (8) will be received after the decision. Upper Confidence Bound (UCB) policy is widely used to address the MAB problem [36]. Traditional UCB policy with the discrete arm setting only has a finite set of choices. However, the value range of pruning ratio in FedMP is a continuous space so that the arm space is infinite. In this paper, we extend the UCB policy to the case where the arms are continuous, namely Extended Upper Confidence Bound (E-UCB), as described in Algorithm 1.

Specifically, E-UCB will create agents for the workers participating in FL and use the decision tree to adaptively learn arm space partitions. In round $k$, each agent maintains a sequence of finite partitions of the arm space $\mathbf{P_k} = \{P_k^1, P_k^2, ..., P_k^{l_k}\}$ with $\cup_{j=1}^{l_k} P_k^j = [0, 1)$ and $\mathbf{P_0} = \{[0, 1)\}$ (Line 1), where $l_k$ is the number of partition regions and may vary over time. These partition regions could be viewed as leaves in the decision tree. E-UCB treats the problem as a finite-arm bandit problem with respect to the partition regions, and chooses the partition region that maximizes the upper bound of a confidence interval for the expected reward (Lines 3-5). All arms within the chosen region are treated as the same and selected randomly (Line 6). After determining the pruning ratio $\alpha_n^k$ for worker $n$, the chosen partition region is split into multiple regions to form the new partition $\mathbf{P_{k+1}}$ (Line 8), and the decision tree grows adaptively as the algorithm runs. E-UCB stops extending the decision tree once leaf diameters (region diameters) are below $\theta$ (Line 7), which represents the granularity of pruning ratio exploration. We will show the influence of $\theta$ on the model training process through experiments in Section V. After receiving all sub-models from workers, the PS can get the completion time $T_n^k$ of worker $n$

**Algorithm 1** Extended Upper Confidence Bound (E-UCB) for Worker $n$

1: Initialize $\mathbf{P_0} = \{[0, 1)\}$;
2: **for** Each round $k \in \{0, 1, ..., K\}$ **do**
3: **for** Each partition region $P_k^j \in \mathbf{P_k}$ **do**
4: Calculate upper confidence bound $U_k(P_k^j) = \overline{R}_k(\lambda, P_k^j) + c_k(\lambda, P_k^j)$;
5: Choose the partition region $P_k^j = \arg\max U_k(P_k^j)$;
6: Select the pruning ratio $\alpha_n^k$ from $P_k^j$;
7: **if** The diameter of $P_k^j$ is larger than $\theta$ **then**
8: Split $P_k^j$ with $\alpha_n^k$ to form $\mathbf{P_{k+1}}$;
9: **else**
10: $\mathbf{P_{k+1}} = \mathbf{P_k}$;
11: Record the completion time $T_n^k$ of worker $n$;
12: Calculate the reward $R(\alpha_n^k)$;

(Line 11). The agent will observe a reward from its interactive environment (Line 12), which has a crucial impact on future decisions. The reward in E-UCB can be defined as

$$R(\alpha_n^k) = \frac{\Delta Loss}{|T_n^k - \frac{1}{N}\sum_{n'=1}^{N} T_{n'}^k|} \tag{8}$$

where the numerator indicates the contribution of the workers to model convergence. The denominator represents the gap between the completion time of worker $n$ and the average completion time. A smaller gap means that the selected pruning ratio fits the worker's capabilities better, leading to a higher reward.

To choose the most appropriate partition region, the agent should pursue a balance between the exploitation of arms that perform well in the past and the exploration of arms that might return higher rewards in the future. E-UCB uses the following definition of upper confidence bound to address the *exploitation vs. exploration* challenge.

**(1) Exploitation**. Let $N_k(\lambda, P_k^j) = \sum_{s=1}^{k-1} \lambda^{k-s} \mathbb{1}_{\{\alpha_n^s \in P_k^j\}}$ record the number of times that the partition region $P_k^j$ is chosen, where $\lambda \in (0, 1)$ is a discount factor. The indicator function $\mathbb{1}_{\{\alpha_n^s \in P_k^j\}}$ is 1 when $\alpha_n^s \in P_k^j$ and 0 otherwise. The discounted empirical average is given by

$$\overline{R}_k(\lambda, P_k^j) = \frac{1}{N_k(\lambda, P_k^j)} \sum_{s=1}^{k-1} \lambda^{k-s} R(\alpha_n^s) \mathbb{1}_{\{\alpha_n^s \in P_k^j\}} \tag{9}$$

**(2) Exploration**. If the agent always selects the pruning ratio from the partition region which it currently regards as the best, it might miss another partition region with a higher expected reward. Thus E-UCB adds an exploration item to the upper bound. Let $n_k(\lambda) = \sum_{j=1}^{l_k} N_k(\lambda, P_k^j)$ hold and the discounted padding function is defined as

$$c_k(\lambda, P_k^j) = \sqrt{\frac{2 \log n_k(\lambda)}{N_k(\lambda, P_k^j)}} \tag{10}$$

The upper confidence bound in E-UCB is defined as

$$U_k(P_k^j) = \overline{R}_k(\lambda, P_k^j) + c_k(\lambda, P_k^j) \tag{11}$$

The region $P_k^j$ with the largest $U_k$ will be chosen. The exploitation item averages all past rewards with a discount factor $\lambda$ while giving more weight to recent observations. Furthermore, the exploration item becomes larger if the partition region is not chosen for a long time. The key idea behind E-UCB is that the agent always chooses the best partition region that fits the worker's capabilities based on the estimated reward. The regions with the largest $U_k$ are carefully split to form the better incremental tree while worse regions are not explored anymore. By fitting the decision tree that grows adaptively, the partitions can be learned from last experience.

With the knowledge of heterogeneous capabilities, some more straightforward methods can be used to determine the pruning ratios. However, it is usually impractical for the PS to obtain these private information of workers in edge computing. To this end, E-UCB adaptively learns the partitions and determines the pruning ratios only through the completion time of workers without requiring any prior knowledge of heterogeneous capabilities. The performance of designed arm-pulling policy is measured by regret, which is defined as the difference between the expected reward by playing the optimal arms and that obtained by the given policy. The goal of E-UCB algorithm is to minimize the expected regret, *i.e.*,

$$\lim_{K \to \infty} \frac{1}{K+1} \sum_{k=0}^{K} (R(\alpha^*) - R(\alpha_n^k)) = 0 \tag{12}$$

where $\alpha^*$ is the optimal pruning ratio. Obviously, bounding the expected regret is essentially equivalent to controlling the number of pulling the sub-optimal arms. The algorithm performance can be guaranteed according to [37]. Furthermore, we implement E-UCB by creating an *agent* object for each worker. Each agent uses an incremental regression tree to fit the upper confidence bound function. The agent treats the regions of the arm space as leaves in the incremental regression tree to make pruning ratio decision. The implementation of E-UCB with the incremental regression trees can contribute to low algorithm complexity, and the overhead will be quantified in the experiments of Section V.

### *D. Asynchronous Setting*

As described above, our proposed framework is based on the synchronous setting, where the PS synchronizes the model parameters after receiving all local models. Due to synchronization barrier [19], the round completion time mainly depends on the maximum time among the workers participating in FL, leading to a long waiting time.

FedMP can also accelerate FL under the asynchronous setting, as described in Algorithm 2. In asynchronous FedMP, the PS aggregates the local models only from a fraction (*e.g.*, $m$) of workers according to their arrival order in each round and does not wait for the completion of all workers. The global model is updated by recovering and aggregating the $m$ (out of $N$) first-arrival local models (Section III-C). Furthermore, the agents in E-UCB observe the rewards from the former $m$ workers and the decision tree grows adaptively in each round (Section IV-C), then the pruning ratios of these $m$ workers will

**Algorithm 2** Asynchronous FedMP

Initialize $m$;
**for** Each round $k \in \{0, 1, ..., K\}$ **do**
  **Processing at the PS**
  $\mathcal{N}_k = \varnothing$;
  **while** $|\mathcal{N}_k| < m$ **do**
    Receive local model from worker $n$;
    $\mathcal{N}_k = \mathcal{N}_k \cup \{n\}$;
  Update the global model;
  Determine the pruning ratios for workers in $\mathcal{N}_k$;
  Distribute the pruned models to workers in $\mathcal{N}_k$;
  **Processing at each Worker**
  **if** Receive the pruned model from the PS **then**
    Perform local updating;
    Upload local model to the PS;

be updated. Based on the determined pruning ratios, the PS distributes the pruned models to these $m$ workers. When these $m$ workers participate in global aggregation, the remaining (*i.e.*, $N-m$) workers are still training and uploading their local models, which will be received by the PS in later rounds. Such an asynchronous scheme can deal with the straggler challenge and reduce the latency per round.

## V. Evaluation

### A. Experimental Setup

**Implementation.** We implement a prototype of FedMP by employing an AMAX deep learning workstation as the PS and 30 NVIDIA Jetson TX2 devices as the workers. The workstation is equipped with one Intel Xeon Octa-core E5-2620 v4 processor and 4 NVIDIA GeForce RTX 2080Ti GPUs with 11GB GDDR6 memory each. Every Jetson TX2 has 8GB LPDDR4 main memory and features an NVIDIA Denver2 (dual-core) CPU cluster, an ARM Cortex-A57 (quad-core) CPU cluster and an NVIDIA Pascal GPU with 256 CUDA capable cores. FedMP also provides a fault-tolerant mechanism to handle failed workers. We expect to set a proper deadline so that the PS can receive enough local models from workers and will not significantly increase the duration time of a round. To this end, we record time $d$ when a certain fraction (*e.g.*, 85%) of the local models are received by the PS, then set the deadline of the current round as $1.5d$. If the PS has not received local updates from some workers before the deadline, FedMP will discard these workers, and periodically send messages to ask whether these workers have recovered when the PS is idle. Moreover, FedMP supports new workers joining during training. The joining or leaving of workers will not affect the workflow of FedMP. Our software implementation is based on Pytorch v1.6, but can be easily extended to other ML frameworks such as TensorFlow.

**Heterogeneous workers.** To reflect the heterogeneous computation capabilities of workers, we set different computing modes for each Jetson TX2, as summarized in Table II. From modes 0 to 3, the computation capability decreases gradually.

TABLE II
Different computing modes for Jetson TX2

| Mode | Denver2 (dual-core) | Cortex-A57 (quad-core) | GPU |
|---|---|---|---|
| 0 | 2.0 GHz×2 | 2.0 GHz×4 | 1.30 GHz |
| 1 | — | 2.0 GHz×4 | 1.12 GHz |
| 2 | 1.4 GHz×2 | 1.4 GHz×4 | 1.12 GHz |
| 3 | — | 1.2 GHz×4 | 0.85 GHz |

TABLE III
Accuracy of different FL methods in a given time

| Models | Time Budget | Syn-FL | UP-FL | FedProx | FlexCom | **FedMP** |
|---|---|---|---|---|---|---|
| CNN | 20000s | 93.83% | 94.31% | 95.82% | 96.21% | **97.17%** |
| AlexNet | 30000s | 81.59% | 81.74% | 81.78% | 81.91% | **82.34%** |
| VGG-19 | 50000s | 85.04% | 84.93% | 85.15% | 85.33% | **85.66%** |
| ResNet-50 | 100000s | 47.15% | 46.43% | 47.55% | 47.37% | **47.85%** |

Furthermore, the communication capabilities of workers may also be different in practice. The workers usually connect to the PS via wireless links in EC, and the signal strength of wireless links may vary with the distance [38]. Hence, we place Jetson TX2 devices at different locations to simulate communication heterogeneity. As shown in Fig. 3, based on different computing modes (X-axis) and locations (Y-axis), we partition the devices into three clusters (*i.e.*, $A$, $B$, $C$). For example, the devices in cluster $A$ work at mode 0 or mode 1. By selecting workers from these clusters, we can create different heterogeneous scenarios.

**Models and datasets.** We evaluate the effectiveness of FedMP on four typical FL tasks: (1) *CNN on MNIST*, (2) *AlexNet on CIFAR-10*, (3) *VGG-19 on EMNIST* and (4) *ResNet-50 on Tiny-ImageNet*. The MNIST dataset contains 60,000 training and 10,000 test greyscale images of handwritten digits with size 28×28. The CIFAR-10 dataset includes 60,000 32×32 color images (50,000 for training and 10,000 for testing) of ten different types of objects. The EMNIST dataset is composed of 731,668 training images and 82,587 testing images which draw from 62 classes of objects [39]. The Tiny-ImageNet dataset contains 200 classes, where each class has 500 training images and 50 test images. The dimension of each image is 64×64×3. The CNN has two 5×5 convolutional layers, a fully-connected layer with 256 units, and a softmax output layer with 10 units [4].

**Baselines.** We compare our framework with the following baselines:

(1) *Syn-FL* [5] transmits and trains the entire models. The PS aggregates the parameters after all of the heterogeneous workers have finished local updates.

(2) *UP-FL* [15] determines a uniform pruning ratio for all workers in each round, and the pruning ratio may vary in different rounds.

(3) *FedProx* [19] allows participating workers to perform different numbers of local iterations based on their heterogeneous capabilities.

(4) *FlexCom* [13] considers heterogeneous communication

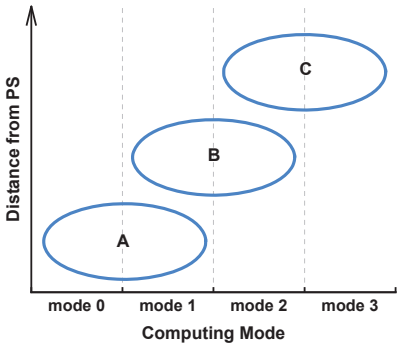



Fig. 3. Worker clusters with different computing modes and locations.

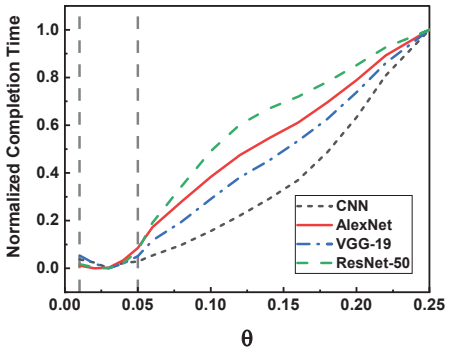



Fig. 4. Effect of pruning granularity $\theta$ on training performance.

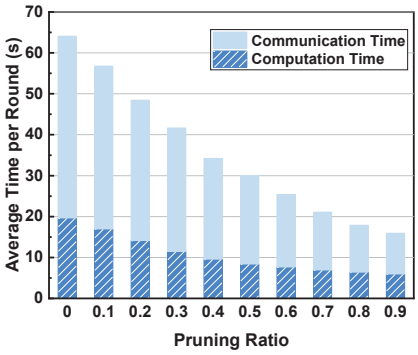



Fig. 5. Effect of different pruning ratios on training time.

condition and enables flexible communication compression, which allows heterogeneous workers to compress the gradients to different levels before uploading.

**Default settings.** Unless noted otherwise, the number of workers is 10 throughout the experiments. Half of them are selected from cluster $A$, and half are selected from cluster $B$. The data samples are assigned to each worker uniformly. We set the discount factor $\lambda$ as 0.95 [40]. For a fair comparison, the baselines are implemented on the same platform as FedMP.

### *B. Effect of Pruning Granularity $\theta$*

The parameter $\theta$ represents the granularity of pruning ratio exploration in E-UCB. In Section IV-C, E-UCB uses the decision tree to adaptively learn arm space partitions and determine the pruning ratios. The decision tree stops growing when region diameters are below $\theta$. To investigate the effect of pruning granularity $\theta$ on training performance, we measure the completion time of different models to reach the target accuracy (*e.g.*, 90% for CNN, 80% for AlexNet, 80% for VGG-19, 45% for ResNet-50) when $\theta$ varies from 0.01 to 0.25. For the sake of comparison, we normalize the completion time, and the results of the four models are shown in Fig. 4.

For all the four models, when the parameter $\theta$ is small (*i.e.*, $\theta \in [0.01, 0.05]$), it only has a minor impact on training performance. As the parameter $\theta$ gets large (*i.e.*, $\theta \in (0.05, 0.25]$), the completion time increases drastically. The reason lies in that large pruning granularity (*i.e.*, larger $\theta$) may cause the pruning ratio selected in a large range, leading to poor training performance. Therefore, we tend to explore the pruning ratio with a relatively small granularity. On the other hand, since model parameters have significant redundancy [18], if the pruning granularity is small enough, it does not affect the completion time too much. That is, $\theta \in [0.01, 0.05]$ achieves almost the same performance, which is demonstrated for different models. Motivated by this, choosing $\theta$ from $[0.01, 0.05]$ is modest for different scenarios.

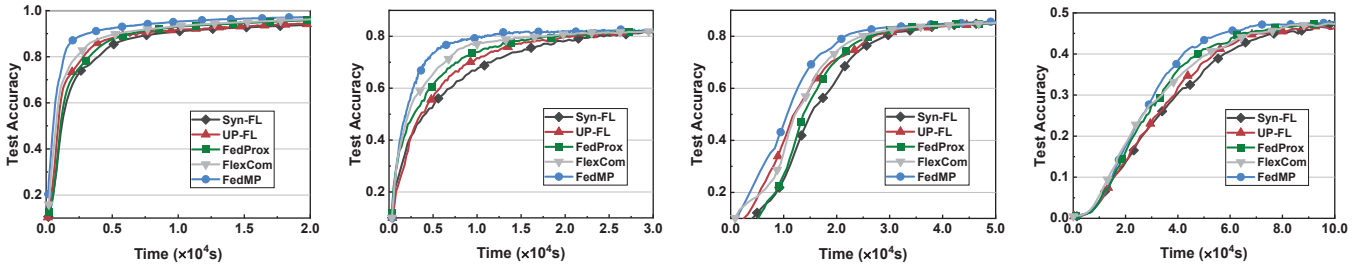
### *C. Overall Effectiveness*

We investigate the performance of FedMP and baselines when they are deployed across heterogeneous workers. Fig. 5 shows that the average per-round time of computation and communication decreases with the increase of pruning ratio. The reason lies in that model pruning significantly reduces the complexity of DNNs, leading to time reduction in computing and delivering local updates. Table III compares the test accuracy that different FL methods can achieve in a given time. We note that FedMP always converges to the similar accuracy as Syn-FL, and achieves higher accuracy in a given time on all the four models. It indicates that distributed model pruning with appropriate pruning ratios does not hurt the model accuracy and can achieve the goal of effective FL.

Fig. 6 shows the test accuracy with time passed on different datasets. From these results, we make three major observations. Firstly, FedMP substantially outperforms Syn-FL. For example, FedMP takes 10,906s to achieve 80% accuracy for AlexNet on CIFAR-10, while Syn-FL takes 24,017s. FedMP provides 2.2× speedup compared to Syn-FL. Since Syn-FL does not adopt any parameter reduction method, the over-parameterized DNNs incur a great deal of computation and communication overhead. In contrast, the pruned models are transmitted and trained in FedMP, which reduce both communication and computation time while ensuring the accuracy.

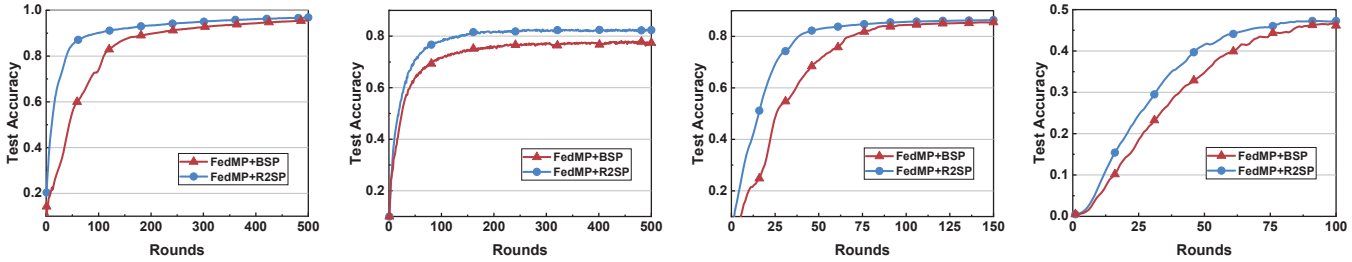
Secondly, FedMP converges much faster than UP-FL on all the four datasets, and achieves a speedup of nearly 2× for AlexNet on CIFAR-10. This is because UP-FL accelerates model training by uniform model pruning, but ignores the heterogeneity of edge nodes. The pruned models are the same across workers, so that the workers with weak capabilities delay the global aggregation. However, FedMP adaptively prunes the global model according to the heterogeneity of edge nodes. The pruned models are customized for workers' capabilities, which accelerates the DNNs training by a significant margin.

Thirdly, FedMP has a clear and consistent advantage on training speed over the other heterogeneity-aware baselines (*i.e.*, FedProx and FlexCom). Particularly, FedMP improves the performance over FedProx by 2.0× for CNN, 1.8× for AlexNet, 1.2× for VGG-19, and 1.1× for ResNet-50. Compared to FlexCom, FedMP achieves about 1.8×, 1.6×, 1.2× and 1.2× speedup, correspondingly. Although FedProx allows heterogeneous workers to perform different numbers of local iterations, it does not incorporate model compression or pruning techniques. As a result, the computation and




(a) CNN on MNIST (b) AlexNet on CIFAR-10 (c) VGG-19 on EMNIST (d) ResNet-50 on Tiny-ImageNet

Fig. 6. Test accuracy of different FL methods.




(a) CNN on MNIST (b) AlexNet on CIFAR-10 (c) VGG-19 on EMNIST (d) ResNet-50 on Tiny-ImageNet

Fig. 7. Test accuracy of different synchronization schemes.

communication overhead in FedProx is still large, leading to poor training efficiency. FlexCom reduces the communication overhead by assigning different compression ratios to heterogeneous workers. However, FlexCom cannot reduce the computation overhead, and each worker trains the same local model, which cannot cope with the computation heterogeneity. Consequently, local training is still very slow due to the large amount of computation overhead. By contrast, the excellent performance of FedMP can be explained by the superiority of adaptive model pruning, which allows each worker to train and transmit the pruned model fitting its own capability. Therefore, FedMP can reduce both computation and communication overhead, thereby significantly accelerating the training process compared with the baselines.

### D. Effect of Synchronization Scheme

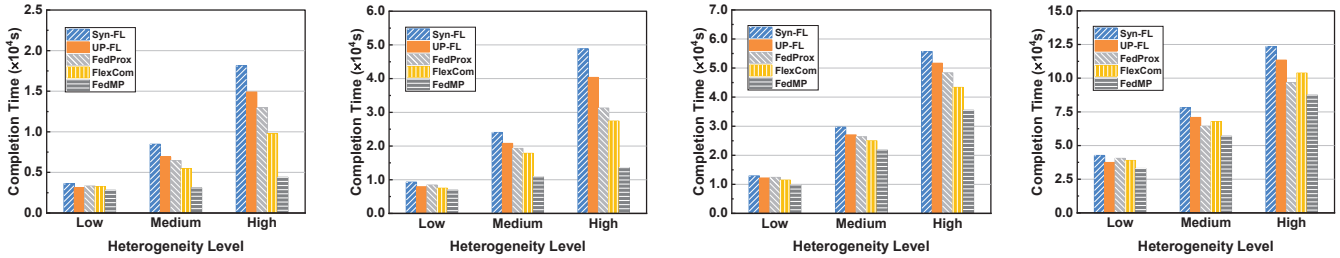
One of the major design considerations of R2SP is to prevent the diverse structures of sub-models from affecting model convergence. To evaluate the effectiveness of our proposed synchronization scheme, we compare R2SP with the traditional BSP scheme on FedMP across heterogeneous workers. Fig. 7 shows the training progress with respect to the number of rounds on different models.

It is obvious that R2SP achieves better convergence performance than BSP on all the four datasets, while BSP may hurt the final model accuracy. For example, FedMP with R2SP achieves 82.3% accuracy after 500 rounds, while FedMP with BSP only achieves 77.4% accuracy for AlexNet on CIFAR-10. The reason lies in that the pruned parameters in R2SP will be recovered during global aggregation and have a chance to be updated in the following training. Such a synchronization scheme makes the PS maintain a complete global model from the perspective of multi-round training, ensuring comprehensive model updates. However, BSP cannot recover the pruned parameters, hence the model accuracy may be degraded if important parameters are removed. These results demonstrate that suboptimal synchronization schemes lead to worse model quality, and R2SP is effective and critical for distributed model pruning in FL.

### E. Effect of Heterogeneity

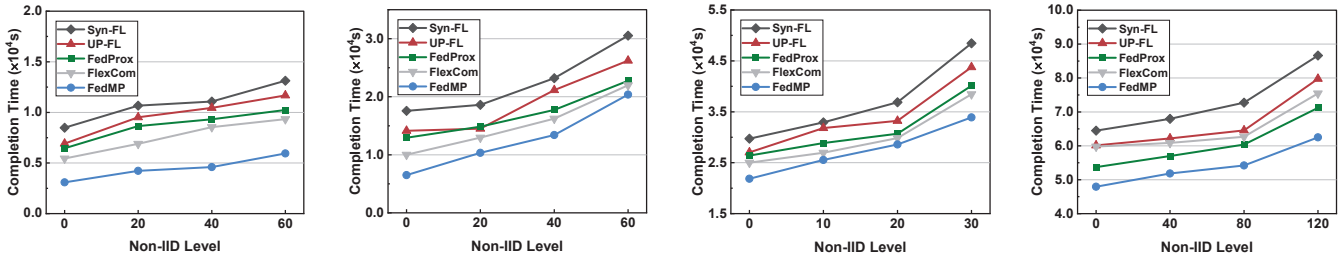
To understand how FedMP performs under heterogeneous scenarios, we deploy FedMP and baselines across the workers with different heterogeneity levels, *i.e.*, *Low*, *Medium*, *High*. For *Low*, we select 10 workers from cluster $A$. For *Medium*, we select 5 workers from cluster $A$, and 5 workers from cluster $B$. For *High*, we select 3 workers from cluster $A$, 3 workers from cluster $B$, and 4 workers from cluster $C$. For our experiments, the desired target accuracy of CNN, AlexNet, VGG-19 and ResNet-50 is 90%, 80%, 80% and 45%, respectively. Fig. 8 shows the required time to reach the target accuracy under different heterogeneous scenarios.

We observe that, from *Low* to *High*, the required time to reach the target accuracy increases accordingly. This is expected because less capable workers are introduced into the system. In the synchronous FL, the PS performs global aggregation after receiving all the local models. Thus the total latency is determined by the "slowest" worker, increasing the completion time of all methods. Nevertheless, FedMP still takes less time to reach the target accuracy compared with the baselines. When training AlexNet on CIFAR-10 in the heterogeneity level of *High*, FedMP achieves 3.6×, 3.0×, 2.3×, and 2.0× speedup compared to Syn-FL, UP-FL, FedProx, and FlexCom, respectively. Moreover, the performance gap can be enlarged with the increase of heterogeneity level. For example, FedMP improves the performance over Syn-FL by




(a) CNN on MNIST (b) AlexNet on CIFAR-10 (c) VGG-19 on EMNIST (d) ResNet-50 on Tiny-ImageNet

Fig. 8. Completion time under different heterogeneous scenarios.




(a) CNN on MNIST (b) AlexNet on CIFAR-10 (c) VGG-19 on EMNIST (d) ResNet-50 on Tiny-ImageNet

Fig. 9. Completion time under different non-IID levels.

1.3× in *Low*, 2.8× in *Medium*, and 4.1× in *High* for CNN on MNIST. This is because FedMP determines the appropriate pruning ratios for the weak workers that are newly introduced into the system, so that each worker can still train the sub-model that best suits its capability, leading to a slight increase in completion time. These results demonstrate that FedMP is robust for different heterogeneous scenarios and can fully utilize the limited resources of workers.

### *F. Effect of non-IID Data*

Considering that the workers in FL collect data from their physical locations directly, the data samples among workers are usually not independent and identically distributed (non-IID). While the previous experiments are based on uniform data partitioning, we now discuss the impact of non-IID data on training performance. As a measurement of non-IIDness in data distribution among workers, we use $y$ to define the non-IID levels.

- For MNIST and CIFAR-10: The non-IID level of $y$ indicates that $y\%$ of the data on each worker belong to one label and the remaining data belong to other labels [41]. As a special case, we use the non-IID level of 0 to denote IID data distribution among workers.
- For EMNIST and Tiny-ImageNet: The non-IID level of $y$ indicates that each worker lacks $y$ classes of data samples, which is similar with the setting of [42].

Fig. 9 shows the required time for FedMP and baselines to achieve the target accuracy in different non-IID levels. We set the target accuracy of CNN, AlexNet, VGG-19 and ResNet-50 as 90%, 77%, 80%, and 42%, respectively. From the results, we note that the required time for all methods to achieve the target accuracy increases with the increase of non-IID levels. Since the local models trained on non-IID data are different from each other, aggregating these divergent models may degrade the training performance and result in more communication rounds until convergence. Nevertheless, for all the four datasets, FedMP still outperforms the baselines in different non-IID levels. For example, even in the non-IID level of 30, FedMP can reduce the completion time by 30%, 23%, 16% and 12% compared to Syn-FL, UP-FL, FedProx and FlexCom for VGG-19 on EMNIST. These results demonstrate the effectiveness of our framework even under non-IID data distribution.

### *G. Effect of Worker Number*

We evaluate the scalability of FedMP with different numbers of workers. In this set of experiments, half of the workers participating in FL are selected from cluster $A$, and half are from cluster $B$. We compare the required time of FedMP and baselines to achieve the target accuracy with the number of workers varying from 10 to 30. The target accuracy is the same as for Fig. 8(b). The results of training AlexNet on CIFAR-10 are shown in Fig. 10. The completion time of FedMP increases slightly with the increasing number of workers. When the number of workers is 30, FedMP still provides 2.4×, 2.0×, 2.0×, and 1.6× speedup compared to Syn-FL, UP-FL, FedProx, and FlexCom, indicating that our design is superior in scalability over other baselines.

Next, we quantify the algorithm overhead which plays a critical role in practical deployment. We measure the average per-round algorithm overhead including the pruning ratio decision time and model pruning time. The results for different numbers of workers are shown in Fig. 11. Apparently, the time overhead increases with the increasing number of workers. However, the maximum time overhead in our experiments is

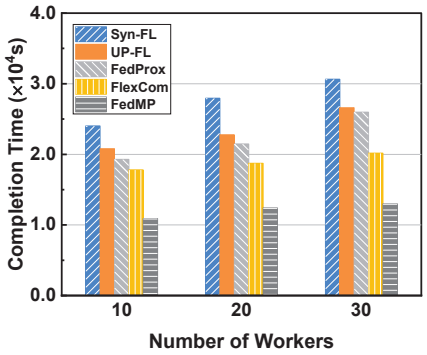



Fig. 10. Completion time with different numbers of workers.

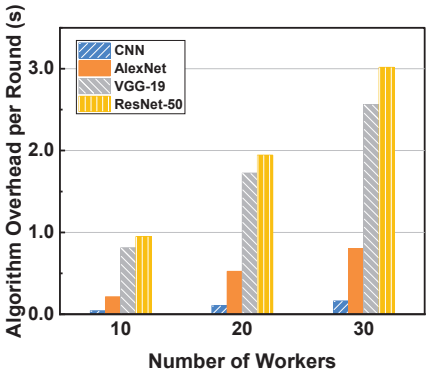



Fig. 11. Average algorithm overhead with different numbers of workers.

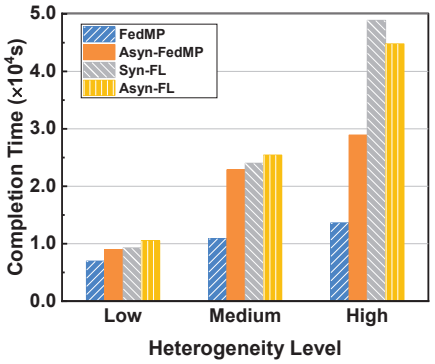



Fig. 12. Completion time under both synchronous and asynchronous settings.

far less than the transmission time and local training time (*e.g.*, hundreds of seconds), thus can be ignored. These results show that FedMP can be deployed in large-scale scenarios, with a small overhead in exchange for substantial training speedup.

### *H. Evaluation of Asynchronous Setting*

To evaluate the performance of FedMP under the asynchronous setting (referred to as *Asyn-FedMP*), we deploy FedMP on 10 heterogeneous workers and set $m = 5$. Fig. 12 shows the required time of different methods while reaching the target accuracy for AlexNet on CIFAR-10. *Asyn-FL* represents the asynchronous FL [43]. Asyn-FedMP can reduce the completion time by 10%∼35% for reaching the target accuracy compared to Asyn-FL, demonstrating the high effectiveness of our design for the asynchronous FL. Furthermore, FedMP outperforms Asyn-FedMP because FedMP aggregates the sub-models from all workers and can get more information compared to Asyn-FedMP.

## VI. Discussion

In this section, we discuss the adaptability and potential extensions of FedMP. We can extend FedMP to accommodate diverse neural networks by easily replacing different pruning strategies. In extended FedMP, the PS still determines different pruning ratios for heterogeneous workers even without knowing any prior knowledge of their capabilities. Given the pruning ratios, the PS performs distributed model pruning based on the specialized strategy so that each worker only trains and transmits a pruned model suiting its own capability. Under these circumstances, FedMP can still reduce computation and communication overhead for different models.

We take Recurrent Neural Networks (RNNs) as an example to show the applicability of FedMP to other models. Compared with Convolutional Neural Networks (CNNs), pruning RNNs is more challenging. Since a recurrent unit is shared across all the time steps in sequence, independently removing the unit will result in mismatch of their dimensions and then inducing invalid recurrent units. Following the intrinsic sparse structure method [44], we update the pruning strategy in Section III-B and keep other designs unchanged. Specifically, we remove weights associated with one component of intrinsic sparse structures, and then the sizes/dimensions (of basic structures) are simultaneously reduced by one. As a result, the obtained RNNs have the original schematic with dense connections but with smaller sizes of these basic structures.

TABLE IV
Perplexity of different FL methods in a given time and speedup for reaching target perplexity

| Methods | Perplexity (validate, test) | Speedup |
|---|---|---|
| Syn-FL | (156.35, 148.15) | 1.0× |
| UP-FL | (158.94, 149.81) | 0.8× |
| FedMP | (155.47, 146.95) | 1.6× |

To study the benefits of FedMP, we train a RNN with two stacked LSTM layers on the Penn TreeBank dataset [45]. The performance of the models is measured by the metric of perplexity, which is the exponent of cross-entropy loss. The results are reported in Table IV. We note that FedMP has lower perplexity in a given time compared to baselines and provides 1.6× speedup for reaching the target test perplexity (*e.g.*, 150). These results prove that FedMP can achieve efficient FL for diverse neural networks.

## VII. Conclusion

In this paper, we design and implement FedMP, which performs federated learning through adaptive model pruning. Specifically, we adopt a structured model pruning approach for federated learning so as to simultaneously reduce computation and communication overhead. We then propose an MAB based online learning algorithm to adaptively determine the pruning ratios for different workers to conquer their heterogeneity. Extensive experiments on the classical models and datasets show the high effectiveness of FedMP.

## Acknowledgment

Yang Xu and Hongli Xu are corresponding authors. This article is supported in part by the National Key Research and Development Program of China (Grant No. 2021YFB3301501), and the National Science Foundation of China (NSFC) under Grants 62132019, 61936015, 62102391 and U1709217.

## REFERENCES


[1] K. L. Lueth, "State of the iot 2018: Number of iot devices now at 7b– market accelerating," *IoT Analytics*, vol. 8, 2018.

[2] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 869–904, 2020.

[3] C. V. Networking, "Cisco global cloud index: Forecast and methodology, 2015-2020. white paper," *Cisco Public, San Jose*, 2016.

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

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

[6] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, 2014, pp. 583–598.

[7] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu, "Gaia: Geo-distributed machine learning approaching LAN speeds," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, 2017, pp. 629–647.

[8] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of internet services and applications*, vol. 1, no. 1, pp. 7–18, 2010.

[9] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[10] F. Sattler, S. Wiedemann, K. R. Muller, and W. Samek, "Robust and communication-efficient federated learning from non-i.i.d. data," *IEEE Transactions on Neural Networks and Learning Systems*, vol. PP, no. 99, pp. 1–14, 2019.

[11] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *Journal of Machine Learning Research*, vol. 18, 2016.

[12] P. Han, S. Wang, and K. K. Leung, "Adaptive gradient sparsification for efficient federated learning: An online learning approach," *arXiv preprint arXiv:2001.04756*, 2020.

[13] L. Li, D. Shi, R. Hou, H. Li, M. Pan, and Z. Han, "To talk or to work: Flexible communication compression for energy efficient federated learning over heterogeneous mobile edge devices," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[14] B. Luo, X. Li, S. Wang, J. Huang, and L. Tassiulas, "Cost-effective federated learning design," *arXiv preprint arXiv:2012.08336*, 2020.

[15] Y. Jiang, S. Wang, B. J. Ko, W.-H. Lee, and L. Tassiulas, "Model pruning enables efficient federated learning on edge devices," *arXiv preprint arXiv:1909.12326*, 2019.

[16] S. Caldas, J. Konečný, H. B. McMahan, and A. Talwalkar, "Expanding the reach of federated learning by reducing client resource requirements," *arXiv preprint arXiv:1812.07210*, 2018.

[17] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[18] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[19] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *arXiv preprint arXiv:1812.06127*, 2018.

[20] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[21] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[22] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[23] M. Kang and B. Han, "Operation-aware soft channel pruning using differentiable masks," in *International Conference on Machine Learning*. PMLR, 2020, pp. 5122–5131.

[24] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: efficient inference engine on compressed deep neural network," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.

[25] S. Lin, R. Ji, Y. Li, Y. Wu, F. Huang, and B. Zhang, "Accelerating convolutional networks via global & dynamic filter pruning." in *IJCAI*, 2018, pp. 2425–2432.

[26] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," in *International conference on artificial neural networks*. Springer, 2014, pp. 281–290.

[27] B. Yuan, A. Kyrillidis, and C. M. Jermaine, "Distributed learning of deep neural networks using independent subnet training," *arXiv preprint arXiv:1910.02120*, 2019.

[28] Y. Boo, S. Shin, and W. Sung, "Memorization capacity of deep neural networks under parameter quantization," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 1383–1387.

[29] D. Lin, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *International conference on machine learning*. PMLR, 2016, pp. 2849–2858.

[30] Z. Ma, Y. Xu, H. Xu, Z. Meng, L. Huang, and Y. Xue, "Adaptive batch size for federated learning in resource-constrained edge computing," *IEEE Transactions on Mobile Computing*, 2021.

[31] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of fedavg on non-iid data," *arXiv preprint arXiv:1907.02189*, 2019.

[32] H. Yu, S. Yang, and S. Zhu, "Parallel restarted sgd with faster convergence and less communication: Demystifying why model averaging works for deep learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 5693–5700.

[33] J. Wang and G. Joshi, "Cooperative sgd: A unified framework for the design and analysis of communication-efficient sgd algorithms," *arXiv preprint arXiv:1808.07576*, 2018.

[34] 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 *Advances in Neural Information Processing Systems*, 2017, pp. 5330–5340.

[35] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "Qsgd: Communication-efficient sgd via gradient quantization and encoding," in *Advances in Neural Information Processing Systems*, 2017, pp. 1709–1720.

[36] G. Gao, J. Wu, M. Xiao, and G. Chen, "Combinatorial multi-armed bandit based unknown worker recruitment in heterogeneouscrowdsensing," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020.

[37] T. Wang, W. Ye, D. Geng, and C. Rudin, "Towards practical lipschitz bandits," in *Proceedings of the 2020 ACM-IMS on Foundations of Data Science Conference*, 2020, pp. 129–138.

[38] D. Tse and P. Viswanath, *Fundamentals of wireless communication*. Cambridge university press, 2005.

[39] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik, "Emnist: Extending mnist to handwritten letters," in *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 2921–2926.

[40] A. Garivier and E. Moulines, "On upper-confidence bound policies for non-stationary bandit problems," *arXiv preprint arXiv:0805.3415*, 2008.

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

[42] D. Li and J. Wang, "Fedmd: Heterogenous federated learning via model distillation," *arXiv preprint arXiv:1910.03581*, 2019.

[43] Q. Ma, Y. Xu, H. Xu, Z. Jiang, L. Huang, and H. Huang, "Fedsa: A semi-asynchronous federated learning mechanism in heterogeneous edge computing," *IEEE Journal on Selected Areas in Communications*, 2021.

[44] W. Wen, Y. He, S. Rajbhandari, M. Zhang, W. Wang, F. Liu, B. Hu, Y. Chen, and H. Li, "Learning intrinsic sparse structures within long short-term memory," *arXiv preprint arXiv:1709.05027*, 2017.

[45] M. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of english: The penn treebank," 1993.