Federated Knowledge Expansion for Collections of Heterogeneous Devices

Xiaopeng Jiang * Cristian Borcea †
* School of Computing, Southern Illinois University, Carbondale, IL USA
† Department of Computer Science, New Jersey Institute of Technology, Newark, NJ, USA
* Email:xiaopeng.jiang@siu.edu † Email:borcea@njit.edu

***Abstract*—Federated Learning (FL) enables privacy preserving training across devices, including PCs, smartphones, and IoT devices. However, when heterogeneous devices collaborate to train a model, devices with the lowest resources throttle the model size and complexity, as FL communicates and trains the same global model across all devices. To address this limitation and train a large model while utilizing all computational power in a system with heterogeneous devices, we propose Federated Knowledge Expansion (FedKE). In FedKE, devices with similar resources form FL groups, which gradually expand a small model with additional computational layers or blocks, progressing toward a large model. Each group trains a model suitable for its resources and then passes the model to a higher-resource group for further expansion and training. FedKE uses a novel training mechanism that allows these groups to train in parallel. Inspired by parallel pipelining in distributed computing, this parallelism leverages the iterative aggregation of FL and allows higher-resource groups to start training as soon as the smaller model of the previous group is aggregated, rather than waiting for it to fully converge. Furthermore, each higher-resource group performs Adaptive Sparse Updating on the weights of the smaller model from the previous group to reduce computation overhead. The large model produced with FedKE theoretically approximates those trained under the ideal condition where all devices have the full capabilities to train this large model. The FedKE prototype is evaluated with two datasets and four model expansion configurations across three groups of real heterogeneous devices. The results show that it outperforms state-of-the-art solutions in terms of model accuracy, reduces the amount of model weights transferred and the total operation time compared to vanilla FL, and achieves good fault-tolerance and scalability.**

***Index Terms*—Federated Learning, Heterogeneous Computing**

I. Introduction

Federated Learning (FL) [20] naturally aligns with applications on personal devices, such as PCs, smartphones, and IoT devices, due to its privacy-preserving capabilities. This training scenario is referred to as cross-device FL [15]. For example, in a photo management system, smartphones, PCs, and IoT cameras from various users collaborate to improve a global model for automatic image classification and tagging. However, personal devices are equipped with varying levels of computing resources, all of which are limited. Because conventional FL aims to train a universal global model, the devices with the lowest computing resources become the bottleneck for the size of the model that can be trained in cross-device FL. Therefore, training large FL models over collections of heterogeneous devices becomes challenging, even though some devices in the system may be capable of handling it. To balance FL performance with resource heterogeneity, it is imperative to have an FL mechanism for training large models while utilizing all the devices with heterogeneous resources.

While large models have demonstrated superior performance across various tasks such as Natural Language Processing and Computer Vision, some devices may only be capable of training smaller models. There exists, however, a significant performance gap between small models and large models. For instance, on ImageNet, ResNet50 with 192 million parameters achieves 84.4% accuracy [2], whereas ResNet18 with 11.7 million parameters only achieves 72.05% accuracy [18]. In cross-device FL, a considerable number of devices may have the capability to train ResNet18 but lack the resources necessary to practically train the larger model - ResNet50. It would be advantageous if the training process in cross-device FL could also leverage all devices with different levels of computing resources to expand the model. During inference, each device utilizes the fully expanded model, as inference demands fewer resources compared to training. Recent works [1], [7], [9] in heterogeneous FL adopt the concept of partial training, leveraging devices with heterogeneous computing resources to train corresponding sub-models derived from a large global model, which are then aggregated into the final global model. However, these approaches only decompose the large global model into sub-models with fewer hidden channels, making them unsuitable for large models with deeper layers or multiple input heads. Additionally, the aggregation of heterogeneous models faces suboptimal convergence due to uneven training of weights across the final global model.

The goal of this study is to design an efficient cross-device FL framework to enable devices to train the largest possible model for their resources. This framework shall meet the following requirements: R1) utilize all available devices in the system regardless of their computing resources; R2) train a large model with diverse architectures, such as deeper layers and multiple input heads, which exceed the capabilities of some devices; R3) achieve high inference performance for the amount of resources available at each type of device; R4) be efficient in terms of communication, computation overhead, and overall training time. All these requirements shall be met under the FL assumption that the aggregation server does not have access to the raw data. To the best of our knowledge,

there is no existing work satisfying all the requirements.

This paper proposes **Federated Knowledge Expansion (FedKE)**, satisfying all the aforementioned requirements. First, we treat the devices with the same level of computing resources as an FL group. Thus, the FedKE system consists of multiple FL groups with different levels of computing resources. Second, each group of devices (from low resources to high resources) is assigned a model that is gradually expanded toward a larger model for the next group. Third, the devices within each group perform FL training, and the server expands the aggregated model for the next group to fine-tune. In this way, the models are iteratively trained and expanded toward the targeted largest model for inference. Furthermore, we introduce a novel **Parallel Training Mechanism for FedKE**, enabling simultaneous training across different groups and reducing the overall training time. This parallelism takes advantage of the iterative aggregation in FL. It enables higher-resource groups to begin training the expanded model as soon as the smaller model from the previous group is aggregated, without waiting for full convergence. Additionally, because the smaller model has been fine-tuned by the previous group, to improve training efficiency, we propose **Adaptive Sparse Updating (ASU)**. FedKE sparsely updates the smaller model within the newly expanded model adaptively over the local training epochs to reduce the computation overhead.

To showcase the effectiveness of FedKE, we prove the large model produced with FedKE theoretically approximates those trained under the ideal condition that all devices are fully capable of handling the large model. We experiment with four model expansion configurations and two datasets, varying in architecture and complexity. The results demonstrates FedKE improves model accuracy by up to 20.4% compared with state-of-the-art (SOTA) solutions, and achieves good fault tolerance and scalability. We also benchmark FedKE prototype on three different groups of heterogeneous devices, including *Nvidia Jetson nano*, *Raspberry Pi 5*, and *Raspberry Pi 3B+*. The on-device results show FedKE reduces the amount of model weights transferred up to 23.43%, reduces the total operation time up to 84.01% compared with vanilla FL.

II. Related Work

The heterogeneous nature of FL manifests in data distributions, network environments, and hardware among participating devices. Among these types of heterogeneity, the most studied is data heterogeneity, also known as non independent and identically distributed (non-IID) data. Several works [8], [12]–[14], [25], [26], [28] have been proposed to mitigate the non-IID issue. In FedProx [25], a regularization term is introduced to reduce gradient distortion across devices. Sarkar et al. [26] propose a cross-entropy loss function that downweights easy-to-classify examples and emphasizes training on harder-to-classify ones. Verma et al. [28] propose estimating the global objective function by averaging different local objective functions within a shared feature region, while maintaining distinct objective functions in other regions of the feature space based on local users' data. Favor [29] is an experience-driven control framework that intelligently selects client devices for each round of FL, aiming to counterbalance the bias introduced by non-IID data and accelerate convergence. However, these works do not address device heterogeneity, whereas FedKE specifically tackles this issue.

For device heterogeneity in FL, Wang et al. [30] propose selecting the optimal combination of sampled nodes and data offloading configurations to maximize FL training accuracy while adhering to realistic constraints related to network topology and device capabilities. Xu et al. [32] assess the importance of heterogeneous devices and proposed algorithms that select only the fast devices, which may lead to deviations in the convergence direction of the global model. In Hermes [17], each device identifies a small subnetwork through structured pruning, allowing only the updates from these subnetworks to be communicated between the server and the devices. Instead of averaging all parameters from all devices, the server averages only the overlapping parameters across each subnetwork. Similarly, Yu and Li [33] use different sub-models for heterogeneous devices in FL, and consider computation, computation, and power budget for the sub-models. FLAME [4] is a user-centered FL approach designed to counter statistical and system heterogeneity in mobile device environments, featuring user-aligned training, accuracy- and efficiency-aware device selection, and model personalization. EDDNN [10] horizontally partitions both the data and the DL model, allowing inference to be executed across multiple devices. In SplitFed [27], the DL model is split between clients and the server, and trained in an FL manner. However, these works utilize a single global model among devices and have not been demonstrated to be effective in scenarios where only high-resource devices can handle the global model.

Adapted from partial training, some approaches vary the width of hidden channels across devices with different computing power. As an early attempt, HectroFL [7] incorporates static batch normalization and a scaling module to enable the aggregation of heterogeneous small sub-models into a larger global model. Split-Mix [9] enables in-situ customization of model sizes and improves robustness for heterogeneous participants. FedRolex [1] adopts a rolling sub-model extraction strategy that ensures all components of the global server model are evenly trained, thereby alleviating client drift caused by mismatches between client models and the server model architecture. TAKFL [21] handles knowledge transfer from each device prototype's ensemble as a distinct task, independently distilling them to retain unique contributions and avoid dilution. FedX [16] introduces a novel adaptive model decomposition and quantization FL system for mobile/IoT devices. However, unlike FedKE, these methods are ineffective for Deep Learning (DL) models with diverse architectures, especially when training larger models with deeper layers or multiple input heads. In addition, these works are evaluated only in simulations using relatively simple datasets (e.g., no more complex than CIFAR-100), which constrains their applicability to real-world scenarios

Beyond FL, many generic distributed DL systems [5], [6], [19], [22], [31] utilize the parallel nature of DL to train large models. Most of these works focus on high-end GPU or CPU clusters rather than consumer electronics, such as mobile and IoT devices, and these works do not address privacy or device heterogeneity. For example, Ryabinin and Gusev [24] use decentralized mixture-of-experts and distributed hash tables toward crowdsourced training of Large Neural Networks. However, the experts have the same architecture, so the method does not work for heterogeneous devices. In addition, compared with a large global model, the inference has to be a distributed process with higher communication cost. Yuan et al [34] propose a scheduling algorithm that allocates different computational tasklets, but device heterogeneity is not considered. FedKE adapts some concepts from generic distributed DL systems, is specifically designed for consumer electronics, utilizes all the computational resources in the system, and targets training a large model collaboratively.

III. FedKE Framework

A. Problem Definition

FedKE is designed to train the largest feasible model that is suitable for inference across all devices. This model can be trained by at least the group of high-resource devices. Despite the heterogeneous resource constraints across devices, FedKE leverages the contributions of all devices in the system. Formally, let $G = \{G^1, G^2, \ldots, G^M\}$ be the set of M groups of devices in the order of their resources. Each group G^m has N^m devices with resources C^m. Due to limited resources, the device d_n^m in group G^m can only train networks w^m with size$(w^m) \leq C^m$. Each device d_n^m keeps their raw data D_n^m local to ensure privacy and avoid data transfer costs. We aim to train a neural network F with parameters Θ using all computing power in the system, but the training of F can only be handled by the group G^M with highest resources (i.e., $F \notin w^m$ where $m < M$ and $F \in w^M$). The problem can be formulated as minimizing the overall training loss, where $\mathcal{L}$ is the loss function:

$$\arg\min_{\Theta} \sum_{m=1}^{M} \sum_{n=1}^{N} \mathcal{L}(F(\Theta), D_n^m)$$

The weights Θ ensures that F generalizes well across all local datasets D_n^m.

B. FedKE Framework

Figure 1 illustrates the FedKE training framework. To initialize the process, the system administrator groups the devices in the system in the order of their computing resources, such as computation, memory, and storage capacity. Each group of devices is assigned to train a model that this group can handle with their resources. The models assigned to these groups are statically defined by the system administrator based on whether a test training process can be completed within a specified time frame. The model for a high-resource group is expanded from a smaller model for a low-resource group by adding computational layers or blocks. Section III-C describes more details. The intra-group training is carried out in conventional FL fashion. FedKE is a process in which the models are gradually expanded from groups with low resources to groups with high resources.

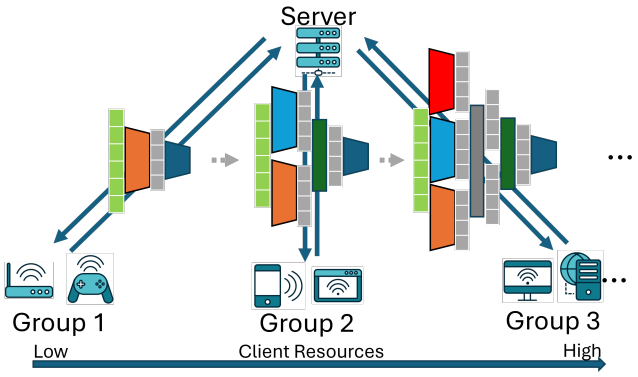

Fig. 1: FedKE training framework

The FedKE training starts with G^1. The server firstly sends the small base model to the devices in G^1. These devices perform training with their local data to update all the model weights, and send back the trained models to the server. After receiving the models from the devices, the server aggregates them, expands the aggregated model to the pre-defined architecture with partial untrained weights initialized randomly, and sends the expanded model to G^2. After the next group of devices receives the model, because some weights have already been trained by the previous group, we apply a novel Adaptive Sparse Updating (ASU) with the local data of the devices to reduce the computation overhead. ASU is described in section III-D. Then, the process repeats iteratively until the model is fully expanded and produced by G^M: devices send the models to the server; the server aggregates the models and expands the aggregated model for the next group; and the next group of devices performs ASU. The fully expanded model is sent to all groups of devices for inference.

Alg. 1 shows the pseudo-code of the FedKE framework. Devices with similar resources form FL groups (line 2). For each group in the order of resource amount (line 3), FedKE executes as a multi-round, iterative FL cycle until convergence (lines 4-12). For each devices in a group (line 6), it performs conventional local updating if it is in the first group with lowest resources (lines 7-8), and performs ASU otherwise (line 9-10). After updating the local model on-device, the server aggregates the models received from the devices (line 11) and expands the aggregated model for the next group (line 12).

C. Knowledge Expansion

Neural networks inherently feature a modular structure that supports extensibility and reuse. A representative example is the ResNet architecture, where models are composed of residual blocks—each consisting of convolutional layers and shortcut connections. Variants like ResNet-18 and ResNet-50 differ mainly in the number of such blocks, providing

Algorithm 1 FedKE Framework Pseudo-code

procedure SERVEREXECUTE:
  require grouping devices into M groups in the order of their resources, and defining the model architecture $w^1, w^2, ..., w^M$ for each group based on their computational capacity (w^{m+1} is expanded from w^m)
  for $group = 1$ to M **do**
    while w^m not converged **do**
      // Update On-device and Returned to Server
      for each devices n **do** // In Parallel
        if $group == 1$ **then**
          $\theta_n \leftarrow n.\text{CLIENTUPDATE}(w^m)$
        else
          $\theta_n \leftarrow n.\text{SPARSEUPDATE}(w^m, w^{m-1})$
      $w^m \leftarrow Aggregate(\{\theta_n^m\}_{n=1}^N)$
    $w^{m+1} \leftarrow Expand(w^m)$

scalable depth while maintaining a consistent design. Meanwhile, transfer learning has proven effective by fine-tuning pre-trained models on new datasets, significantly improving efficiency and performance. It benefits from the modularity of neural networks by reusing earlier layers that capture generalizable features. Knowledge Expansion (KE) builds on these principles by integrating a smaller, pre-trained model into a larger one, allowing higher-resource devices to expand their capabilities without retraining from scratch. This approach not only transfers knowledge but also extends the model structurally, enabling efficient training of more powerful models.

In FedKE, a small base model is defined for low-resource devices. For devices with greater capacity, extended models are constructed by adding layers or blocks. When hidden state dimensions do not align, projection layers such as linear or 1×1 convolutional layers are inserted to match dimensions between components. FedKE leverages device heterogeneity by adapting model complexity to local capabilities. It reuses pre-trained weights with structured expansion, offering a practical path to build larger and more accurate models collaboratively across a federated network without excluding weaker devices.

KE Theoretical Analysis. In the following, we first analyze KE as a two-step model training and expansion process and prove that the parameter difference between a large model trained with KE and one trained conventionally under the ideal conditions (i.e., all devices have full capabilities to train the large model) is bounded and approaches zero. Secondly, we use induction to prove that KE, as an n-step model training and expansion process, can also train the large model that approximates the one trained conventionally.

Theorem 1 (two-step KE bound over D_1, D_2)**.** *Let $L(w, D)$ be a twice-differentiable loss function, where $D = D_1 \cup D_2$ and $D_1 \cap D_2 = \emptyset$. Given that*

- *w_l^{ideal} minimizes $L(w, D)$ over the full dataset D.*
- *w_s is trained on D_1 and $w_l^{two\text{-}step}$ is obtained by expanding the model from w_s and fine-tuning on D_2.*

Then, the parameter difference $\|w_l^{two\text{-}step} - w_l^{ideal}\|$ is bounded as $\|w_l^{two\text{-}step} - w_l^{ideal}\| \le \|H^{-1}\| \cdot \|\nabla L(w_l^{two\text{-}step}, D_1)\|$, where $H = \nabla^2 L(w_l^{ideal}, D)$ is the Hessian of the loss function at w_l^{ideal}.

Proof. Using the Taylor expansion for the gradient at $w_l^{\text{two-step}}$ around w_l^{ideal}, we have $\nabla L(w_l^{\text{two-step}}, D) \approx \nabla L(w_l^{\text{ideal}}, D) + H(w_l^{\text{two-step}} - w_l^{\text{ideal}})$, where $H = \nabla^2 L(w_l^{\text{ideal}}, D)$ is the Hessian of the loss function at w_l^{ideal}.

From the ideal training, w_l^{ideal} minimizes $L(w, D)$, so $\nabla L(w_l^{\text{ideal}}, D) = 0$. Since $\nabla L(w_l^{\text{ideal}}, D) = 0$, the expansion simplifies to $\nabla L(w_l^{\text{two-step}}, D) \approx H(w_l^{\text{two-step}} - w_l^{\text{ideal}})$.

Rewriting the above equation, we get $w_l^{\text{two-step}} - w_l^{\text{ideal}} \approx H^{-1}\nabla L(w_l^{\text{two-step}}, D)$. Taking the norm, we have $\|w_l^{\text{two-step}} - w_l^{\text{ideal}}\| \le \|H^{-1}\| \cdot \|\nabla L(w_l^{\text{two-step}}, D)\|$. The total gradient at $w_l^{\text{two-step}}$ is $\nabla L(w_l^{\text{two-step}}, D) = \nabla L(w_l^{\text{two-step}}, D_1) + \nabla L(w_l^{\text{two-step}}, D_2)$. From the two-step training, $w_l^{\text{two-step}}$ minimizes $L(w, D_2)$, so $\nabla L(w_l^{\text{two-step}}, D_2) = 0$. Thus, $\nabla L(w_l^{\text{two-step}}, D) = \nabla L(w_l^{\text{two-step}}, D_1)$. Combining the above results, the parameter difference is bounded as $\|w_l^{\text{two-step}} - w_l^{\text{ideal}}\| \le \|H^{-1}\| \cdot \|\nabla L(w_l^{\text{two-step}}, D_1)\|$. □

If D_1 is a representative subset of D, $\nabla L(w_l^{\text{two-step}}, D_1)$ is close to zero. Because $H = \nabla^2 L(w_l^{\text{ideal}}, D)$ is the Hessian of the loss function at w_l^{ideal}, it does not depends on $w_l^{\text{two-step}}$. Given the equality above, $\|w_l^{\text{two-step}} - w_l^{\text{ideal}}\| \le \|H^{-1}\| \cdot \|\nabla L(w_l^{\text{two-step}}, D_1)\|$ is also close to zero. Next, we generalize the two-step expansion to a general case of n-step expansion, and prove it through induction.

Theorem 2 (general KE bound over $D_1, \ldots, D_n$)**.** *Let $L(w, D)$ be twice differentiable, with $D = \bigsqcup_{k=1}^n D_k$ (disjoint union). Let w^{ideal} minimize $L(w, D)$ over the full dataset, and let $H = \nabla^2 L(w^{ideal}, D)$.*

Consider the KE training schedule that finishes by fine-tuning on the last sub-dataset D_n. Then, $\|w^{(n)} - w^{ideal}\| \le \|H^{-1}\| \cdot \left\| \sum_1^n \nabla L(w^{(n)}, D_1) \right\|$.

Proof. Define $w^{(k)}$ as the parameter obtained after finishing stage k (i.e., the last stage is optimized with sub-dataset D_k).

For $k = 2$ (the two-step case), according to **Theorem 1**, the bound is $\|w^{(2)} - w^{\text{ideal}}\| \le \|H^{-1}\| \cdot |\nabla L(w^{(2)}, D_1)\|$. Because $\nabla L(w^{(1)}, D_1) = 0$, $\|H^{-1}\| \cdot \|\nabla L(w^{(2)}, D_1)\| = \|H^{-1}\| \cdot \|0 + \nabla L(w^{(2)}, D_1)\| = \|H^{-1}\| \cdot \| \sum_1^2 \nabla L(w^{(n)}, D_1)\|$, which confirms **Theorem 2** holds.

We assume for some $k \ge 2$ that $\|w^{(k)} - w^{\text{ideal}}\| \le \|H^{-1}\| \cdot \| \sum_1^k \nabla L(w^{(n)}, D_1)\|$.

Advancing one stage and finishing on D_{k+1}, we have

$$
\begin{aligned}
&\|w^{(k+1)} - w^{\text{ideal}}\| \\
&\le \|w^{(k+1)} - w^{\text{ideal}}\| \\
&= \|w^{(k+1)} - w^{(k)} + w^{(k)} - w^{\text{ideal}}\| \\
&\le \|w^{(k+1)} - w^{(k)}\| + \|w^{(k)} - w^{\text{ideal}}\| \\
&\le \|w^{(k+1)} - w^{(k)}\| + \|H^{-1}\| \cdot \left\| \sum_1^k \nabla L(w^{(n)}, D_1) \right\|
\end{aligned}
$$

$$= \nabla L(w^{(k+1)}, D_{k+1}) + \|H^{-1}\| \cdot \Big\| \sum_{1}^{k} \nabla L(w^{(n)}, D_1) \Big\|$$

$$\leq \nabla L(w^{(k+1)}, D_1) + \|H^{-1}\| \cdot \Big\| \sum_{1}^{k} \nabla L(w^{(n)}, D_1) \Big\|$$

$$\leq \|H^{-1}\| \cdot \nabla L(w^{(k+1)}, D_1) + \|H^{-1}\| \cdot \Big\| \sum_{1}^{k} \nabla L(w^{(n)}, D_1) \Big\|$$

$$= \|H^{-1}\| \cdot \Big\| \sum_{1}^{k+1} \nabla L(w^{(n)}, D_1) \Big\|$$

This is exactly the statement with $k+1$. Hence, by induction, **Theorem 2** holds for all n. □

Similar as in **Theorem 1**, $\|w^{(n)} - w^{\text{ideal}}\| \leq \|H^{-1}\| \cdot \|\sum_1^n \nabla L(w^{(n)}, D_1)\|$ is close to zero. This phenomenon is further supported by our experimental results. The accuracy difference between FedKE and vanilla FL (under the assumption that all devices train the expanded model) ranges from 1.8% higher to 1.0% lower.

D. Adaptive Sparse Updating (ASU)

When updating a model that has been expanded from a previous device group, it is inefficient to fine-tune all of the model's weights, especially given that the smaller base model has already captured useful representations from previous group's local data. To improve efficiency without sacrificing performance, we propose Adaptive Sparse Update (ASU), which updates only a subset of weights—specifically, those with relatively low magnitudes. The rationale is that low-magnitude weights have not yet learned enough from the data and thus require further optimization [11], [35]. ASU begins with a full model update and then progressively reduces the proportion of weights updated in each local training epoch. This reduction is adaptive and controlled by a model sparsity threshold that balances the computational capabilities of the device with the need to maintain high model accuracy. High-magnitude weights are increasingly excluded from updates, under the assumption that they have already learned significant patterns, and thus do not require further fine-tuning. This approach also reduces redundancy in computation as training progresses and model convergence slows down. By prioritizing updates on the less-converged, lower-magnitude weights, ASU makes the training process more resource-efficient while preserving model quality. These design choices contribute directly to the overall effectiveness and scalability of the FedKE framework.

Alg. 2 shows the pseudo-code of ASU. It linearly decreases the percentile used as sparsity threshold of the model weights (lines 3-7), and does not update the weights of the previous smaller model which are high in magnitude (lines 8-17). We determine the sparsity threshold by evaluating the trade-off between training efficiency and model accuracy. Empirically, we observe the threshold to be as low as 50% to maintain the model accuracy, as discussed in Section IV-F.

Algorithm 2 Adaptive Sparse Updating Pseudo-code

1: **procedure** SPARSEUPDATE:(w^m, w^{m-1})
2: // Executed at Devices
3: require step size hyperparameter η, local dataset of current round $\mathcal{D}$, model sparsity $p\%$, number of epochs E
4: $unit \leftarrow (100 - p)/E$
5: $x_n \leftarrow \mathcal{D}$ divided into minibatches
6: **for** $epoch = 1$ to E **do**
7: $threshold \leftarrow (100 - unit * epoch)$ percentile in w^{m-1}
8: $mask \leftarrow w^m$
9: **for** each element $e \in mask$ **do**
10: **if** $e > threshold$ and $e \in w^{m-1}$ **then**
11: $e \leftarrow 0$
12: **else**
13: $e \leftarrow 1$
14: **for** each batch $b \in x_n$ **do**
15: $\theta_n \leftarrow w^m - \eta \nabla L(w^m, b) \odot mask$
16: // Results Returned to Server
17: **return** θ_n

Alg. 3 shows the pseudo-code of the conventional client update which updates the model with the gradients calculated through batches of data (lines 3-8). As described in Alg. 1, the lowest-resource group (the first group) performs conventional client update, because the devices in it train the small base model from scratch. The other groups perform ASU, since their model is build upon a pre-trained smaller model from previous group.

Algorithm 3 Conventional Client Update Pseudo-code

1: **procedure** CLIENTUPDATE:(w^m)
2: // Executed at Clients
3: require step size hyperparameter η, local dataset of current round $\mathcal{D}$
4: $x_n \leftarrow \mathcal{D}$ divided into minibatches
5: **for** each batch $b \in x_n$ **do**
6: $\theta_n \leftarrow w^m - \eta \nabla L(w^m, b)$
7: // Results Returned to Server
8: **return** θ_n

E. Inter-Group Parallel Training

We propose a novel parallel training mechanism in FedKE that enables different groups of devices to train concurrently. This design is inspired by data parallelism and pipeline parallelism in distributed systems, and it effectively exploits the key characteristics of FedKE (i.e., the progressive model expansion across device groups and the iterative aggregation of models in rounds). Importantly, this approach relaxes the need for full model convergence before moving to the next group, thus accelerating the overall training process. In each communication round, a group of devices performs local

Algorithm 4 FedKE Inter-group Parallel Training Pseudo-code

1: **procedure** SERVEREXECUTE:
2: require grouping devices into M groups in the order of their resources, defining the model architecture $w^1, w^2, ..., w^M$ for each group based on their computational capacity (w^{m+1} is expanded from w^m), and the total number of training rounds R
3: **for** $r = 1$ to R **do**
4: **for** $group = 1$ to M **do** // In Parallel
5: **if** $group == 1$ **then**
6: **for** each devices n **do** // In Parallel
7: $\theta_n \leftarrow n.\text{CLIENTUPDATE}(w^1_{r-1})$
8: **else**
9: **if** w^m_{r-1} exists **then**
10: $w^m_r \leftarrow Aggregate(w^m_{r-1}, w^m_r)$
11: **for** each devices n **do** // In Parallel
12: $\theta_n \leftarrow n.\text{SPARSEUPDATE}(w^m, w^{m-1})$
13: $w^m_r \leftarrow Aggregate(\{\theta^m_n\}^N_{n=1})$
14: $w^{m+1}_r \leftarrow Expand(w^m_r)$

training and sends their updated models to the server. Upon receiving and aggregating these models, the server expands the model by adding new layers or blocks and immediately forwards the expanded model to the next group. Crucially, this next group can begin fine-tuning the expanded model while the previous group continues with its next round of training on a fresh version of the base model. This results in inter-group parallelism, where multiple groups are engaged in different phases of the training pipeline simultaneously. Once both groups complete their respective operations (fine-tuning and further training), FedKE aggregates the resulting models using standard techniques from generic FL, such as FedAvg. The aggregated model is then expanded again for the subsequent group, continuing the pipeline. This parallelized mechanism substantially increases system throughput and reduces total training time compared to traditional sequential inter-group updates. It maximizes resource utilization across heterogeneous devices and contributes to the scalability and efficiency of FedKE in practical deployments.

Alg. 4 shows the pseudo-code of FedKE Inter-group Parallel Training. For each round of training (line 3), in the order of the resources of the groups running in parallel (line 4), the first group performs conventional client update (lines 5-7).The server aggregates the devices model (line 14), and expands it before convergence (line 15). Starting from group 2, each group starts to train as soon as an expanded model from the previous group was generated (line 8), instead of waiting for the model full convergence of previous group. If there has been a model aggregated in previous round (line 9), the server shall aggregate the model of the previous round with the newly expanded model of the previous group (line 10). Then, the devices in the group perform ASU (line 11-12). In this way, the server aggregates the devices models for each group in every round (line 13) and expands the aggregated model for the next group (line 14).

IV. EVALUATION

The evaluation has six goals: (i) Compare FedKE with vanilla FL; (ii) Compare FedKE with state-of-the-art (SOTA) solutions; (iii) Investigate the effectiveness of ASU; (iv) Quantify FedKE end-to-end operation time; (v) Quantify the communication and computation savings; (vi) Investigate the fault tolerance and scalability of FedKE.

A. Datasets

We utilize two popular image datasets: Cifar-100 and Tiny-ImageNet. Cifar-100 consists of 60,000 32x32 color images spread across 100 classes, with each class containing 600 images, offering a diverse set of categories ranging from animals to vehicles. TinyImageNet, a scaled-down version of the ImageNet dataset, includes a total of 120,000 images, with 100,000 training images, 10,000 validation images, and 10,000 test images across 200 classes, with images resized to 64x64 pixels. Both datasets present significant challenges for image classification models due to their diverse and complex image content across numerous categories.

B. Models and Expansion Configurations

We evaluate FedKE with four different DL model setups varying in both complexity and expansion configurations. These configurations involve 2 (*configuration 1-3*) or 3 (*configuration 4*) groups of devices, and incorporate diverse model expansion scenarios, such as deeper layers or multiple input heads. Instead of maximizing the model accuracy, the expansion configurations are designed so that the group of lower-resource devices is capable to train a small base model, but cannot efficiently handle training the fully expanded model, due to issues such as running out of memory and thermal throttling.

Configuration 1. We expand a CNN model for the Cifar100 dataset. The base model includes two convolutional layers with 32 and 64 filters (3x3), followed by max pooling, flattening, and a fully connected classifier with dropout for regularization. The classifier consists of a linear layer reducing the features to 512 dimensions, a ReLU activation, and a final linear layer for class output. The expanded model adds two convolutional layers with 128 and 256 filters, followed by a max pooling layer and a matching dropout and fully connected layer to match the base model's output dimensions. All CNN layers are followed by ReLU activation.

Configuration 2. We expand ResNet6 to ResNet18 for the more challenging TinyImageNet dataset. The base ResNet6 model includes four CNN layers with 64 filters, followed by adaptive average pooling, and a fully connected layer to produce class predictions. The expanded ResNet18 adds 4 more CNN layers with 128, 256, and 512 filters. The output layer is adjusted to match the feature size.

Configuration 3. We fuse two feature extractors and concatenate their outputs for a multi-branch network. The base model

is a Vision Transformer (ViT) [23], which processes input images into patches, adds positional encoding, and passes them through a transformer encoder with multi-head self-attention and dropout for regularization. After global average pooling, the output is passed through a linear classifier. The second model is a simple CNN, which consists of two convolutional layers with 16 filters, ReLU activation, and max pooling. The outputs from both the ViT and CNN are concatenated, passed through a fully connected layer, and classified using the base model's classifier.

Configuration 4. To demonstrate FedKE's performance with multi-step expansion, we expand a CNN model in three stages with three device groups. The base model starts with a 2D convolutional layer (32 filters, 3x3), followed by a max pooling layer. The output is then passed through a fully connected layer with 1024 neurons and ReLU activation. The medium model adds two CNN blocks with 64 and 128 filters, followed by max pooling. The large model adds two more blocks with 256 and 512 filters, using batch normalization and ReLU activation throughout. The classifier for both models consists of fully connected layers. Both *Configurations 3* and *4* are evaluated using Cifar100.

C. Prototypes and Devices

We evaluate FedKE through simulations and an on-device prototype, implemented using Pytorch and Flower [3]. For the model performance in FedKE, we run the simulation on a GPU station with 4 NVIDIA GeForce Titan Xp GPUs (12G each), Intel Xeon E5-2637 v4 CPU (3.50GHz), Linux (Ubuntu 16.04), and CUDA 11.3. The FedKE on-device prototype, implemented in Python on Linux, is tested on three diverse IoT devices with heterogeneous resources to benchmark communication and computation overhead.

Nvidia Jetson Nano (Jetson). This device is an entry-level edge computing platform, featuring a quad-core ARM Cortex-A57 CPU with a clock speed of 1.43 GHz and 4 GB RAM.

Raspberry Pi 5 (Pi5). Launched in 2024, *Pi5* features a quad-core ARM Cortex-A72 CPU with clock speeds of 2.4GHz, and 4GB RAM.

Raspberry Pi 3B+ (Pi3B+). Launched in 2018, *Pi3B+* features a quad-core ARM Cortex-A53 CPU with clock speeds of 1.4GHz, and 1GB RAM.

D. Experimental Setup

The datasets are randomly split according to the number of device groups. We set the number of devices per group as 10. The group data is further distributed to each device in a non-IID manner under a Dirichlet distribution. Each device applies the Adam optimizer with learning rate of 0.001, batch size of 64, and performs 10 epochs of local update in each training round. Unless otherwise specified, each device performs a full model update, and the system performs parallel FedKE from one group to another. ASU starts with a full model update and linearly decreases the update ratio of high-magnitude weights over 10 local epochs. We performed experiments across different sparsity levels, ensuring that each device updated a minimum of 50%, 30%, and 10% of the model weights, respectively. The devices and the server are connected via a WiFi network with a speed of 300 Mbps. The system runs 100 rounds of training for each experiment with FedAvg aggregation algorithm. We apply the same experimental setup for both simulation and on-device benchmarks. For model performance, we report the best results, while for computation time, we provide the average from 10 experiments.

E. Ablation and Comparison Study

To evaluate FedKE, we first run FL under the unrealistic assumption that all devices in the system are capable of training the fully expanded model. Although this assumption may not hold in the real world, it provides an upper bound on the FL model's performance given the data setup. Next, we run FL under more realistic scenarios where the groups perform FL on a model suitable for their resources, which also functions as the ablation study of FedKE. We compare the learning curves of the above scenarios with FedKE learning curves to demonstrate the effectiveness of FedKE.

Meanwhile, we compare FedKE with FedRolex [1] and SplitFed [27] as they are the influential solutions capable of functioning effectively when the model is expanded with deeper layers. FedRolex adapts partial training similar to FedKE, and uses a rolling sub-model extraction scheme that allows different parts of the global server model to be evenly trained. In SplitFed, the DL model is split between devices and the server. Similar to FedKE, this split model architecture makes SplitFed an option for resource-constrained environments. We considered several other SOTA baselines, such as HeteroFL [7], FedX [16], and Split-Mix [9], but these methods are limited to model expansion through increasing the number of channels or neurons per layer, and are incompatible with our configuration, which involves vertical expansion by adding layers and horizontal expansion by increasing the number of computational blocks.

F. Results

Ablation Study. Figure 2 show FedKE superior learning process over the four different configurations. FedKE can sucessfully expand a small base model to the fully expanded model, and increase the model accuracy by 11.1%, 8.9%, 5.6%, and 14.3%, respectively. Without FedKE, when only the group with high resources is used to train the expanded model, FedKE outperforms it by 10.7%, 4.8%, 3.1%, and 1.7%, respectively. To further showcase the superiority of FedKE, we compare it with an unrealistic assumption that all devices train the expanded model. Superisingly, FedKE achieves higher model accuracy by 1.8% and 5.0% in *Configurations 1* and *3*, respectively. For *Configurations 2* and *4*, although full-participation FL outperforms FedKE, but its assumption that low-resource groups can handle the fully expanded model is unrealistic in real-world scenarios.

Comparison Study. Figure 3 shows FedKE outperforms FedRolex and SplitFed by 3.1% and 20.4% for *Configuration 1*, and by 4.0% and 12.0% for *Configuration 2*, respectively.

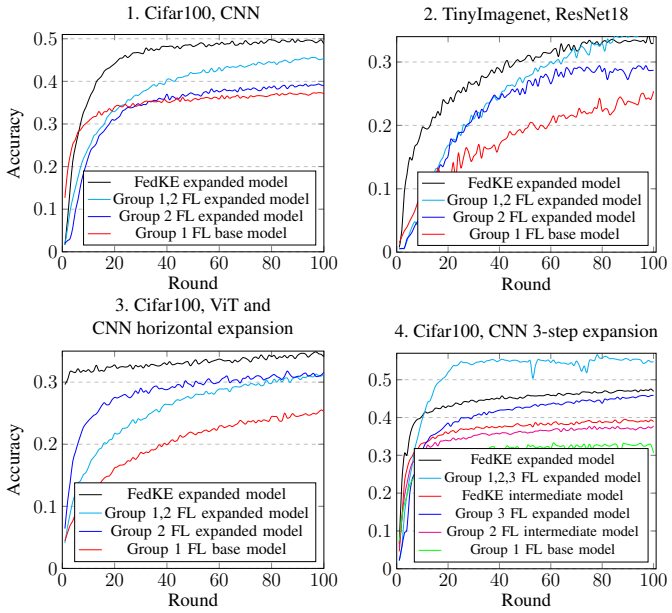

Fig. 2: Ablation study of FedKE

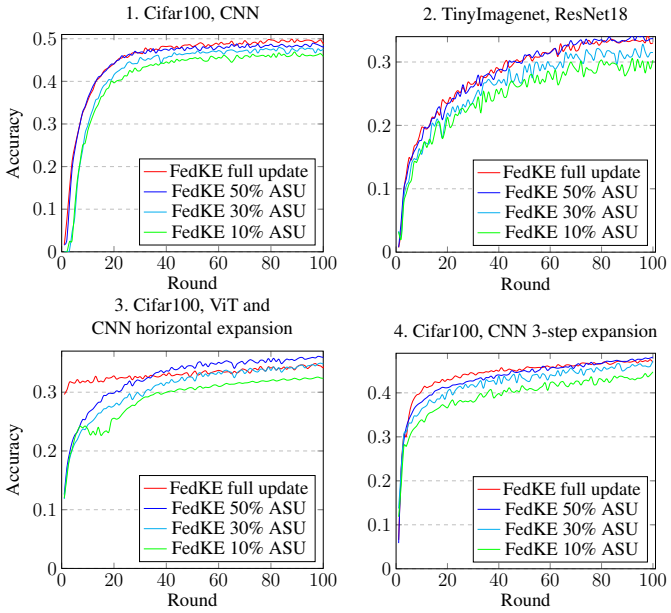

Fig. 4: FedKE full update vs. ASU

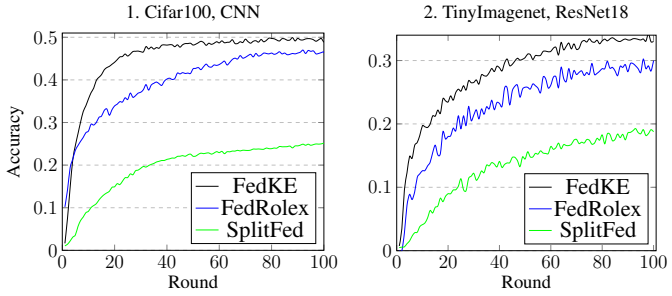

Fig. 3: Comparison study of FedKE

SplitFed splits the training process between the server and clients, and does not work well in expanding the models. FedRolex performs reasonably well, though it still falls short of FedKE. However, FedRolex and SplitFed cannot handle diverse expansion configurations as FedKE. Because SplitFed and FedRolex do not work under *Configuration 3* and *4*, we can not include their learning curves in the comparison study.

Effectiveness of ASU. Figure 4 compares the learning curves between FedKE full update and ASU over different sparsities. Starting from a full update, we linearly reduce the update ratio to the 50th, 30th, and 10th percentiles of the weights in magnitude over 10 local epochs, respectively, to balance the computational load across devices while preserving model accuracy. The results show that they achieve comparable model performance. When the update ratio is gradually reduced to 50%, the model's accuracy differs by less than 1.2% compared to the full update. As we decrease the update ratio to 30% or 10%, the model accuracy decreases, as expected, reflecting the trade-off between model performance and computational load. Nevertheless, the reduction in performance is limited to less than 4.5%. This suggests that ASU maintains performance comparable to the full update, indicating that a 50% update ratio is a safe choice without sacrificing accuracy. Furthermore, computation costs can be reduced even more by gradually decreasing the update ratio to as low as 10% of the model weights over the training epochs.

End-to-end Operation Time. Tables I shows the breakdown of one round end-to-end operation time on-device. We observe the computation capability of the devices has a significant impact on the operation time. Among *Jetson*, *Pi5*, and *Pi3B+*, *Jetson* is the most powerful for training, as it can utilize its GPU for DL tasks. The *Pi5* is next in line, while the *Pi3B+* is the weakest. For training, *Pi5* takes 2.1× longer than *Jetson* to train the expanded model in horizontal expansion, while *Pi3B+* takes 20.6× longer than *Jetson* to train the expanded model in 3-step expansion. The time for sending and receiving model is calculated by transferring the models through a 300 Mbps WiFi network. We also measure the latency of the aggregation at the server. Transferring the models and aggregations consume a negligible portion of the overall operation time, accounting for less than 1.3%. Overall, training the expanded model on the *Pi3B+* is far less feasible compared to *Jetson*, taking 60 minutes vs. 3 minutes. This demonstrates why FedKE is important in practice.

On-device Inference Time. We also benchmark the inference time for the three devices. For example, it takes *Jetson*, *Pi5*, and *Pi3B+* 0.14 ±0.01(one standard deviation), 0.16 ±0.02, and 0.53 ±0.06 seconds respectively to perform one inference with the fully expanded model in 3-step expansion. The difference is minimal, allowing all groups to use the fully expanded model during inference.

TABLE I: Operation time (s) details on Cifar100 in one round

		Receiving model	Training	Sending model	Aggregation	Total
ViT and CNN horizontal expansion	*Pi5*_base	0.12	95.61	0.12	0.03	95.88
	*Pi5*_expanded	0.22	125.65	0.22	0.05	126.14
	*Jetson*_expanded	0.22	60.65	0.22	0.05	61.14
CNN 3-step expansion	*Pi3B+*_expnded	1.02	3600.24	1.02	0.23	3602.52
	*Pi3B+*_base	0.86	570.37	0.86	0.20	572.29
	*Pi5*_intermediate	0.87	213.58	0.87	0.23	215.56
	*Jetson*_expanded	1.02	174.82	1.02	0.23	177.10

TABLE II: Total operation time (h) and model weights transfer amount (GB) compared with vanilla FL over 100 rounds

		Total time (h)	Model weights transferred (GB)
ViT and CNN horizontal expansion	Vanilla FL	3.50	32.27
	FedKE	2.65	24.71
	Saving (%)	24.27	23.43
CNN 3-step expansion	Vanilla FL	100.07	225.00
	FedKE	16.01	202.34
	Saving (%)	84.01	10.07

Computation and communication savings. Tables II quantifies computation and communication savings of FedKE in terms of the total operation time and the amount of model weights transferred. Compared to vanilla FL, FedKE saves up to 84% in total operation time, and 23.43% in the amount of model weights transferred. In vanilla FL, the devices with high resources wait for the devices with low resources to complete the training, before the server performs aggregation. Because all the devices train the same model in vanilla FL, the time is significantly higher compared to FedKE, where low-resource devices only train a smaller model. Even though the high-resource group in FedKE has to wait for the expanded model from the low-resource group as shown in Alg. 4, its total operation time is still significantly lower than that of vanilla FL. In terms of the amount of model weights transferred, because the devices with low resources in FedKE only transfer a smaller model, the overall model weights transferred can be significantly reduced. This is beneficial for mobile and IoT devices, as their data usage is usually metered.

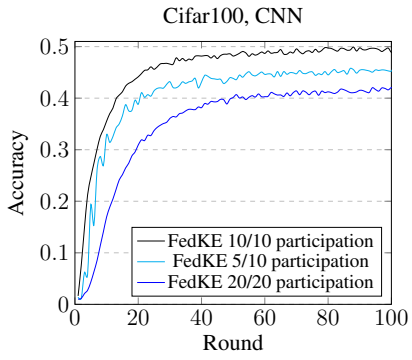

Fig. 5: FedKE fault tolerance and scalability in terms of the number of participating devices

FedKE Fault Tolerance and Scalability. In the original setup, all 10 users in each group participate in every training round. To simulate device fault, only 50% of devices (5/10 participation) are randomly selected to participate in each round. To demonstrate scalability, the number of devices is doubled (20/20 participation) by splitting the data from each device into two parts of equal size.

Figure 5 compares the learning curves of FedKE in *Configuration 1* under the three scenarios above. We observe that the model performance consistently improves in both additional scenarios, indicating that FedKE is resilient to faults and can scale effectively. The final model accuracy shows a minor decrease compared to the original setup due to the halved number of participating devices or the reduced data per device.

V. Conclusion

We present FedKE, a method that incrementally enhances a small model by adding computational layers or blocks, ultimately evolving into a larger model. FedKE is designed to balance between model accuracy/size and device resources in systems of heterogeneous devices. Devices with similar resources are organized into FL groups, where knowledge is expanded. Each group trains a model that matches its resources, expands it, and subsequently transfers this model to a higher-resource group. FedKE employs an innovative parallelism that enables these groups to train concurrently. Additionally, each higher-resource group applies ASU to the weights of the smaller model from the prior group, effectively reducing computational overhead. The model generated by FedKE theoretically approaches the performance of models trained under ideal conditions, where all devices possess the full capability to handle training the model. Our evaluation demonstrates that FedKE surpasses SOTA solutions in model accuracy while also decreasing the volume of model weights transferred and the total operational time.

Acknowledgements

This research was supported by the National Science Foundation (NSF) under Grant No. OAC 2451611, CNS 2237328, and DGE 2043104. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

REFERENCES

[1] Samiul Alam, Luyang Liu, Ming Yan, and Mi Zhang. Fedrolex: Model-heterogeneous federated learning with rolling sub-model extraction. *Advances in neural information processing systems*, 35:29677–29690, 2022.

[2] Irwan Bello, William Fedus, Xianzhi Du, Ekin Dogus Cubuk, Aravind Srinivas, Tsung-Yi Lin, Jonathon Shlens, and Barret Zoph. Revisiting resnets: Improved training and scaling strategies. *Advances in Neural Information Processing Systems*, 34:22614–22627, 2021.

[3] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, Pedro Porto Buarque de Gusmão, and Nicholas D. Lane. Flower: A friendly federated learning research framework, 2021.

[4] Hyunsung Cho, Akhil Mathur, and Fahim Kawsar. Flame: Federated learning across multi-device environments. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 6(3):1–29, 2022.

[5] Harshit Daga, Patrick K Nicholson, Ada Gavrilovska, and Diego Lugones. Cartel: A system for collaborative transfer learning at the edge. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 25–37, 2019.

[6] Jason Jinquan Dai, Yiheng Wang, Xin Qiu, Ding Ding, Yao Zhang, Yanzhang Wang, Xianyan Jia, Cherry Li Zhang, Yan Wan, Zhichao Li, et al. Bigdl: A distributed deep learning framework for big data. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 50–60, 2019.

[7] Enmao Diao, Jie Ding, and Vahid Tarokh. Heterofl: Computation and communication efficient federated learning for heterogeneous clients. In *International Conference on Learning Representations*, 2021.

[8] M. Duan, D. Liu, X. Chen, R. Liu, Y. Tan, and L. Liang. Self-balancing federated learning with global imbalanced data in mobile systems. *IEEE Transactions on Parallel & Distributed Systems*, 32(01):59–71, jan 2021.

[9] Junyuan Hong, Haotao Wang, Zhangyang Wang, and Jiayu Zhou. Efficient split-mix federated learning for on-demand and in-situ customization. In *International Conference on Learning Representations*, 2022.

[10] Yakun Huang, Xiuquan Qiao, Wenhai Lai, Schahram Dustdar, Jianwei Zhang, and Jiulin Li. Enabling dnn acceleration with data and model parallelization over ubiquitous end devices. *IEEE Internet of Things Journal*, 9(16):15053–15065, 2021.

[11] Xiaopeng Jiang and Cristian Borcea. Complement sparsification: Low-overhead model pruning for federated learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 8087–8095, 2023.

[12] Xiaopeng Jiang and Cristian Borcea. Concept matching: clustering-based federated continual learning. *arXiv preprint arXiv:2311.06921*, 2023.

[13] Xiaopeng Jiang, Han Hu, Thinh On, Phung Lai, Vijaya Datta Mayyuri, An Chen, Devu M Shila, Adriaan Larmuseau, Ruoming Jin, Cristian Borcea, et al. Flsys: Toward an open ecosystem for federated learning mobile apps. *IEEE Transactions on Mobile Computing*, 23(1):501–519, 2022.

[14] Xiaopeng Jiang, Thinh On, NhatHai Phan, Hessamaldin Mohammadi, Vijaya Datta Mayyuri, An Chen, Ruoming Jin, and Cristian Borcea. Zone-based federated learning for mobile sensing data. In *2023 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 141–148. IEEE, 2023.

[15] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and trends® in machine learning*, 14(1–2):1–210, 2021.

[16] Phung Lai, Xiaopeng Jiang, Hai Phan, Cristian Borcea, Khang Tran, An Chen, Vijaya Datta Mayyuri, and Ruoming Jin. Fedx: Adaptive model decomposition and quantization for iot federated learning. In *2025 21st International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*, pages 212–220, 2025.

[17] Ang Li, Jingwei Sun, Pengcheng Li, Yu Pu, Hai Li, and Yiran Chen. Hermes: an efficient federated learning framework for heterogeneous mobile clients. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, pages 420–437, 2021.

[18] Zicheng Liu, Siyuan Li, Di Wu, Zihan Liu, Zhiyuan Chen, Lirong Wu, and Stan Z Li. Automix: Unveiling the power of mixup for stronger classifiers. In *European Conference on Computer Vision*, pages 441–458. Springer, 2022.

[19] Chengfei Lv, Chaoyue Niu, Renjie Gu, Xiaotang Jiang, Zhaode Wang, Bin Liu, Ziqi Wu, Qiulin Yao, Congyu Huang, Panos Huang, et al. Walle: An end-to-end, general-purpose, and large-scale production system for device-cloud collaborative machine learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 249–265, 2022.

[20] 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*, pages 1273–1282, 2017.

[21] Mahdi Morafah, Vyacheslav Kungurtsev, Hojin Chang, Chen Chen, and Bill Lin. Towards diverse device heterogeneous federated learning via task arithmetic knowledge integration. *Advances in Neural Information Processing Systems*, 37:127834–127877, 2024.

[22] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. Ray: A distributed framework for emerging {AI} applications. In *13th USENIX symposium on operating systems design and implementation (OSDI 18)*, pages 561–577, 2018.

[23] Prajit Ramachandran, Niki Parmar, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jon Shlens. Stand-alone self-attention in vision models. *Advances in neural information processing systems*, 32, 2019.

[24] Max Ryabinin and Anton Gusev. Towards crowdsourced training of large neural networks using decentralized mixture-of-experts. *Advances in Neural Information Processing Systems*, 33:3659–3672, 2020.

[25] Anit Kumar Sahu, Tian Li, Maziar Sanjabi, Manzil Zaheer, Ameet Talwalkar, and Virginia Smith. On the convergence of federated optimization in heterogeneous networks. *CoRR*, abs/1812.06127, 2018.

[26] Dipankar Sarkar, Ankur Narang, and Sumit Rai. Fed-focal loss for imbalanced data classification in federated learning, 2020.

[27] Chandra Thapa, Pathum Chamikara Mahawaga Arachchige, Seyit Camtepe, and Lichao Sun. Splitfed: When federated learning meets split learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8485–8493, 2022.

[28] Dinesh C. Verma, Graham White, Simon Julier, Stepehen Pasteris, Supriyo Chakraborty, and Greg Cirincione. Approaches to address the data skew problem in federated learning. In Tien Pham, editor, *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, volume 11006, pages 542 – 557. International Society for Optics and Photonics, SPIE, 2019.

[29] Hao Wang, Zakhary Kaplan, Di Niu, and Baochun Li. Optimizing federated learning on non-iid data with reinforcement learning. In *IEEE INFOCOM 2020-IEEE conference on computer communications*, pages 1698–1707. IEEE, 2020.

[30] Su Wang, Mengyuan Lee, Seyyedali Hosseinalipour, Roberto Morabito, Mung Chiang, and Christopher G Brinton. Device sampling for heterogeneous federated learning: Theory, algorithms, and implementation. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2021.

[31] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, et al. Gandiva: Introspective cluster scheduling for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 595–610, 2018.

[32] Xiaohui Xu, Sijing Duan, Jinrui Zhang, Yunzhen Luo, and Deyu Zhang. Optimizing federated learning on device heterogeneity with a sampling strategy. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, pages 1–10. IEEE, 2021.

[33] Rong Yu and Peichun Li. Toward resource-efficient federated learning in mobile edge computing. *IEEE Network*, 35(1):148–155, 2021.

[34] Binhang Yuan, Yongjun He, Jared Davis, Tianyi Zhang, Tri Dao, Beidi Chen, Percy S Liang, Christopher Re, and Ce Zhang. Decentralized training of foundation models in heterogeneous environments. *Advances in Neural Information Processing Systems*, 35:25464–25477, 2022.

[35] Yihua Zhang, Yuguang Yao, Parikshit Ram, Pu Zhao, Tianlong Chen, Mingyi Hong, Yanzhi Wang, and Sijia Liu. Advancing model pruning via bi-level optimization. *Advances in Neural Information Processing Systems*, 35:18309–18326, 2022.