FlexFL: Heterogeneous Federated Learning via APoZ-Guided Flexible Pruning in Uncertain Scenarios

Zekai Chen, Chentao Jia, Ming Hu, *Member, IEEE*, Xiaofei Xie, *Member, IEEE*, Anran Li, *Member, IEEE*, and Mingsong Chen, *Senior Member, IEEE*

***Abstract*—Along with the increasing popularity of deep learning (DL) techniques, more and more Artificial Intelligence of Things (AIoT) systems are adopting federated learning (FL) to enable privacy-aware collaborative learning among the AIoT devices. However, due to the inherent data and device heterogeneity issues, the existing FL-based AIoT systems suffer from the model selection problem. Although various heterogeneous FL methods have been investigated to enable collaborative training among the heterogeneous models, there is still a lack of 1) wise heterogeneous model generation methods for the devices; 2) consideration of uncertain factors; and 3) performance guarantee for the large models, thus strongly limiting the overall FL performance. To address the above issues, this article introduces a novel heterogeneous FL framework named FlexFL. By adopting our average percentage of zeros (APoZ)-guided flexible pruning strategy, FlexFL can effectively derive best-fit models for the heterogeneous devices to explore their greatest potential. Meanwhile, our proposed adaptive local pruning strategy allows the AIoT devices to prune their received models according to their varying resources within uncertain scenarios. Moreover, based on the self-knowledge distillation, FlexFL can enhance the inference performance of the large models by learning the knowledge from the small models. Comprehensive experimental results show that, compared to the state-of-the-art heterogeneous FL methods, FlexFL can significantly improve the overall inference accuracy by up to 14.24%. Our code can be found here https://github.com/mastlab-T3S/FlexFL.**

***Index Terms*—Artificial Intelligence of Things (AIoT), APoZ, heterogeneous federated learning (FL), model pruning, uncertain scenario.**

Manuscript received 11 August 2024; accepted 12 August 2024. This work was supported in part by the Natural Science Foundation of China under Grant 62272170; in part by the "Digital Silk Road" Shanghai International Joint Lab of Trustworthy Intelligent Software under Grant 22510750100; in part by the Shanghai Trusted Industry Internet Software Collaborative Innovation Center; and in part by the National Research Foundation, Singapore, and the Cyber Security Agency under its National Cybersecurity Research and Development Programme under Grant NCRP25-P04-TAICeN. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore and Cyber Security Agency of Singapore. This article was presented at the International Conference on Hardware/Software Codesign and System Synthesis (CODES + ISSS) 2024 and appeared as part of the ESWEEK-TCAD Special Issue. This article was recommended by Associate Editor S. Dailey. *(Corresponding author: Ming Hu.)*

Zekai Chen, Chentao Jia, and Mingsong Chen are with the MoE Engineering Research Center of Hardware/Software Co-design Technology and Application, East China Normal University, Shanghai 200062, China.

Ming Hu and Xiaofei Xie are with the School of Computing and Information Systems, Singapore Management University, Singapore (e-mail: hu.ming.work@gmail.com).

Anran Li is with the Department of Biomedical Informatics & Data Science, School of Medicine, Yale University, New Haven, CT 06520 USA.

Digital Object Identifier 10.1109/TCAD.2024.3444695

I. Introduction

ALONG with the prosperity of artificial intelligence (AI) and the Internet of Things (IoT), federated learning (FL) [1], [2], [3], [4], [5], [6] is becoming a mainstream distributed deep learning (DL) paradigm in the design AI of Things (AIoT) systems [7], [8], [9], since it enables collaborative learning among the devices without compromising their data privacy. So far, FL has been widely investigated in various AIoT applications, such as edge-based mobile computing [10], [11], real-time control [12], [13], and healthcare systems [14], [15]. Typically, an FL-based AIoT system is based on a client-server architecture involving a cloud server and numerous AIoT devices. In each FL training round, the cloud server first dispatches the latest global model to multiple selected (activated) devices for local training and then aggregates the trained local models to update the global model. Since, the communication between the cloud server and devices is based on the model gradients, FL enables the knowledge sharing among the AIoT devices without the privacy leaks.

Although the existing FL methods are promising for sharing the knowledge among the devices, they are not well-suited for the large-scale AIoT applications involving various heterogeneous devices with different available resources [16]. This is because the traditional FL methods assume that all the device models are of the same architecture. According to the Cannikin Law, only the models best fit for the weakest devices can be used for FL training. Typically, such models are of small sizes with limited inference capability, thus strongly suppressing the potential FL learning performance. To maximize the knowledge learned on the heterogeneous devices, various heterogeneous FL methods have been investigated to use heterogeneous models for local training, which can be mainly classified into two categories, i.e., *completely heterogeneous methods* and *partially heterogeneous methods*. Specifically, completely heterogeneous methods [17] apply the knowledge distillation (KD) strategies [18], [19], [20] on the heterogeneous models with totally different structures for knowledge sharing, while the partially heterogeneous methods [21], [22], [23], [24] derive heterogeneous models from the same large global model for local training and knowledge aggregation. As an example of partially heterogeneous methods, for a given large global model, HeteroFL [21] can generate heterogeneous models to fit the devices by pruning the parameters of each model layer.

1937-4151 © 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

When dealing with real-world AIoT applications, the existing heterogeneous FL methods greatly suffer from the following three problems.

1) Low-performance heterogeneous models derived by unwise model pruning strategies.
2) Inefficient or ineffective local training within uncertain scenarios.
3) Low inference performance of large models caused by resource-constrained scenarios.

Specifically, the existing methods generate heterogeneous models coarsely by pruning the parameters of each model layer with the same ratio or directly removing the entire layer. Without considering the different functions of parameters within layers, such unwise pruning strategies severely limit the performance of the heterogeneous models. Meanwhile, when encountering various uncertainty factors, such as hardware performance fluctuations caused by process variations [13], [25], dynamic resource utilization (e.g., available memory size), the traditional FL may fail in local training since they assume static device resources during FL training. Due to the inaccurate estimation of available device resources, the overall training performance can be deteriorated. Furthermore, within a large-scale AIoT application, typically large models cannot be accommodated by most resource-constrained devices. As a result, the small amount of training data will inevitably influence the inference capability of the large models. Therefore, how to wisely generate high-performance heterogeneous models to fit for uncertain scenarios is becoming an urgent issue in heterogeneous FL design.

Intuitively, to achieve high-performance pruned models, a model pruning method should delete the least significant neurons first. As a promising measure, activation information can be used to evaluate the importance of neurons, where the neurons with more activation times have greater importance. In other words, if a model layer consists of more neurons with fewer activation times, it has more parameters to be pruned. According to [26], the activation percentage of zeros (APoZ) can be used to measure the percentage of zero neuron activation times under the rectified linear unit (ReLU) mapping. Therefore, the higher the APoZ score of a model layer, the higher the pruning ratio we can apply to the layer. Based on this motivation, this article proposes a novel heterogeneous FL approach named FlexFL, which utilizes the APoZ scores of the model layers to perform finer-grained pruning to generate high-performance heterogeneous to best fit their target devices for high-quality local training. To accommodate various uncertain scenarios, FlexFL allows devices to adaptively prune their received models according to their available resources. Meanwhile, based on a self-KD-based training strategy, FlexFL enables large models to learn from the small models, thus improving their inference performance. Note that in FlexFL, the small models are derived from the large models, and the self-KD-based training is only performed by devices. In this way, FlexFL can effectively explore the greatest potential of devices, thus improving the overall FL training performance. This article makes the following four major contributions.

1) We propose an APoZ-guided flexible pruning strategy to wisely generate the heterogeneous models best for the devices.
2) We design an adaptive local pruning strategy to enable devices to further prune their local models to adapt to varying available resources within uncertain scenarios.
3) We present a self-KD-based local training strategy that utilizes the knowledge of the small models to enhance the training of the large models.
4) We perform extensive experiments based on the simulation and real test-beds to evaluate the performance of FlexFL.

II. Background and Related Work

A. Background

1) Federated Learning: FL is a distributed machine learning approach that addresses the data privacy protection and decentralization issues. Traditional FL framework usually consists of a server and multiple devices. In FL training, the server maintains a global model and dispatches it to the selected devices for local training in each round. Each device then trains locally on its own data and uploads the trained model to the server after training. Finally, the server aggregates all the received models to generate a new global model. Specifically, the optimization objective of FL is based on FedAvg [1], which is defined as follows:

$$\min_{w} F(w) = \frac{1}{|D|}\sum_{k=1}^{|D|} f_k(w),\ \text{s.t.}, f_k(w) = \frac{1}{|\mathcal{D}_k|}\sum_{i=1}^{|\mathcal{D}_k|} \ell(w, \langle x_i, y_i\rangle)$$

where $|D|$ denotes the number of devices and the function $f_k(w)$ is the loss value of the model on the device k, $|\mathcal{D}_k|$ denotes the data set size in the device k and ℓ denotes the loss function (e.g., cross-entropy (CE) loss), and w is the model parameter as well as optimization objective, and x_i and y_i, are the samples and the corresponding labels, respectively.

2) Model Pruning: In the field of machine learning and DL, model pruning is a technique to reduce the model complexity and computational resource requirements by reducing the redundant parameters and connections in neural network models. The main objective of model pruning is to achieve a more compact and efficient model without significantly sacrificing its performance. Initially, the trained model is analysed to identify parameters or connections that contribute less to the overall model performance. These parameters are considered redundant and can be pruned without affecting the model's performance. Common approaches include magnitude-based pruning [27], which removes parameters with small weights; sensitivity-based pruning [28], which measures the impact of each parameter on the model's output; and structured pruning [29], which removes entire neurons or channels.

APoZ [26] is a metric used in the model pruning to quantify the sparsity level of neural network activations. It measures the percentage of zero activations in a layer or network after applying a pruning technique. A high APoZ score indicates that a large proportion of activations in the network are zero, indicating that the network has achieved significant sparsity. In

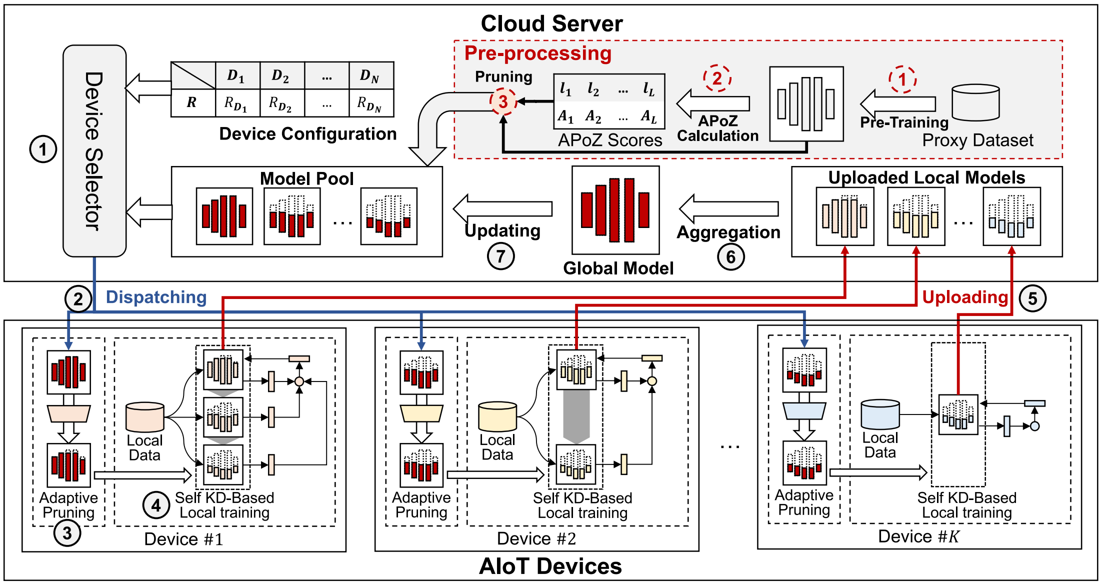


Fig. 1. Framework and workflow of FlexFL.

the essence, APoZ provides a quantitative measure of sparsity, allowing us to assess the impact of pruning methods on the neural network architectures and optimize pruning strategies to achieve the desired tradeoff between the model size and performance.

B. Model Heterogeneous Federated Learning

Model heterogeneous FL [21], [22], [24], [30] has a natural advantage in solving the systemic heterogeneity. Different from the traditional FL, model heterogeneous FL usually maintains some models of different sizes on the cloud server, so as to better deal with the heterogeneous resources of different devices. The current model heterogeneous FL methods can be classified into three categories, i.e., width-wise pruning, depth-wise pruning, and 2-D pruning. For the width-wise pruning, Diao et al. [21] proposed HeteroFL, which alleviated the problem of the device system heterogeneity by tailoring the model width and conducted parameter-averaging over the heterogeneous models. Similarly, Horváth et al. [31] proposed FJoRD, which used the ordered dropout mechanism to extract the lower footprint submodels. For the depth-wise pruning, DepthFL [22] prunes the later parts of deeper networks to reduce the number of network parameters. In InclusiveFL Liu et al. [32] proposed a layer-wise model pruning method with momentum KD to better transfer knowledge among submodels. For 2-D pruning, in ScaleFL Ilhan et al. [24] introduced a pruning method that scales from both the width and depth dimensions, aiming to balance the proportions of the model width and depth. Additionally, it incorporates skip connections to facilitate connections between the shallower models and the network's classification layers. However, the existing approaches seldom consider the characteristics of each model and the differences in neuronal activation distribution on different datasets, and most of them adopt a fixed pruning method while ignoring the different model architectures. Moreover, the existing methods largely lack consideration of device-related resource uncertainties in real-world environments. Most of them are studied under the assumption of fixed device resources, which deviates from the dynamic nature of AIoT scenarios.

To the best of our knowledge, FlexFL is the first attempt to utilize a flexible pruning strategy to generate the heterogeneous models in FL under the resource uncertainty scenarios. Using APoZ scores and the number of parameters of each layer, FlexFL can generate higher-performance heterogeneous models for local training. To deal with various uncertain and resource-constrained scenarios, FlexFL integrates an adaptive local pruning mechanism and self-KD-based local training strategy, which enables the devices to adaptively prune their received model according to their available resources and effectively improves the performance of FL training.

III. Our FlexFL Approach

A. Overview of FlexFL

Fig. 1 presents the framework and workflow of FlexFL, which consists of two stages, i.e., the *preprocessing stage* and the *FL training stage*. FlexFL maintains a large model as the global model and generates multiple heterogeneous models for local training based on the global model. The preprocessing stage aims to calculate the APoZ scores of each layer, which are used for the model pruning to generate the heterogeneous models. The FL training stage aims to train the multiple heterogeneous models, which are pruned from the global model.

As shown in Fig. 1, in the preprocessing stage, the cloud server maintains a proxy dataset to pretrain the global model and calculates the APoZ scores for each layer according to the neuron activation. Since, APoZ calculation does not require the model to be fully trained, the proxy dataset requires only a small amount of data compared to the training dataset. The cloud server then prunes the global model to generate multiple heterogeneous models based on the calculated APoZ scores. Specifically, the workflow of the preprocessing stage consists of three steps as follows.

1) *Step 1 (Pretraining):* Since, the APoZ scores are calculated based on a trained model, the cloud server uses a proxy dataset to train the global model. Note that, the proxy dataset consists of two parts, i.e., a training part and a test part.
2) *Step 2 (APoZ Score Calculation):* The cloud server inputs the test part of the proxy dataset into the pretrained model and records the activation of each neuron. Then, the could server calculates the APoZ scores of each layer based on the activation of its neurons.
3) *Step 3 (Heterogeneous Model Generation):* The cloud server first specifies multiple levels of heterogeneous models with varying parameter sizes. For example, the cloud server can specify to generate the three levels of heterogeneous models with 25% (small), 50% (medium), and 100% (large) parameters of the original global model, respectively. For each heterogeneous model, the cloud server calculates the pruning ratio for each layer according to its APoZ score, adjustment weight, and target model pruning ratio. The cloud server then generates multiple heterogeneous models by pruning the global model according to its calculated pruning ratio. Our pruning strategy ensures that a small model is a submodel of any model larger than it. The generated heterogeneous models are stored in the model pool.

The FL training stage consists of multiple FL training rounds. As shown in Fig. 1, the cloud server maintains a table to record the device resource configuration, which can be requested directly from each device. In each FL training round, the cloud server selects multiple devices for local training according to their resources. Then, the cloud server dispatches the heterogeneous models in the model pool to the selected AIoT devices. Note that, each device is dispatched with a model coupled with its APoZ scores to guide local pruning. Here, the APoZ score is an array with the length of the number of global model layers, whose communication overhead is negligible. The device adaptively prunes the model according to its currently available resources before conducting local training. To improve the performance of the large model, the devices perform a self-KD-based training strategy, which uses the output of the small models to guide the training of the large model. Note that, since small models are pruned from the large model, devices can directly obtain small models from their dispatched model. After local training, devices upload their trained model to the cloud server. The cloud server aggregates all the local models to update the global model and uses the new global model to update the models in the model pool. Specifically, as shown in Fig. 1, the workflow of each FL training round consists of seven steps as follows.

1) *Step 1 (Model and Device Selection):* The cloud server selects devices for local training and assigns a model from the pool to each device according to their resources.
2) *Step 2 (Model Dispatching):* The cloud server dispatches models from the model pool to its corresponding selected devices for local training. Note that, the cloud server also sends the APoZ scores to devices for local pruning.
3) *Step 3 (Adaptive Local Pruning):* Due to various uncertain factors, the available resources of a device may not be sufficient to enable training of the dispatched model. To facilitate local training, a device prunes its received models when its available resources are insufficient. FlexFL enables a device to prune partial parameters based on its received model. If the device cannot train the model, it will be pruned directly to a smaller model.
4) *Step 4 (Self-KD-Based Local Training):* Each device uses its local data to train the pruned model. Since, a small model is the subset of any larger model, the pruned model includes all the parameters of the small models. Small models can be trained on more devices, which means that small models are more adequately trained. To improve the performance of large models, devices calculate the loss for the large model training using the soft label of the small models together with the true label of the data samples.
5) *Step 5 (Model Uploading):* Each device uploads its trained model to the cloud server for aggregation.
6) *Step 6 (Model Aggregation):* The cloud server aggregates the corresponding parameters of the local models to generate a new global model.
7) *Step 7 (Model Pool Updating):* The cloud server uses the new global model to update the parameters of each model in the model pool.

B. APoZ-Guided Model Generation

FlexFL generates multiple heterogeneous models by pruning the global model. To generate high-performance heterogeneous models for local training, FlexFL aims to assign a higher pruning ratio to the layers with more redundant parameters. Based on this motivation, FlexFL adopts the APoZ [26] score as a metric to guide the model generation.

1) APoZ Score Calculation (APoZCal$(\cdot)$): APoZ is a metric that measures the number of neuron activations after the ReLU layer. For each ReLU layer i, APoZ is defined as

$$A_i = \frac{\sum_{j=1}^{|\mathcal{D}_p^{\text{test}}|} \sum_{k=1}^{N} f(h_k^i(\mathcal{S}_j) = 0)}{|\mathcal{D}_p^{\text{test}}| \times N} \tag{1}$$

where $f(\delta)$ is a Boolean function, which returns 1 when the Boolean statement $\delta \models \top$, N denotes the dimension of the output feature map after the ReLU layer, $|\mathcal{D}_p^{\text{test}}|$ denotes the total size of the test part of the proxy dataset, and $h_k^i(\mathcal{S}_j)$ denotes the kth output feature map of the jth sample $\mathcal{S}_j$ after the ith ReLU layer.

For models consisting of multiple residual blocks, e.g., ResNet [33] and MobileNet [34], we adjust the number of channels between the blocks. When a block contains multiple ReLU layers, we average its APoZs as the APoZ score of

Algorithm 1: Heterogeneous Model Generation

Input: i) S_{APoZ}, the set of APoZ scores for each layer; ii) M, global model, iii) L_p, list of target model pruning ratios.

Output: P, the model pool.

```
1  P ← {}
2  s[i][j] ← 1 for i ∈ [1, len(L_p)], j ∈ [1, len(S_APoZ)]
3  for i = 1, ..., len(L_p) do
4      γ ← 0
5      p_i ← L_p[i] × size(M)
6      M' ← M
7      while |p_i − size(M')| > ϵ do
8          for j = 1, ..., len(S_APoZ) do
9              ⟨l_j, A_j⟩ ← S_APoZ[j]
10             AdjW_j ← AdjWCal(l_j, M)
11             s[i][j] ← (1 − A_j × AdjW_j) × γ
12             s[i][j] ← max(min(s[i][j], 1), 0.01)
13         end
14         M' ← prune(M, s[i])
15         γ ← γ + ξ      //ξ = 0.01 is the iteration step.
16     end
17     P ← P ∪ {M'}
18 end
19 return P
```

this block. Specifically, if the ith block contains the K_i ReLU layers, the block APoZ is calculated as

$$A_i = \frac{1}{K_i}\sum_{t=1}^{K_i} \frac{\sum_{j=1}^{|\mathcal{D}_p^{\text{test}}|}\sum_{k=1}^{N} f(h_k^t(\mathcal{S}_j)=0)}{|\mathcal{D}_p^{test}| \times N}. \tag{2}$$

2) Adjustment Weight Calculation (AdjWCal$(\cdot)$): Typically, the model layers with more parameters often contain more redundant neurons, diminishing the significance of individual neurons within the layers. When pruning a specific number of neurons, prioritizing the layers with more parameters is likely to have less impact on the overall model performance compared to the layers with fewer parameters. Therefore, we calculate the adjustment weight as follows to adjust the APoZ scores:

$$\text{AdjWCal}(l_i, M) = \frac{\log \text{size}(l_i)}{\log \max\big(\text{size}(l_1), \ldots, \text{size}(l_{\text{len}(M)})\big)} \tag{3}$$

where l_i is the ith layer of M, the function size(l_i) calculates the number of parameters of the i^{th} layer, and len(M) denotes the number of layers of the model M.

3) Heterogeneous Model Generation (ModelGen$(\cdot)$): Based on the calculated APoZ scores and adjustment weights, FlexFL can prune the global model to generate the heterogeneous models. Note that, the heterogeneous model generation is performed on the server side and only once upon initialization. Algorithm 1 presents the process of the heterogeneous model generation. Lines 1 and 2 initialize the model pool P and the pruning ratios s for each target model. Lines 3–18 generates the multiple models according to the target model pruning ratios in L_p. Lines 4–6 initialize the pruning control variable γ, the target model size p_i, and the target model M', respectively. Line 7 evaluates the gap between the size of M' and a target model size p_i. Line 10 uses (3) to calculate the adjustment weight AdjW$_j$ for the layer l_j. Line 11 generates pruning ratio $s[i][j]$ based on the APoZ score A_j and the adjustment weight AdjW$_j$. In line 12, we set the minimum pruning ratios of each layer to 0.01 to avoid the parameters of a layer being completely pruned. In line 14, according to the pruning ratios $s[i][]$ generated, we prune the global model M to M'. Specifically, for the layer l_j, y_j and x_j represent the numbers of output and input channels, according to the pruning ratios $s[i][]$, we prune it to a model M' with $x_j \times s[i][j-1]$ input channels and $y_i \times s[i][j]$ output channels. Note that, if $W_i \in \mathbb{R}^{y_j \times x_j}$ is the hidden weight matrix of the global model M in the layer l_j, after pruning, $W_i' \in \mathbb{R}^{(y_j \times s[i][j]) \times (x_j \times s[i][j-1])}$ is the new hidden weight matrix of the pruned model M' in the layer l_j. In line 17, for M' with an error less than or equal to ϵ, we add M' to the model pool P as the pruned model corresponding to the target model pruning ratio $L_p[i]$.

C. Adaptive Local Model Pruning (AdaPrune$(\cdot)$)

To address the problem of insufficient available resources within uncertain scenarios, FlexFL enables devices to adaptively prune their received models to the adaptive models for local training, focusing on the memory resource constraints. Specifically, when a device does not have sufficient memory resources for training its received model, it first prunes $\Gamma \times$ *size*(M) parameters to generate an adaptive model for local training, where Γ is the adaptive pruning size and *size*(M) is the number of parameters of the global model M. Note that, our approach requires that the size of the pruned parameters (i.e., $\Gamma \times$ size(M)) should be smaller than the smallest size of the parameter differences between any two models. When its available resources are still insufficient to train the pruned model, the device directly prunes it to a smaller model that best fits the device. For example, assume that M_1, M_2, and M_3 denote the small, medium, and large models, respectively. Let M_2' and M_3' be the adaptive models of M_2 and M_3, respectively. If a model of type M_3 is dispatched to some device with insufficient resources, the client will adaptively prune the model in the order of M_3', M_2, M_2', and M_1 until the pruned model can be accommodated by the device.

Similar to Algorithm 1, the adaptive pruning is still based on our calculated APoZ scores. Since, our APoZ scores are calculated before FL training and are not updated within the training process, the cloud server can directly send the APoZ scores to all the devices. In addition, since all the heterogeneous models in FlexFL are pruned from the same global model without any extra exit layers, and a smaller model is a submodel of a larger model, the devices can prune the received model according to the corresponding pruning scheme.

D. Self-Knowledge Distillation-Based Local Training

In resource-constrained scenarios, the majority of devices cannot train the large models, which results in inadequate training for the large models. To improve the performance of the large models, FlexFL adopts a KD [19], [35] strategy. Since, small models are the submodel of the large models, the devices can utilize adequately trained small models to enhance the training of large training. Specifically, during the model training, the device can obtain the outputs (i.e., soft labels) of the large model and small models. Then, the device calculates the CE loss using the output of the large model and true labels and calculates the Kullback–Leibler (KL) loss based on the outputs of the large and small models. Finally, the device uses both the CE and KL losses to update the model. Assume that M_i is a large model, $\hat{y}_i$ is the soft labels of the model M_i and y_i is ground truth, the CE loss of M_i is defined as

$$\mathcal{L}_{\mathrm{CE}} = -\log h\big(\hat{y}_i\big)\big[y_i\big]$$

where $h(\cdot)$ is the softmax function. Assume that, M_1, M_2, ..., M_{i-1} are the smaller models for M_i and $\hat{y}_1$, $\hat{y}_2$, ..., $\hat{y}_{i-1}$ are the soft labels of these models. The KL loss can be calculated as follows:

$$\mathcal{L}_{\mathrm{KL}} = \frac{1}{i-1}\sum_{j=1}^{i-1} \mathrm{sum}\left(h\big(\hat{y}_j\big)/\tau\right) \cdot \tau^2 \log \frac{h\big(\hat{y}_j\big)}{h\big(\hat{y}_i\big)}$$

where τ is the temperature to control the distillation process. Note that, a higher value of τ leads to smoother probability distributions, making the model focus more on relatively difficult samples, and a lower value of τ makes the probability distribution sharper, making the model more confident and prone to overfitting.

According to the CE and KL losses, we can obtain the final loss $\mathcal{L}$ of the model M_i as follows:

$$\mathcal{L} = \mathcal{L}_{\mathrm{CE}} + \lambda \mathcal{L}_{\mathrm{KL}} \tag{4}$$

where λ is a hyperparameter that controls the training preference for the two types of losses. Note that, a large value of λ makes the model training more influenced by the KL divergence loss. On the contrary, a small value of λ guides the training to focus more on the CE loss.

E. Heterogeneous Model Aggregation (Aggr$(\cdot)$)

When all the local models are received and saved in S_{upload}, the cloud server can perform the aggregation. Since, all the heterogeneous models are pruned from the global model, the cloud server generates a new global model by aggregating the corresponding parameters of the models in S_{upload} with the weights determined by the number of its trained data.

Assume that p is a parameter in the global model M, and $\sigma(p, S_{\mathrm{upload}})$ extracts the model in the set that contains the corresponding parameter of p and the numbers of their training data. Let θ be the parameters of the aggregated global model M, which can be calculated as follows:

$$\forall p \in \theta,\ p = \frac{\sum_{m \in \sigma(p, S_{\mathrm{upload}})} p^m \times d^m}{\sum_{m \in \sigma(p, S_{\mathrm{upload}})} d^m}$$

where p^m denotes the corresponding parameter of p in m and d^m is the number of training data of m.

By applying the above equation to aggregate all the parameters of models in S_{upload}, the cloud server can generate an updated global model. Subsequently, the cloud server updates all the heterogeneous models in the model pool P by assigning the parameter values of the aggregated global model to their corresponding parameters.

Algorithm 2: Implementation of FlexFL

Input: i) T, training rounds; ii) D, the set of devices; iii)f, fraction of selected devices; iv) M, global model; v) $\mathcal{D}_p$, proxy dataset, vi) L_p, list of pruning ratios of heterogeneous models.

```
1  T_r ← ResConfRequest(D)
2  S_APoZ ← APoZCal(M, D_p)
3  Reset(M)
4  {M_1, M_2, ..., M_p} ← ModelGen(S_APoZ, M, L_p)
5  P ← {M_1, M_2, ..., M_p}
6  for epoch e = 1, ..., T do
7      K ← max(1, f|D|)
8      S_d ← DevSel(D, P, K)
9      S_upload ← {}
10     /*parallel for*/
11     for ⟨d_k, m_k⟩ in S_d do
12         r_{d_k} ← ResRequest(d_k)
13         m'_k ← AdaPrune(r_{d_k}, m_k)
14         L ← L(m'_k, D_k)
15         θ'_{m_k} ← θ'_{m_k} − ∂L/∂θ'_{m_k}
16         S_upload ← S_upload ∪ {⟨m'_k, |D_k|⟩}
17     end
18     M ← Aggr(S_upload)
19     P ← update(P, M)
20 end
```

F. Implementation of FlexFL

Algorithm 2 presents the implementation of our FlexFL approach. Lines 1–5 denote the operations of preprocessing. Line 1 initializes the resource configuration table, where the function *ResConfRequest*$(\cdot)$ requests all the devices to upload their available resource information. In line 2, the function *APoZCalculate*$(\cdot)$ pretrains the global model M using the training part of the proxy dataset $\mathcal{D}_p^{\mathrm{train}}$ and calculates the APoZ scores for each layer of M using the test part of the proxy dataset $\mathcal{D}_p^{\mathrm{test}}$, where S_{APoZ} is a set of two-tuples $\langle l_i, A_i \rangle$, l_i denotes the ith layer of M, and A_i denotes the APoZ score of the ith layer. Line 3 resets the global model. Line 4 generates len(L_p) heterogeneous models according to the calculated APoZ scores. Line 5 stores the generated models in the model pool P. Lines 6–20 present the process of the FL training stage. Line 7 calculates the number of devices needed to participate in local training. In line 8, the function *DevSel*$(\cdot)$ selects K devices and their respective trained models, where S_d is a set of two tuples $\langle d, m \rangle$, $d \in D$ is a selected device, and $m \in P$ is a model that will be dispatched to d. Line 9 initializes the model set S_{upload}, a set of two tuples $\langle m, \mathrm{num} \rangle$, where m is a local model and *num* is the number of data samples used to train m. Lines 11–17 present the local training process. Line 12 requests the current resources of d_k and Line 13 uses our adaptive local pruning strategy to prune the received model m_k according to r_{d_k}. Line 14 employs (4) to calculate the loss L and line 15 updates the parameters of m'_k according to L, where θ'_{m_k} denotes the parameters of m'_k. In line 16, the device uploads its trained model m'_k together with the number of data

TABLE I
DEVICE UNCERTAINTY SETTINGS

Level	# Device	Maximum Capacity r_M	Variance u
Weak	40%	$r_M = 35$	$\sigma^2 \in [5, 8, 10]$
Medium	30%	$r_M = 60$	$\sigma^2 \in [5, 8, 10]$
Strong	30%	$r_M = 110$	$\sigma^2 \in [5, 8, 10]$

samples $|\mathcal{D}_k|$ to the model set S_{upload}. Line 18 aggregates all the models in S_{upload} to update the global model M. Line 19 updates the models in P using the global model M.

IV. PERFORMANCE EVALUATION

To evaluate FlexFL performance, we implemented FlexFL using PyTorch. For all the investigated FL methods, we adopted the same SGD optimizer with a learning rate of 0.01 and a momentum of 0.5. For local training, we set the batch size to 50 and the local epoch to 5. We assumed that $|D| = 100$ AIoT devices were involved in total, and $f = 10\%$ of them were selected in each FL training round. All the experiments were conducted on an Ubuntu workstation with one Intel i9 13900k CPU, 64 GB memory, and one NVIDIA RTX 4090 GPU.

A. Experimental Settings

1) Device Heterogeneity Settings: To evaluate the performance of FlexFL in uncertain and resource-constrained scenarios, we simulated various devices with different dynamic resources (available memory size). Specifically, we employed the Gaussian distribution to define dynamic device resources as follows: $r = r_M - |u|$, where r_M is the maximum memory capacity of the device and $u \sim \mathcal{N}(0, \sigma^2)$.

In our experiment, we adopted three levels of devices, i.e., weak, medium, and strong. We set the ratio of the number of devices at these three levels to 40%, 30%, and 30%, respectively. The distribution of their available memory size across these devices is uncertain as shown in Table I. If the device memory is smaller than its received model m, i.e., $r \leq (\text{size}(m)/\text{size}(M)) \times 100$, our approach will not train m due to insufficient memory resources. In this case, m will be pruned to be an adaptive model to fit the device. For example, if r is 30, the number of parameters of a pruned model cannot exceed 30% of its original counterpart.

2) Data Settings: In our experiments, we utilized three well-known datasets, i.e., CIFAR-10 [36], CIFAR-100 [36], and TinyImagenet [37]. To investigate the performance on non-IID scenarios, we adopted the Dirichlet distribution Dir(α) to assign the data to the devices involved. By controlling the hyperparameter α of the Dirichlet distribution, we managed the degree of the IID bias in the data, where the smaller values of α indicate higher data heterogeneity.

3) Model Settings: To validate the generality of our method, we conducted experiments under the models of different sizes and different architectures, i.e., VGG16 [38], ResNet34 [33], and MobileNetV2 [34].

We adopted $p = 3$ and uniformly set the list of target model pruning ratios L_p to [25%, 50%, 100%] for all the methods, yielding a model pool $P = \{M_1, M_2, M_3\}$ with three

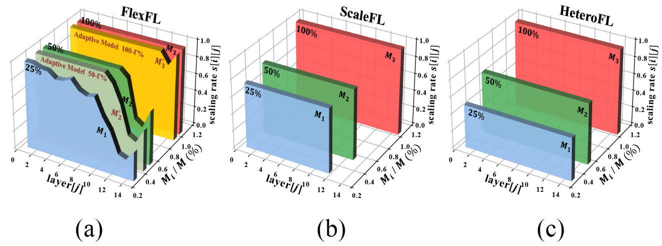


Fig. 2. Comparison of submodels (VGG16 on CIFAR10). (a) FlexFL. (b) ScaleFL. (c) HeteroFL.

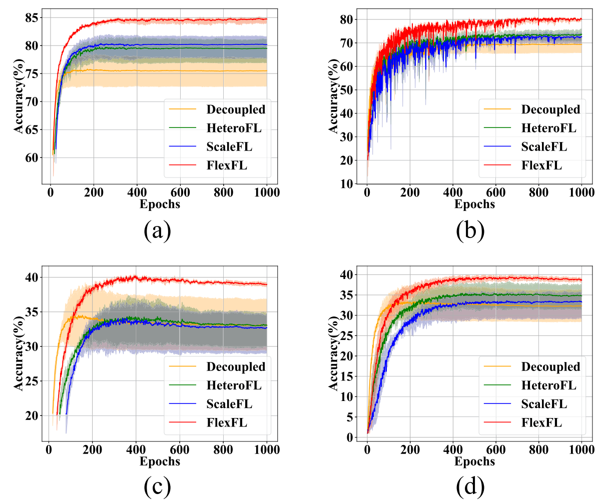


Fig. 3. Learning curves of FlexFL and three baselines. (a) CIFAR10, IID. (b) CIFAR10, $\alpha = 0.3$. (c) CIFAR100, IID. (d) CIFAR100, $\alpha = 0.3$.

models, where the model M_3 is the largest model and also the global model. Fig. 2 presents an example to visualize the pruned models based on different FL methods. In the case of FlexFL, the adaptive models are M_2' and M_3', which are obtained by pruning $\Gamma \times \text{size}(M_3)$ parameters from M_2 and M_3, respectively, where $\Gamma = 10\%$. For self-KD hyperparameters, we set $\tau = 3$ and $\lambda = 10$.

B. Performance Comparison

In our experiment, we compared three methods to our method: 1) decoupled [1]; 2) HeteroFL [21]; and 3) ScaleFL [24]. Decoupled follows a strategy similar to FedAvg [1], where large, medium, and small models are trained on the devices capable of hosting them without considering the model aggregation. HeteroFL generates corresponding models based on the width-wise pruning. ScaleFL, on the other hand, the prunes models based on both the width and depth proportions to create their corresponding models. Fig. 3 illustrates the comparison between the accuracy of our method and three other baselines. The solid line in the middle represents the average accuracy of the 25%, 50%, and 100% models, and the boundaries filled with corresponding colors represent the highest and lowest accuracies among all the models.

1) Large Model Performance Analysis: In Table II, we use the notation "x/y" to specify the test accuracy, where x denotes the average accuracy of the 25%, 50%, and 100% models and y indicates the accuracy of the 100% model.

TABLE II
TEST ACCURACY (%) OF AVERAGE AND LARGE MODELS. THE BEST RESULTS ARE SHOWN IN **BOLD**

Model	Algorithm	CIFAR10			CIFAR100			TinyImagenet		
		IID	$\alpha = 0.6$	$\alpha = 0.3$	IID	$\alpha = 0.6$	$\alpha = 0.3$	IID	$\alpha = 0.6$	$\alpha = 0.3$
VGG16	Decoupled	75.81/72.76	73.57/69.91	69.95/66.01	34.53/29.44	33.68/29.12	33.33/28.77	25.62/21.14	26.56/22.53	26.70/23.51
	HeteroFL	79.75/76.92	77.53/75.45	73.89/71.50	34.37/29.92	35.02/31.31	35.43/31.56	25.51/23.19	26.63/25.01	27.90/26.78
	ScaleFL	80.36/77.72	75.99/74.00	72.95/69.87	34.10/29.00	34.26/29.38	33.60/29.18	22.37/19.38	23.15/20.96	24.67/21.62
	FlexFL	**84.75/85.16**	**83.11/83.45**	**80.60/81.06**	**40.27/40.35**	**41.29/41.30**	**39.55/39.99**	**26.61/26.97**	**29.19/29.56**	**31.07/31.42**
Resnet34	Decoupled	62.92/56.97	59.01/54.11	55.35/51.32	26.88/22.28	26.68/22.01	25.24/19.81	33.13/29.07	32.95/25.03	31.33 24.80
	HeteroFL	69.56/63.17	65.14/61.25	61.29/57.02	31.79/24.52	31.01/25.37	30.92/24.68	38.35/31.90	36.52/33.19	35.02/32.67
	ScaleFL	76.65/74.23	71.57/68.50	66.56/61.67	36.84/31.87	34.95/29.51	32.81/27.50	38.31/32.84	36.68/31.32	36.02/31.77
	FlexFL	**77.48/78.06**	**72.89/73.71**	**69.08/69.60**	**37.76/37.38**	**37.63/37.31**	**37.34/37.38**	**40.53/40.85**	**38.64/38.32**	**37.89/37.88**
MobileNetV2	Decoupled	53.01/52.03	48.34/47.71	42.42/40.29	20.20/17.74	20.74/18.08	20.22/17.58	21.57/16.11	20.34/17.60	20.11 16.35
	HeteroFL	57.60/52.69	51.31/48.33	44.00/40.16	23.49/19.31	22.60/19.20	21.60/17.80	24.96/20.05	24.96/22.23	22.52/20.82
	ScaleFL	63.42/59.83	54.57/48.77	49.90/45.10	26.53/21.72	25.09/19.90	24.30/18.24	26.64/23.63	26.31/23.43	24.69/22.25
	FlexFL	**68.47/69.18**	**60.00/61.32**	**56.87/58.23**	**29.11/28.30**	**27.86/27.14**	**26.78/26.41**	**28.26/27.44**	**27.46/25.55**	**25.38/24.00**

It is evident that FlexFL consistently achieves an accuracy improvement ranging from 1.75% to 13.13% in terms of large model accuracy, regardless of whether in the IID or non-IID scenarios. This indicates that our method performs better in obtaining high accuracy on the larger models. This discrepancy can be attributed to the greater flexibility in pruning offered by VGG16 and MobileNetV2. Specifically, VGG16 permits pruning of up to the first 15 layers, whereas MobileNetV2 allows for pruning of up to nine blocks. In contrast, due to the inability to disrupt the internal structure of residual blocks in ResNet34, we can only prune five blocks, resulting in a relatively smaller improvement compared to the baselines.

2) Average Model Performance Analysis: Our experiment also evaluated the average model accuracy as shown in Table II. Based on our observations of the datasets, our approach demonstrates an accuracy improvement ranging from 0.92% to 7.65% compared to ScaleFL. We also observe that in most datasets, the accuracy of the largest model is higher than the average model. Conversely, in the ScaleFL, HeteroFL, and decoupled approaches, the accuracy of the largest model is lower than that of the average model. This indicates that in our approach, large models can effectively leverage their greater number of parameters, while in the other methods, the large models exhibit a paradoxical scenario where they possess more parameters but lower accuracy compared to the smaller or medium-sized models. This phenomenon arises from the fact that the small models can be trained on all the devices, whereas the large models can only be trained on the devices with ample resources. Consequently, the small models encapsulate a broader spectrum of knowledge. In our approach, by distilling knowledge from the large models to smaller ones, the larger models can enhance accuracy by assimilating knowledge from the other models.

C. Impacts of Different Configurations

1) Proxy Dataset Size: To evaluate the impact of the pruning ratio s on different proxy dataset sizes, we used the training datasets of sizes 100%, 50%, 20%, 10%, 5%, and 1% as the proxy datasets, where 80% of the proxy dataset was used as the training set for pretraining. After training for 100 rounds, the remaining portion was used as the test set to calculate the APoZ scores. Based on Algorithm 1, we compared the similarity of pruning ratio $s_{p\%}[i]$ obtained from the model M_i in the model pool P, where $p\%$ represents the

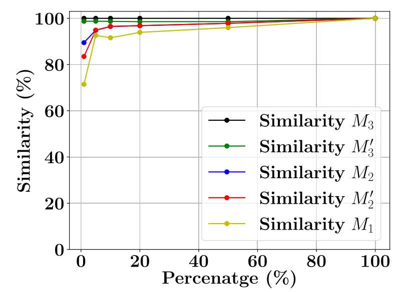


Fig. 4. Model pruning ratios similarity with different proxy dataset sizes.

TABLE III
TEST ACCURACY (%) IN DIFFERENT PROXY DATASET SIZE

Proxy Dataset Size	Global Model Accuracy	Avg Model Accuracy
1%	84.90%	85.19%
5%	84.37%	84.63%
10%	84.11%	84.31%
20%	84.60%	84.88%
50%	84.02%	84.37%
100%	84.56%	84.90%

proxy dataset size. The similarity is defined as

$$\mathrm{sim}_{(p\%, M_i)} = 1 - \mathrm{avg}\left(\frac{|s_{p\%}[i] - s_{100\%}[i]|}{s_{100\%}[i]}\right).$$

Fig. 4 shows the similarity for each level model M_i. We can observe that even with only 1% of the data, FlexFL achieves pruning ratios similar to those using the full dataset.

We conducted heterogeneous FL training using different model pools P generated by different proxy dataset sizes. As shown in Table III, FlexFL achieves inference accuracy similar to that of the full dataset when using only 1% data as a proxy dataset. Therefore, FlexFL can achieve good performance only by using a very small proxy dataset.

2) Numbers of Involved Devices: To investigate the scalability of our approach in various heterogeneous FL scenarios, we studied the impact of varying numbers of involved devices on the inference accuracy. Specifically, we conducted experiments based on CIFAR10 and VGG16 within the IID scenarios involving $|D| = 50$, 100, 200, and 500 devices, respectively. In each training round, 10% of the devices were selected. Fig. 5 shows that our method consistently improves the performance across different numbers of devices. Moreover, as $|D|$ increases, the accuracy of all the methods decreases. And

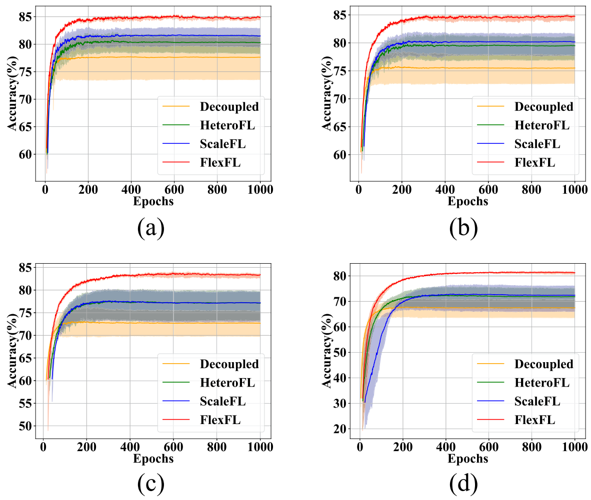


Fig. 5. Learning curves for different numbers of involved devices. (a) $|D| = 50$. (b) $|D| = 100$. (c) $|D| = 200$. (d) $|D| = 500$.

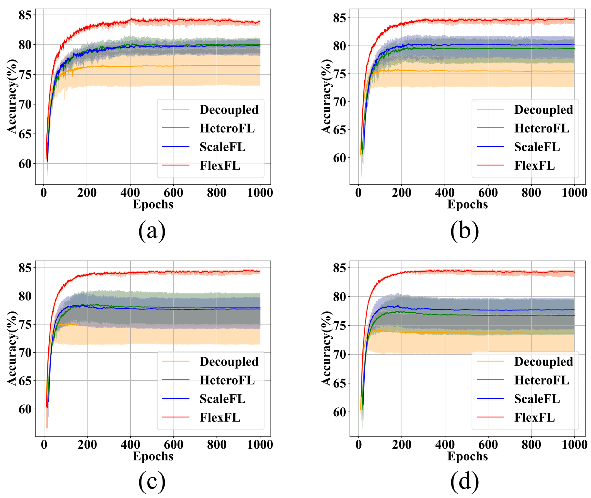


Fig. 6. Learning curves for different ratios of selected devices. (a) $f = 5\%$. (b) $f = 10\%$. (c) $f = 20\%$. (d) $f = 50\%$.

it indicates that our method is more practical and adaptable to the heterogeneous FL scenarios with large numbers of devices.

3) Numbers of Selected Devices: Assume that there are a total of $|C| = 100$ devices. Fig. 6 compares the training performance of the FL methods considering different ratios f of selected devices based on the CIFAR10 and VGG16 within an IID scenario. We can find that our approach achieves the best performance in all the four cases. Moreover, as the ratio f increases, the accuracy of our method remains stable, and the differences between the accuracy of the large and small models are the smallest.

4) Proportions of Different Devices: We compared our method with three baselines in terms of accuracy under different proportions of devices as shown in Fig. 7. We categorized all the devices into three groups, and the uncertainty configurations for each group of devices are shown in Table I. We varied the device proportions to 1:1:8, 1:8:1, and 8:1:1. According to our experimental results, we observed that our method outperformed the baselines in terms of the average

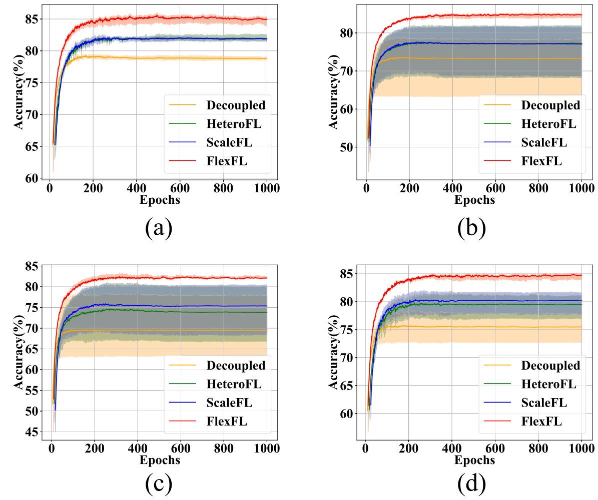


Fig. 7. Learning curves for different proportions (small:medium:large) of devices using VGG on CIFAR10 in the IID scenario. (a) 1:1:8. (b) 1:8:1. (c) 8:1:1. (d) 4:3:3.

TABLE IV
TEST ACCURACY COMPARISON WITH DIFFERENT Γ

Γ	Global Accuracy	Avg Accuracy
1%	84.38%	84.22%
2%	84.86%	84.65%
5%	84.88%	84.67%
10%	85.16%	84.75%
15%	84.40%	84.21%

model accuracy across all the device proportions. Furthermore, as the device proportions changed, our method's accuracy remained relatively stable, while the other baselines exhibited performance degradation.

5) Adaptive Local Model Pruning Size: We conducted experiments on the CIFAR10 within an IID scenario with VGG16 to evaluate the impact of adaptive pruning sizes Γ. The experimental results are presented in Table IV. We can find that the optimal value for the hyperparameter Γ in our experiments is 10%. This is mainly because a low value of Γ leads to lower utilization of the adaptive models, meaning more devices degrade their received models directly to the smaller models. In contrast, although a high value of Γ can improve the utilization of adaptive models, it causes smaller sizes of the adaptive models, which results in a lower utilization of resources.

6) Self-KD Hyperparameter Settings: We investigated the impact of the hyperparameter λ in self-KD on our experiments. In our experimental setup, we fixed the temperature parameter $\tau = 3$ for self-KD and varied the coefficient of KL-loss λ during local training as $\lambda = 0, 5, 10, 20, 50$.

Fig. 8 shows that without self-KD ($\lambda = 0$), the accuracy of our method decreased by approximately 2–4% compared to when the distillation was used. Furthermore, with increasing distillation coefficient λ, the accuracy showed an initial increase followed by a decreasing trend in both the IID and non-IID scenarios. When the distillation coefficient λ is at a reasonable range, i.e., $\lambda \in [10, 20]$, the overall loss can be well balanced between the distillation loss and CE loss. When λ is

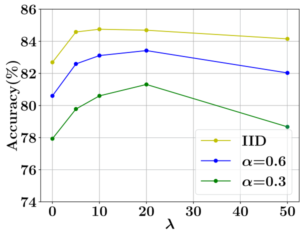


Fig. 8. Average accuracy of VGG16 on CIFAR10 with different λ.

TABLE V
CONFIGURATIONS OF DIFFERENT DEVICE RESOURCE DISTRIBUTIONS

Configuration	# Device	Max Capacity r_M	Variance u
Conf1	40%	35	$\sigma^2 = 0$
	30%	60	$\sigma^2 = 0$
	30%	110	$\sigma^2 = 0$
Conf2	40%	35	$\sigma^2 \in [5, 8, 10]$
	30%	60	$\sigma^2 \in [5, 8, 10]$
	30%	110	$\sigma^2 \in [5, 8, 10]$
Conf3	40%	35	$\sigma^2 \in [10, 20, 30]$
	30%	60	$\sigma^2 \in [10, 20, 30]$
	30%	110	$\sigma^2 \in [10, 20, 30]$

TABLE VI
TEST ACCURACY (%) OF MODELS IN DIFFERENT CONFIGURATIONS

Configuration	Algoroithm	Accuarcy		
		IID	$\alpha = 0.6$	$\alpha = 0.3$
Conf1	Decoupled	75.83/72.00	73.20/69.62	70.23/65.20
	HeteroFL	79.08/76.21	76.91/74.27	73.98/69.96
	ScaleFL	80.14/77.52	76.71/74.15	72.24/69.27
	FlexFL	**84.46/84.64**	**83.36/83.70**	**80.70/81.17**
Conf2	Decoupled	75.81/72.76	73.57/69.91	69.95/66.01
	HeteroFL	79.75/76.92	77.53/75.45	73.89/71.50
	ScaleFL	80.36/77.72	75.99/74.00	72.95/69.87
	FlexFL	**84.75/85.16**	**83.11/83.45**	**80.60/81.06**
Conf3	Decoupled	76.90/73.16	74.38/69.42	71.35/66.27
	HeteroFL	79.23/77.67	76.49/73.16	73.12/71.36
	ScaleFL	80.24/78.79	76.37/74.96	73.08/71.50
	FlexFL	**84.31/84.74**	**83.70/84.03**	**81.21/81.58**

set to a high value, the overall loss is dominated by distillation loss, which hinders effective learning of knowledge from the local dataset, resulting in an accuracy decrease.

7) Different Settings of Resource Distributions: To explore our method's adaptability in different resource allocation scenarios, we constructed three distinct configuration plans as shown in Table V. *Conf1* indicates a constant number of resources for each device, *Conf2* indicates slight fluctuations in the resources of devices, and *Conf3* suggests significant resource fluctuations across devices.

We conducted a study comparing the accuracy differences between our method and three baseline methods with results shown in Table VI. Our method achieved approximately a 4% performance improvement compared to ScaleFL across all the configs, indicating that our method can maintain high accuracy when dealing with various degrees of resource fluctuations.

8) Real-World Datasets: To validate the generalization ability of our approach, we extended our experiments to include the real-world datasets, i.e., FEMNIST [39] and Widar [40], in addition to the image recognition datasets. The FEMNIST dataset comprises 180 devices, with each training round selecting 10% devices. The data distribution on the devices is naturally non-IID. We assumed that the

TABLE VII
TEST ACCURACY (%) COMPARISON ON REAL-WORLD DATASETS

Model	Algorithm	FEMNIST	WIDAR		
			IID	$\alpha = 0.6$	$\alpha = 0.3$
VGG16	Decoupled	78.70/71.63	67.51/60.80	66.41/60.28	64.12/58.47
	HeteroFL	**79.84**/72.54	70.81/67.35	68.82/64.90	**67.28**/64.11
	ScaleFL	70.85/63.61	69.99/67.80	68.42/65.95	64.50/62.99
	FlexFL	77.16/**75.94**	**71.66/72.13**	**71.04/70.92**	67.20/**68.11**
Resnet34	Decoupled	74.02/64.27	63.54/58.36	60.65/54.89	58.84/53.84
	HeteroFL	76.99/68.07	67.90/63.05	65.17/60.59	59.83/56.71
	ScaleFL	76.94/69.25	64.77/61.17	61.44/58.92	58.97/57.67
	FlexFL	**83.64/80.46**	**72.43/73.10**	**71.19/71.57**	**67.91/68.77**
MobileNetV2	Decoupled	68.40/56.33	51.28/44.45	45.92/44.04	43.76/38.92
	HeteroFL	70.94/61.87	56.49/53.05	51.74/49.15	47.67/45.02
	ScaleFL	71.38/62.02	58.17/55.01	53.13/47.45	46.93/44.12
	FlexFL	**77.25/71.38**	**64.92/65.67**	**63.43/63.98**	**57.61/59.26**

Widar dataset involves 100 devices following given Dirichlet distributions, and ten devices are selected for local training in each FL round. We applied the uncertainty settings in Table I to all devices.

The results presented in Table VII demonstrate the performance of our method on ResNet34 and MobileNetV2, with performance improvements of up to 10.63%. There is minimal difference between the average model accuracy and the accuracy of the best-performing model. On VGG16, although FlexFL exhibits a 2.68% lower average accuracy compared to HeteroFL on the FEMNIST dataset, FlexFL still demonstrates improved accuracy for the largest model.

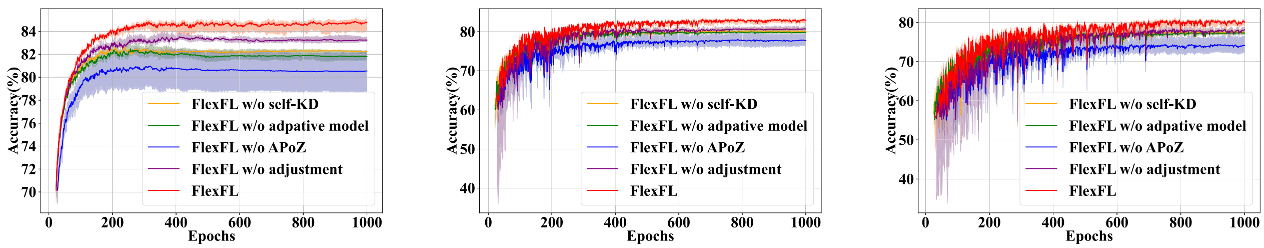
D. Ablation Study

We conducted a study on the effectiveness of each component within our method to investigate their respective impacts on the accuracy of our approach. We designed four varieties of FlexFL: 1) "w/o self-KD" indicates the absence of self-distillation during local training; 2) "w/o adaptive model" implies the utilization of only models M_1, M_2, and M_3, with the current model M_i being pruned to the model M_{i-1} under the resource constraints; 3) "w/o APoZ" involves only using adjustment weight $AdjW$ for the model pruning; and 4) "w/o adjustment" entails utilizing APoZ for model pruning without adjustment weight $AdjW$. Fig. 9 shows the superiority of FlexFL against its four variants, indicating that the absence of our proposed components will decrease model accuracy, with the lack of APoZ causing the most significant decline.

E. Evaluation on Real Test-Bed

To demonstrate the effectiveness of FlexFL in real AIoT scenarios, we conducted experiments on a real test-bed platform, which consists of 17 different AIoT devices and a cloud server. Table VIII shows the details of the configuration of these AIoT devices. Based on our real test-bed platform, we conducted experiments with a non-IID $\alpha = 0.1$ scenario on CIFAR10 [36] dataset using MobileNetV2 [34] models and selected ten devices to participate in local training in each FL training round. We set an uniform time limit of 70 000 s for FlexFL, ScaleFL, and HeteroFL.

Fig. 10 illustrates our real test-bed devices and the learning curves of the methods. From Fig. 10(b), we can observe that FlexFL achieves the highest accuracy compared to ScaleFL and HeteroFL. In addition, we can also find that compared

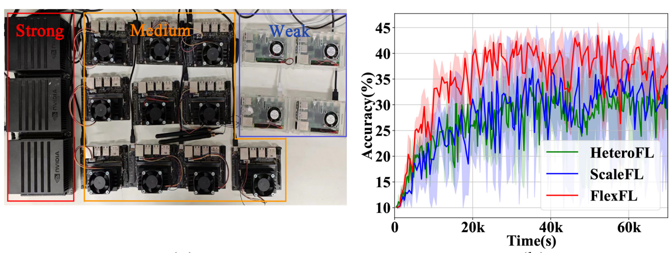


(a) (b) (c)

Fig. 9. Ablation study results for FlexFL (VGG on CIFAR10). (a) IID. (b) $\alpha = 0.6$. (c) $\alpha = 0.3$.

TABLE VIII
REAL TEST-BED PLATFORM CONFIGURATION

Device	Comp	Mem	Num
Raspberry Pi 4B	ARM Cortex-A72 CPU	2G	4
Jetson Nano	128-core Maxwell GPU	8G	10
Jetson Xavier AGX	512-core NVIDIA GPU	32G	3
Workstation	NVIDIA RTX 4090 GPU	64G	1



(a) (b)

Fig. 10. Real test-bed devices and the learning curves. (a) Real test-bed platform. (b) Learning curves.

TABLE IX
TIME OVERHEAD OF COMPONENTS PER ROUND (VGG ON IID CIFAR10)

Method	Dispatched Model Size	Adaptive Pruning	Local Training	Total
FlexFL	100%	1.45s (1.3%)	108.59s (98.7%)	110.04s
	50%	1.45s (1.9%)	73.92s (98.1%)	75.37s
	25%	0s (0%)	46.06s (100%)	46.06s
	Avg	1.45s (2.0%)	70.37s (98.0%)	71.82s
FlexFL w/o self-KD	100%	1.44s (2.2%)	64.17s (97.8%)	65.61s
	50%	1.45s (2.6%)	52.88s (97.4%)	54.33s
	25%	0s (0%)	45.77s (100%)	45.77s
	Avg	1.44s (2.8%)	50.61s (97.2%)	52.05s

TABLE X
TRAINING TIME AND COMMUNICATION OVERHEAD TO ACHIEVE THE SAME ACCURACY (VGG ON IID CIFAR10)

Target Accuracy		70%	75%	80%	85%
FlexFL	Time (s)	4849	6857	11736	32810
	Dispatch (MB)	18337	26580	45860	132298
	Upload (MB)	17751	25914	44736	129075
FlexFL w/o self-KD	Time (s)	2566	4013	7985	N/A
	Dispatch (MB)	16520	28767	59857	N/A
	Upload (MB)	16022	27973	58208	N/A
ScaleFL	Time (s)	9566	14808	77926	N/A
	Dispatch (MB)	35935	55727	300499	N/A
	Upload (MB)	34252	53226	288020	N/A
HeteroFL	Time (s)	4900	8328	N/A	N/A
	Dispatch (MB)	26412	45624	N/A	N/A
	Upload (MB)	24023	41822	N/A	N/A
All Large	Time (s)	3821	N/A	N/A	N/A
	Dispatch (MB)	37684	N/A	N/A	N/A
	Upload (MB)	37684	N/A	N/A	N/A

with the two baselines, FlexFL has relatively small accuracy fluctuations. Therefore, compared to ScaleFL and HereroFL, FlexFL still achieves the best inference accuracy and stability on the real test-bed platform.

F. Discussion

1) Computation Overhead: To evaluate the computation overhead of components (i.e., preprocessing, adaptive local pruning, and self-KD local training) introduced by FlexFL, we conducted various experiments in our real test-bed platform to investigate their impacts on the overall training time. Specifically, we evaluated from two aspects: 1) time overhead of components per FL training round and 2) training time to achieve a specific test accuracy. From Table IX, we can find that the introduction of self-KD will result in longer local training time. Note that, our method's preprocessing time accounts for approximately 0.3% of the total training time (219 s for 1000 rounds of training), and its adaptive pruning time accounts for about 2% of the local time. Overall, the computational overhead of adaptive pruning and pretraining is almost negligible, and the main additional computational overhead of FlexFL comes from self-KD. However, as shown in Table X, FlexFL needs much less training time to achieve a specific test accuracy than HeteroFL and ScaleFL. Specifically, compared with ScaleFL, FlexFL can achieve a training speedup of up to 6.63×. Here, "all large" represents the results obtained by only dispatching the largest models for the FL training under FedAvg, and the notation "N/A" means not available. Moreover, FlexFL can achieve an accuracy of 85%, while all the other methods fail, mainly benefit from the performance improvement brought about by our proposed self-KD technique.

2) Communication Overhead: From Table X, we can observe that FlexFL achieves the optimal communication overhead under all the target accuracy levels and can reduce the communication overhead (Dispatch+Upload) by up to 85%. Since, APoZ-guided pruning, adaptive local pruning, and self-distillation do not rely on any extra complex data structures, the additional memory overhead they introduce is negligible. Note that, APoZ-guided pruning is performed on the server side and only once upon initialization, it does not impose any burden on the devices.

V. CONCLUSION

Due to the lack of strategies to generate high-performance heterogeneous models, existing heterogeneous FL suffers from low inference performance, especially for various uncertain scenarios. To address this problem, this article presents a novel heterogeneous FL approach named FlexFL, which adopts an APoZ-guided flexible pruning strategy to wisely generate heterogeneous models to fit various heterogeneous AIoT devices. Based on our proposed adaptive local pruning mechanism, FlexFL enables devices to further prune their received models to accommodate various uncertain scenarios. Meanwhile, FlexFL introduces an effective self-KD-based local training strategy, which can improve the inference capability of large models by learning from small models, thus boosting the overall FL performance. Comprehensive experimental results obtained from simulation- and real test-bed-based AIoT systems show that our approach can achieve better inference performance compared with state-of-the-art heterogeneous FL methods.

REFERENCES

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Int. Conf. Artif. Intell. Statist. (AISTATS)*, 2017, pp. 1273–1282.

[2] A. Li, R. Liu, M. Hu, L. A. Tuan, and H. Yu, "Towards interpretable federated learning," 2023, *arXiv:2302.13473*.

[3] M. Hu et al., "FedCross: Towards accurate federated learning via multi-model cross-aggregation," in *Proc. IEEE Int. Conf. Data Eng. (ICDE)*, 2024, pp. 2137–2150.

[4] K. Mo, C. Chen, J. Li, H. Xu, and C. J. Xue, "Two-dimensional learning rate decay: Towards accurate federated learning with non-IID data," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2021, pp. 1–7.

[5] M. Hu et al., "FedMut: Generalized federated learning via stochastic mutation," in *Proc. AAAI Conf. Artif. Intell. (AAAI)*, 2024, pp. 12528–12537.

[6] D. Yan et al., "Have your cake and eat it too: Toward efficient and accurate split federated learning," 2023, *arXiv:2311.13163*.

[7] X. Zhang, M. Hu, J. Xia, T. Wei, M. Chen, and S. Hu, "Efficient federated learning for cloud-based AIoT applications," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. (TCAD)*, vol. 40, no. 11, pp. 2211–2223, Nov. 2021.

[8] C. Hao et al., "FPGA/DNN co-design: An efficient design methodology for IoT intelligence on the edge," in *Proc. Design Autom. Conf. (DAC)*, 2019, pp. 1–6.

[9] M. Hu, E. Cao, H. Huang, M. Zhang, X. Chen, and M. Chen, "AIoTML: A unified modeling language for AIoT-based cyber-physical systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. (TCAD)*, vol. 42, no. 11, pp. 3545–3558, Nov. 2023.

[10] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-edge AI: Intelligentizing mobile edge computing, caching and communication by federated learning," *IEEE Netw.*, vol. 33, no. 5, pp. 156–165, Sep./Oct. 2019.

[11] A. Li, J. Sun, P. Li, Y. Pu, H. Li, and Y. Chen, "Hermes: An efficient federated learning framework for heterogeneous mobile clients," in *Proc. Int. Conf. Mobile Comput. Netw. (MobiCom)*, 2021, pp. 420–437.

[12] L. Li, H. Xiong, Z. Guo, J. Wang, and C.-Z. Xu, "SmartPC: Hierarchical pace control in real-time federated learning system," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, 2019, pp. 406–418.

[13] M. Hu et al., "GitFL: Uncertainty-aware real-time asynchronous federated learning using version control," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, 2023, pp. 145–157.

[14] Y. Wu et al., "Federated contrastive learning for dermatological disease diagnosis via on-device learning," in *Proc. Int. Conf. Comput. Aided Design (ICCAD)*, 2021, pp. 1–7.

[15] D. C. Nguyen et al., "Federated learning for smart healthcare: A survey," *ACM Comput. Surv.*, vol. 55, no. 3, pp. 1–37, 2022.

[16] A. K. Singh, K. R. Basireddy, A. Prakash, G. V. Merrett, and B. M. Al-Hashimi, "Collaborative adaptation for energy-efficient heterogeneous mobile SoCs," *IEEE Trans. Comput. (TC)*, vol. 69, no. 2, pp. 185–197, Feb. 2020.

[17] Y. J. Cho, A. Manoel, G. Joshi, R. Sim, and D. Dimitriadis, "Heterogeneous ensemble knowledge transfer for training large models in federated learning," in *Proc. Int. Joint Conf. Artif. Intell. (IJCAI)*, 2022, pp. 2881–2887.

[18] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *Int. J. Comput. Vis.*, vol. 129, no. 6, pp. 1789–1819, 2021.

[19] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*.

[20] W. Park, D. Kim, Y. Lu, and M. Cho, "Relational knowledge distillation," in *Proc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 3967–3976.

[21] E. Diao, J. Ding, and V. Tarokh, "HeteroFL: Computation and communication efficient federated learning for heterogeneous clients," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2021, pp. 1–24.

[22] M. Kim, S. Yu, S. Kim, and S.-M. Moon, "DepthFL: Depthwise federated learning for heterogeneous clients," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2022, pp. 1–20.

[23] C. Jia et al., "AdaptiveFL: Adaptive heterogeneous federated learning for resource-constrained AIoT systems," in *Proc. Design Autom. Conf. (DAC)*, 2024, pp. 1–6.

[24] F. Ilhan, G. Su, and L. Liu, "ScaleFL: Resource-adaptive federated learning with heterogeneous clients," in *Proc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2023, pp. 24532–24541.

[25] M. Hu, W. Duan, M. Zhang, T. Wei, and M. Chen, "Quantitative timing analysis for cyber-physical systems using uncertainty-aware scenario-based specifications," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. (TCAD)*, vol. 39, no. 11, pp. 4006–4017, Nov. 2020.

[26] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," 2016, *arXiv:1607.03250*.

[27] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Annu. Conf. Neural Inf. Process. Syst. (NIPS)*, 2015, pp. 1135–1143.

[28] M. C. Mozer and P. Smolensky, "Skeletonization: A technique for trimming the fat from a network via relevance assessment," in *Proc. Annu. Conf. Neural Inf. Process. Syst. (NIPS)*, 1988, pp. 107–115.

[29] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *Proc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 4340–4349.

[30] R. Liu et al., "AdapterFL: Adaptive heterogeneous federated learning for resource-constrained mobile computing systems," 2023, *arXiv:2311.14037*.

[31] S. Horváth, S. Laskaridis, M. Almeida, I. Leontiadis, S. I. Venieris, and N. D. Lane, "FjORD: Fair and accurate federated learning under heterogeneous targets with ordered dropout," in *Proc. Annu. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2021, pp. 12876–12889.

[32] R. Liu et al., "No one left behind: Inclusive federated learning over heterogeneous devices," in *Proc. Conf. Knowl. Discov. Data Min. (KDD)*, 2022, pp. 3398–3406.

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 770–778.

[34] M. Sandler et al., "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018, pp. 4510–4520.

[35] M. Phuong and C. Lampert, "Towards understanding knowledge distillation," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2019, pp. 5142–5151.

[36] A. Krizhevsky, "Learning multiple layers of features from tiny images," Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, Rep. TR-2009, 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:18268744

[37] Y. Le and X. Yang. "Tiny ImageNet visual recognition challenge." 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:16664790

[38] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2014, pp. 1–14.

[39] S. Caldas et al., "LEAF: A benchmark for federated settings," 2018, *arXiv:1812.01097*.

[40] S. Alam et al., "FedAIoT: A federated learning benchmark for artificial intelligence of things," 2023, *arXiv:2310.00109*.