FedAD: An Adaptive Block Dropping Approach for Federated Learning on Resource-Constrained Devices

Qiqi Cai, Jian Cao, *Senior Member, IEEE,* Jianqing Zhang, Wei Guan,
Shiyou Qian, *Member, IEEE*, Rajkumar Buyya, *Fellow, IEEE*

***Abstract*—In a federated learning environment with heterogeneous, limited and time-varying computational resource capabilities, it is challenging to update deep neural network in real-time. To address this challenge, we propose FedAD, an adaptive block-dropping approach for federated learning on resource-constrained devices. FedAD dynamically adjusts the size of the local training model in real-time by dropping blocks based on the device's varying resource availability. This is controlled by a policy network, which determines the drop probability for each block depending on data inputs. Using the policy network's output, FedAD generates a drop policy that decides which blocks in the neural network are dropped during local training. Applying this method, each device can independently generate the drop policy that dynamically adapts to the input and real-time available resources. Extensive experiments demonstrate that FedAD converges faster and outperforms state-of-the-art methods in resource-constrained settings, performing on par with or better (between 10% to 50%) than baselines in resource-rich environments. Additionally, FedAD demonstrates strong adaptability to real-time resource fluctuations and high flexibility in accommodating heterogeneous resources.**

***Index Terms*—Federated learning, device heterogeneity, resource-constrained, real-time system, conditional computation**

I. Introduction

Federated Learning (FL) [1] enables distributed clients with private data to collaboratively train models under the coordination of a central server, addressing privacy concerns and regulatory requirements such as GDPR [2] and CCPA [3]. However, training deep neural networks (NN) locally is resource-intensive, and device computational capabilities in FL systems are often heterogeneous [4], posing significant challenges for real-world deployment [5].

Existing methods to address computational bottlenecks include adjusting training configurations such as tolerating longer learning delays [6], adapting the number of local training iterations [7], using variable batch sizes [8], employing asynchronous aggregation [9], and optimizing device selection through excluding stragglers [10] or intelligent scheduling such as AutoFL [11]. Other approaches reduce local computational burden through external assistance, like AnytimeFL, [12] it leverages early-exit mechanisms and additional adapters for pre-trained Transformers. However, these approaches often sacrifice model performance, require additional server-side computation, or are restricted to specific model architectures. In contrast, **partial training-based model-heterogeneous FL** offers a more flexible solution by extracting sub-networks from the global model for local training, directly adapting model complexity to each device's available resources without external dependencies. These approaches can be classified into three types: *width scaling*, *depth scaling*, and *joint width-depth scaling*. Width scaling-based methods (e.g. HeteroFL [13], FjORD [4]) adjust model size by scaling convolutional layer channels but introduce channel mismatches during aggregation. Depth scaling-based methods (e.g., DepthFL [14], FedSplitX [15]) allow clients to train networks of varying depths, showing better performance. Joint width-depth scaling methods (e.g., ScaleFL [16], NeFL [17]) simultaneously reduce both width and depth to construct nested sub-networks.

However, existing depth scaling-based methods still face several challenges. **(1) Fixed submodel size limits global performance**: Most methods extract submodels by dropping or freezing deep blocks, preventing deeper layers from being trained on resource-constrained devices [14]. **(2) Submodel extraction strategies are complex and inefficient**: While some approaches enhance performance through block selection or progressive training, their submodel extraction strategies are heuristic, lacking efficiency and with limited applicability [18]. **(3) Significant additional computational and communication overhead**: Some methods introduce extra overhead through self-distillation or offloading training tasks to the server [15]. **(4) Neglecting time-varying resource availability**: Most methods assume stable device resources, making them unsuitable for real-time systems where resources fluctuate due to competing tasks [19]. For instance, Google GBoard [20] trains models only when devices are charging and idle; once conditions change, training halts, resulting in accuracy reduction [21].

To address these challenges, we propose FedAD, an **AD**aptive block dropping approach for **Fed**erated learning on resource-constrained devices. It dynamically adapts to both input samples and changing device computational resources, ensuring resource awareness and efficient resource utilization during training, and maintaining model performance in real-

Qiqi Cai, Jian Cao, Jianqing Zhang, Wei Guan and Shiyou Qian are with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China, 200240. Jian Cao is also with the Shanghai Key Laboratory of Trusted Data Circulation and Governance and Web3, Shanghai, China, 200240. Jian Cao is the corresponding author. Rajkumar Buyya and also Qiqi Cai are with the Quantum Cloud Computing and Distributed Systems (qCLOUDS) Lab, School of Computing and Information Systems, the University of Melbourne, Australia. E-mail: cai_qiqi, cao-jian, tsingz, guan-wei, qshiyou@sjtu.edu.cn, rbuyya@unimelb.edu.au.

time system. Specifically, we focus on modular neural networks which are composed of multiple similar blocks stacked together for image classification (referred to as the classifier), such as ResNet [22], MobileNet [23], etc.. In FedAD, a policy network based on conditional computation [24] is introduced to determine the drop probability for each block based on the data input. Using the output of the policy network, along with time constraints and device's computational capabilities, FedAD generates a drop policy that determines which blocks in the classifier are dropped during local training. The policy network and classifier are jointly trained with the goal of minimizing the loss. Furthermore, to minimize additional resource consumption, the introduced policy network is designed to be lightweight, details of which will be discussed in section V.

The main contributions of our work are as follows:

- We address the problem of resource availability in real-time systems, considering devices with heterogeneous and time-varying resources, which has received relatively little research attention.
- We propose FedAD, a method that combines conditional computation with federated learning to dynamically adjust model complexity during training. FedAD generates input-specific drop policies in real-time, adapting flexibly to heterogeneous and fluctuating device resources.
- We conduct extensive experiments across various datasets and modular neural networks to validate the effectiveness of FedAD.

The rest of the paper is organized as follows. The related work is presented in section II. Section III presents the preliminaries and formalizes the problem. In section IV, the proposed framework FedAD is illustrated. Experimental results and analysis are reported in section V. And section VI concludes the paper with future directions.

The rest of this paper is organized as follows. Section II reviews the related work. Section III presents the preliminaries and formalizes the problem. Section IV describes the proposed FedAD framework in detail. Section V reports the experimental results and analysis. Finally, Section VI concludes the paper and outlines directions for future work.

II. Related Work

In this section, we briefly review the literature related to our research, covering two domains: partial training-based model-heterogeneity federated learning and conditional computation.

A. Partial Training-based Model-Heterogeneous FL

Federated learning methods aware of computational resources include full neural network training and partial training-based approaches. Our work belongs to the latter, which extracts sub-networks from the global model to match device resource availability [21]. Existing methods can be categorized into width scaling and depth scaling [14].

Width scaling-based methods reduce model size by narrowing network width. HeteroFL [13] and FjORD [4] create hierarchical structures where smaller sub-networks are contained within larger ones, but this limits resource granularity and leaves parts of the global model untrained on resource-constrained devices, affecting overall performance. To address these issues, randomized sub-network selection methods have been introduced. Federated Dropout [5] applies server-side dropout, sending smaller networks to devices for training, FLFISH [25] assigns customized dropout masks based on device resources, and DISTREAL [21] adjusts computational complexity by randomly dropping convolutional filters without a fixed layer subset ratio. However, random sub-model selection struggles with highly heterogeneous data environments with small client populations. To mitigate this, FedRolex [26] uses a rolling window approach to improve sub-model extraction and balance random and static selection issues.

Despite progress, width scaling-based methods suffer from parameter mismatches during aggregation, leading to lower accuracy [14]. **Depth scaling-based** methods address this issue. CoCoFL [18] reduces communication and computation demands by freezing and quantizing selected layers while keeping other layers in full precision, but requires additional computation and time-consuming heuristic searches. In contrast, FedSplitX [15] and FedAdapt [27] split models into client-side and server-side components to alleviate the computational burden on the device, but uploading smashed data increases privacy risks and communication costs. DepthFL [14] prunes deeper layers based on client resources but uses fixed sub-model structures, which can affect the global model's performance. Although the authors alleviate this by using mutual self-distillation, it complicates the training process. To mitigate these issues, Wu et al. introduced ProFL [28], a progressive block training based method, where only part of the model is trained in each round. However, the block-freezing strategy is complex and introduces significant computational overhead.

Joint width-depth scaling-based methods simultaneously adjust both network width and depth. ScaleFL [16] constructs nested sub-networks by progressively pruning both channels and layers with self-distillation. NeFL [17] generates sub-networks via joint width-depth scaling inspired by ODE solvers, allowing each client to dynamically select a sub-network based on its resource constraints. However, both methods assign sub-networks solely based on device resource levels, without adapting to the complexity of individual input samples.

Additionally, many existing partial-training methods assume constant resource availability, making them unsuitable for real-time systems where device resources vary over time. Motivated by these challenges, we propose FedAD, which introduces a lightweight policy network that dynamically generates sub-networks tailored to both each client's real-time computational constraints and the difficulty of each input sample. This approach minimizes additional computation and communication overhead while maintaining the global model's performance.

B. Conditional Computation

Conditional computing is a method that adds dynamic behavior to models by adapting them based on task-specific conditional inputs [24]. Formally, given a conditional input C and an auxiliary module $AM(\cdot;\theta)$, a signal S can be generated

as $S = AM(C;\theta)$. This signal can then be used to modulate the model, facilitating processes such as dynamic routing or feature adaptation based on the conditional input.

Several methods for conditional computation have been proposed to dynamically activate different modules of a network model based on specific inputs. To reduce computational overhead, Wang et al. introduce SkipNet [29], a modified residual network that employs a gating mechanism to selectively bypass convolutional blocks depending on the activations of the preceding layer. SpotTune [30] further extends this idea by using a policy network to make per-image decisions, determining which blocks in a pre-trained residual network should be fine-tuned. For inference efficiency, ConvNet-AIG [31] uses Gumbel Softmax [32] to activate layers dynamically based on input image complexity. Recognizing that varying image complexities demand different network depths, BlockDrop [33] learns layer selection to optimize computation efficiency without compromising accuracy. To further accelerate inference and reduce computational costs, BranchyNet [34] introduces a novel architecture with additional side branch classifiers which enable early exits, effectively speeding up inference for simpler cases.

In the realm of federated learning, FedCP [35] generates conditional policies for each sample to separate global and personalized features to enhance personalization. Similarly, Yin et al. introduced FedCLCC [36], which employs conditional computing to separate features, integrating it with contrastive learning to achieve higher personalization. While these approaches focus on personalization, our work leverages conditional computation to address the challenge of heterogeneous device resources in FL.

III. Preliminaries

Before formulate the problem we solve, we provide a definition of the modular neural network.

Modular Neural Network: Consider a neural network $\mathcal{N}_\theta$ composed of an input layer $\mathbf{I}$, an output layer $\mathbf{O}$, and B blocks $\mathcal{B}$. The blocks are composable functional modules, and can be dropped from the network. We denote the complete network as

$$\mathcal{N}_\theta = \mathbf{O}_{\theta_\mathbf{O}} \circ \mathcal{B}_{\theta_B} \circ \mathcal{B}_{\theta_{B-1}} \circ \cdots \circ \mathcal{B}_{\theta_1} \circ \mathbf{I}_{\theta_\mathbf{I}} \tag{1}$$

where $\mathcal{B}_{\theta_i}$ represents the i-th block with parameters θ_i. Given a routing $\mathbf{a} = [a_1, a_2, \ldots, a_B]$ with $a_i \in \{0,1\}$ (a binary vector where $a_i = 1$ indicates the block is retained and $a_i = 0$ indicates it is dropped) and it corresponding to block $\mathcal{B}_i$, a sub-network $\mathcal{N}_{\theta'}$ is obtained. Let $S = \{i_1, \ldots, i_k\}$ $(i_1 < \cdots < i_k)$ denotes the indices of the kept blocks, the sub-network with parameters $\theta' = \{\theta_\mathbf{I}, \theta_\mathbf{O}, \theta_i | i \in S\}$ is defined as

$$\mathcal{N}_{\theta'} = \mathbf{O}_{\theta_\mathbf{O}} \circ \mathcal{B}_{\theta_{i_k}} \circ \mathcal{B}_{\theta_{i_{k-1}}} \circ \cdots \circ \mathcal{B}_{\theta_{i_1}} \circ \mathbf{I}_{\theta_\mathbf{I}} \tag{2}$$

where $i_1 < i_2 < \cdots < i_k$. An example is given by Fig. 1.

Problem Statement: Suppose we have N clients with their private training data $D = \{D_1, D_2, \ldots, D_N\}$ and device computational resource constraints $R = \{r_1, r_2, \ldots, r_N\}$. The main object of our problem is to minimize the global loss under computation constraint, by generating the sub-network for each device per round per input:

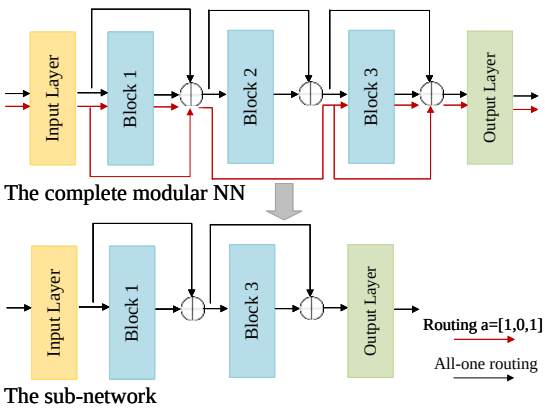


Fig. 1: A toy of modular neural network and sub-network.

$$\begin{aligned} \min_{\Theta,\tilde{\Theta}} \quad & \mathcal{L}_{global}(\Theta,\tilde{\Theta}) \triangleq \sum_{n=1}^{N} k_n \mathcal{L}_n(\Theta_n,\tilde{\Theta}_n) \\ s.t. \quad & t_n(\Theta_n) \le T, \quad \forall n \in N \end{aligned} \tag{3}$$

with

$$\mathcal{L}_n(\Theta_n,\tilde{\Theta}_n) \triangleq \frac{1}{|D_n|} \sum_{i=1}^{|D_n|} \mathcal{L}(\Theta_n,\tilde{\Theta}_n;\mathbf{x}_i,y_i,r_n) \tag{4}$$

where $\mathcal{L}(\cdot)$ is the loss function, $k_n = |D_n| / \sum_{j=1}^{N} |D_j|$, and $|D_n|$ and $|D_j|$ are the amount of local data samples on client n and j. $t_n(\cdot)$ is client n's training time per round, and T is the pre-defined round time. Θ and Θ_n are global and local model parameters, which are both the complete NN described as formula (1). Differently, the client trains only a sub-network of Θ_n defined by formula (2) being dependent on the policy network $\tilde{\Theta}_n$ and the device computation resource constraint r_n. Note that resource constraints may change over time, and input samples are different, causing dynamic sub-network changes. For simplicity, we represent them with the same notation here.

IV. Methodology

In this section, we provide a detailed overview of FedAD. FedAD dynamically generates a drop policy (a binary vector as defined routing $\mathbf{a} \in \{0,1\}^B$ in Section III)) to construct sub-networks during training based on the data input and computational constraints of devices, using a policy network. This process will be explained in section IV-A. Following that, we will introduce the training process of the policy network and the overall framework of FedAD. Finally, we will discuss the effectiveness of FedAD in saving resources.

A. Resource-Aware Sub-network Extraction

In FedAD, both the global model and local models are complete modular NN, while clients train sub-networks of their local models. These sub-networks are dynamically generated based on the input, the round time and computational resource constraints. The policy network predicts each block's drop probability, while constraints determine how many blocks to remove, together defining the drop policy, which generates

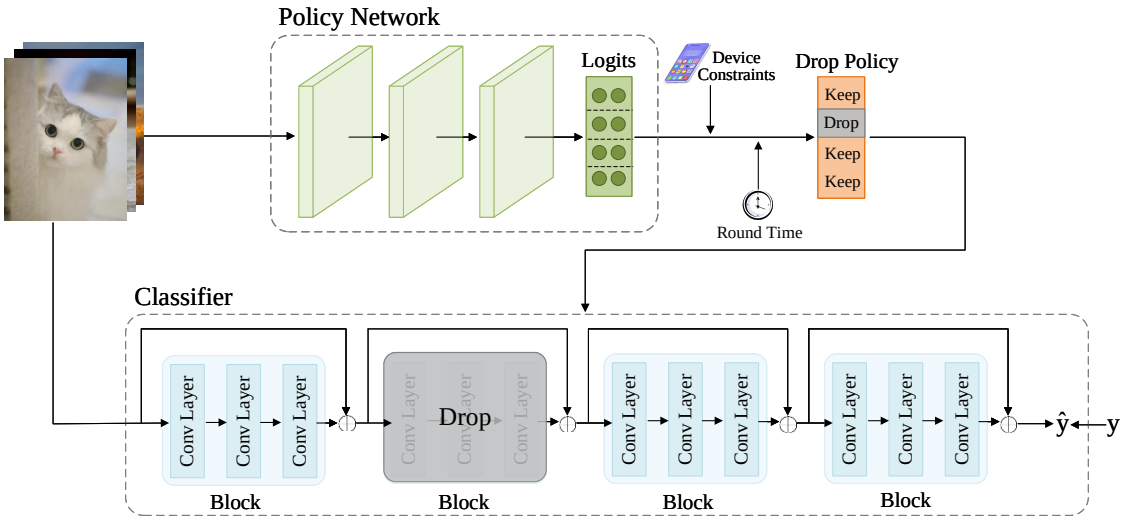


Fig. 2: Illustration of our proposed resource-aware local model generation process.

sub-networks. This process is illustrated in Fig. 2. Note that only blocks with identical input and output dimensions are droppable. Dimension-changing blocks and the input/output layers form a fixed backbone, ensuring that a valid gradient flow path always exists regardless of the drop policy.

Resource Quantification. To generate the drop policy, we need to quantify the computational resource. We use FLOPS [37] to measure the computing power r_n of client n, and MACs [21] to measure the computational demand of a model [38]. For global model Θ consisting of basic modules (such as the input and output layers) and droppable blocks $\mathcal{B}$, the computational demand is

$$C_\Theta = c_{basic} + c_{\mathcal{B}_{\theta_1}} + \cdots + c_{\mathcal{B}_{\theta_B}} \tag{5}$$

where c_{basic} is the MACs of basic modules, and $c_{\mathcal{B}_{\theta_i}}$ represents the MACs of the block $\mathcal{B}_i$. To simplify, we denote the MACs of blocks $\mathcal{B}$ as the vector $\mathbf{c} = [c_{\mathcal{B}_{\theta_1}}, \ldots, c_{\mathcal{B}_{\theta_B}}]$.

For client n with device computing power constraint r_n, before each round, $\bar{C}_n$ is the maximum MACs that can be supported for model training in each batch (called upper bound MACs) is calculated, which is the average of the maximum supported MACs per round across the batches. According to [39], the local training time per round is given by

$$t_n = \frac{2 \times C_{\Theta_n} \times |D_n| \times E}{r_n} \times 2 \tag{6}$$

where Θ_n represents the computational demand of local model Θ_n and E is the number of epochs. The first multiplication by 2 account for $1MACs = 2FLOPS$ [40], and the second one because a training step (forward pass, backpropagation and weight updates) requires about $2\times$ more MACs than forward pass alone [41]. Therefore,

$$\bar{C}_n = \frac{r_n \times T \times batch\_size}{2 \times E \times |D_n|}. \tag{7}$$

Discussion on Modeling Assumptions. The FLOPS/MACs abstraction in Eqs.(5)–(7) is a first-order approximation that does not capture hardware effects such as memory bandwidth, framework overhead, and operator-level utilization, and the peak FLOPS r_n overestimates sustained throughput. However, this does not compromise FedAD's decision quality: the drop policy inequality in Eq. (10) is a MACs-domain comparison in which both the candidate sub-network cost ($\mathbf{c} \cdot \mathbf{d} + c_{\text{basic}}$) and the upper bound $\bar{C}_n$ are computed under the same peak-FLOPS convention, so any systematic utilization gap cancels out and only the relative ranking of configurations matters. The round time T further serves as a soft buffer rather than a hard deadline, absorbing residual imprecision. Empirical validation is provided in Section V-E.

Drop Policy Generation. For batch input $\mathcal{X}_n$ and modular network Θ with B blocks, the drop policy $\mathbf{d}$ is obtained through the following steps:

- *Step1:* Input $\mathcal{X}_n$ into the policy network:

$$\mathbf{S} = f_{pn}(\mathcal{X}_n, \tilde{\Theta}_n) \tag{8}$$

  where f_{pn} is the policy network parameterized by $\tilde{\Theta}_n$, and $\mathbf{S} \in \mathbb{R}^{batch\_size \times B \times 2}$ is the output tensor of the network. After averaging along the batch dimension and applying softmax [42], we get $\bar{\mathbf{S}} \in \mathbb{R}^{B \times 2}$.
- *Step2:* We employ the Gumbel-Softmax sampling strategy to acquire the probability of each block being kept and dropped. To make the probability more deterministic, i.e., to have it approach values closer to 0 or 1, we introduce an adjustment matrix $\mathbf{V} \in \mathbb{R}^{B \times 2}$ (where $V_{i1} = \sigma$, $V_{i2} = \gamma$), to amplify the difference between $\bar{S}_{i1}$ and $\bar{S}_{i2}$. The probability matrix $\mathbf{P} \in \mathbb{R}^{B \times 2}$ is calculated by:

$$\mathbf{P} = Gumbel - Softmax(\bar{\mathbf{S}} + \mathbf{V}) \tag{9}$$

  For each block $\mathbf{P}_i = [P_{i1}, P_{i2}]$ ($P_{i1}, P_{i2} \in [0, 1]$ and $P_{i1} + P_{i2} = 1$) where P_{i1} represents the likelihood of block $\mathcal{B}_i$ being kept, and P_{i2} indicates the probability of being dropped, accordingly.
- *Step3:* In *Step 2*, $\mathbf{P}$ is the logits (shown in Fig. 2) obtained based on the input samples, but determining the drop policy also requires considering the device's

Algorithm 1 Resource-Aware Sub-network Extraction

Input: Batch dataset $\mathcal{X}_n$, upper bound Macs $\bar{C}_n$, policy network parameters $\tilde{\Theta}_n$, adjustment matrix V, block MACs vector $\mathbf{c}$, MACs of basic modules c_{basic}, $batch\_size$

1: Compute probability matrix $\mathbf{P}$ using formula (8) and formula (9)
2: Initialize $\mathbf{d} = [1, \ldots, 1]$ and get $\mathbf{p} = \mathbf{P}[:, 1]$
3: Sort $\mathbf{p}$ in descending order
4: **if** $2 \times (\mathbf{c} \cdot \mathbf{d} + c_{basic}) \times batch\_size \leq \bar{C}_n$ **then**
5: **break**
6: **else**
7: **for** element p_i in $\mathbf{p}$ **do**
8: **if** $2 \times (\mathbf{c} \cdot \mathbf{d} + c_{\text{basic}}) \times batch\_size \leq \bar{C}_n$ **then**
9: **break** {Condition satisfied, stop dropping blocks}
10: **end if**
11: $idx \leftarrow \text{GetIdx}(p_i)$
12: $d_{idx} \leftarrow 0$ {Drop block $\mathcal{B}_{idx}$}
13: **end for**
14: **end if**
15: $\mathbf{d} = (\mathbf{d} - \mathbf{p}).detach() + \mathbf{p}$ {Ensure the entire network structure is continuously differentiable.}
16: Input $\mathbf{d}$ into the classifier

computing power and round time constraints. Therefore, we binarize $\mathbf{P}$ (generating the drop policy $\mathbf{d}$) in the following way. i) Initialize $\mathbf{d}$ as an all-ones vector. ii) Sort drop probabilities $\mathbf{p} = \mathbf{P}[:, 1]$ in descending order. iii) Check whether inequality (10) is satisfied. If it does, the device is capable of training the complete global model. Otherwise, iteratively set $d_j = 0$ for blocks with highest drop probability until:

$$2 \times (\mathbf{c} \cdot \mathbf{d} + c_{basic}) \times batch\_size \leq \bar{C}_n \tag{10}$$

where $\cdot$ is the dot product operation. In this way, we can retain as many blocks as possible under constraints to ensure the performance of the local model [43]. In $\mathbf{d}$, 1 indicates keep the corresponding block, and 0 means drop.

Finally, we can obtain the sub-network for the current batch based on drop policy $\mathbf{d}$. The obtained sub-network through our method not only meets the resource constraints but also ensures model performance. The per-batch process for client n is shown in Algorithm 1.

B. Policy Network Training

As shown in Fig. 2, the drop policy $\mathbf{d}$ is input into the classifier, linking the policy network and classifier for joint training. To maintain differentiability, we apply the $detach()$ operation [44] (line 15 of Algorithm 1) to link $\mathbf{d}$ and $\mathbf{P}$. The $detach()$ method separates the gradient of $\mathbf{d}$ from the computation graph, ensuring this part does not affect gradient calculations during backpropagation. In this way, the model focuses on the drop policy $\mathbf{d}$ during the forward pass, while during backpropagation, it concentrates on the continuous variable $\mathbf{p}$.

Therefore, by using the *detach()* operation, and sampling the probability matrix from a Gumbel Softmax distribution, we can backpropagate through the discrete samples to the policy network. By employing the cross entropy loss [45] L_c for the multi-class classification task (formula 11), the policy network is jointly trained with the classifier to identify the optimal drop strategy that minimizes the loss while adhering to device constraints.

$$L_c = -\sum_{i=1}^{C} y_i log(\hat{y}_i) \tag{11}$$

where C is the number of category, y_i is a one-hot encoding vector of the truth label, and $\hat{y}_i$ is the predicted probability belonging to category i.

C. FedAD Framework

FedAD follows standard FL with N clients and a server, and each training round is completed by four steps: i) broadcast global model; ii) local update; iii) upload local models; iv) server aggregation. However, there are some key differences between FedAD and traditional FL need to be noted. i) in FedAD, we introduce a policy network which is trained together with the classifier, so we bundle it and classifier for model transmission, training, and aggregation. ii) in FedAD, the local training might involve only a sub-network is trained, and routing varies between batches, dynamically adjusting sub-networks. It is important to note that dropped blocks are just excluded from current training, not removed from the local model entirely. iii) the aggregation method in FedAD varies depending on the situation.

Aggregation. We denote the parameters set of local model for client n as $\Theta_n = \{_{basic}\theta_n, _1\theta_n, \ldots, _B\theta_n\}$. Additionally, client n uploads a binary signal vector $\mathbf{u}_n$ of length B after each training round, where $u_n^i = 1$ indicates block $\mathcal{B}_i$ is updated by client n, and $u_n^i = 0$ indicates it has not. Policy network and basic blocks are updated by all selected clients, and we use the method of FedAvg [46] for aggregation:

$$\tilde{\Theta}^t = \sum_{n \in S} \frac{|D_n|}{\sum_{j \in S} |D_j|} \tilde{\Theta}_n^t \tag{12}$$

$$_{basic}\theta^t = \sum_{n \in S} \frac{|D_n|}{\sum_{j \in S} |D_j|} {}_{basic}\theta_n{}^t \tag{13}$$

For blocks $\mathcal{B}_i$ in classifier, if the block $\mathcal{B}_i$ is not updated by any client, its parameters in the global from the previous round are retained:

$$_i\theta^t = _i\theta^{t-1} \tag{14}$$

Otherwise, we only aggregate the parameters from the clients that have updated them:

$$_i\theta^t = \sum_{n \in S \bigwedge u_n^i = 1} \frac{|D_n|}{\sum_{j \in S \bigwedge u_j^i = 1} |D_j|} {}_i\theta_n{}^t \tag{15}$$

where t represents the round number. The pseudocode of FedAD is in Algorithm 2.

Remark on the sufficiency of partial data-proportional aggregation. FedAD's aggregation strategy is a block-wise partial averaging scheme: for each droppable block $\mathcal{B}_i$, only

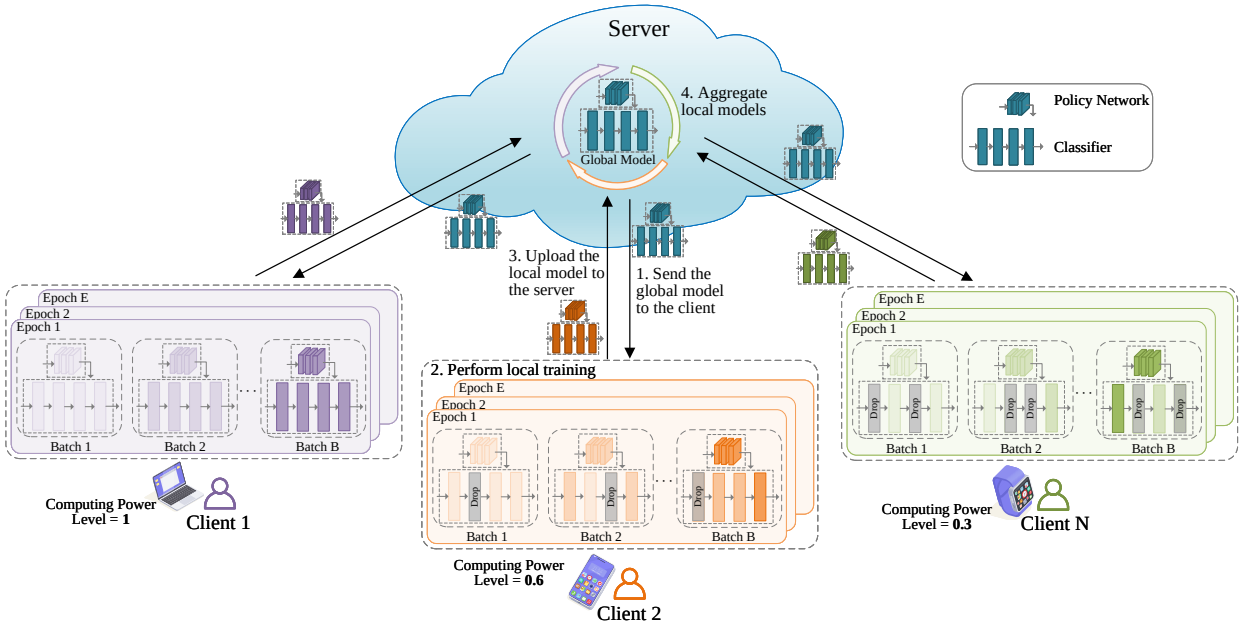


Fig. 3: The framework of FedAD. Three clients have different types of devices with varying computational capability. A higher computing power level indicates more available resources. During local training, the darker the color of the blocks, the more frequently they have been trained.

Algorithm 2 FedAD (proposed approach)

Input: N clients, global round number T,

$\Theta^0, \tilde{\Theta}^0 \leftarrow$ Initialize the global model and the global policy network
for each round $t = 0, 1, \ldots, T$ **do**
  $S_t \leftarrow$ Samples a subset of clients
  Broadcast Θ^t and $\tilde{\Theta}^t$ to clients $n \in S_t$
  for each client $n \in S_t$ in parallel **do**
    $\Theta_n^t, \tilde{\Theta}_n^t, \mathbf{u}_n^t \leftarrow$ Perform ClientUpdate and Receive updated local model, local policy network and signal vector {Call Algorithm 3}
  end for
  $\Theta^{t+1}, \tilde{\Theta}^{t+1} \leftarrow$ Aggregate received models according to formula (12)-(15)
end for

Algorithm 3 ClientUpdate

Input: Local dataset D_n, training epochs E, *batch_size*, learning rate for the classifier η_1 and policy network η_2

$\Theta^t, \tilde{\Theta}^t \leftarrow$ Receive the global model and the global policy network
$H_n \leftarrow$ Split local dataset D_n into batches of *batch_size*
for each epoch e from 1 to E **do**
  $\bar{C}_n \leftarrow$ Calculate upper bound MACs according to formula (7)
  for batch $h_n \in H_t$ **do**
    $\Theta_n^t = \Theta_n^t - \eta_1 \nabla_{\Theta_n^t} \mathcal{L}(\Theta_n^t, \tilde{\Theta}_n^t; h_n, \bar{C}_n)$;
    $\tilde{\Theta}_n^t = \tilde{\Theta}_n^t - \eta_2 \nabla_{\tilde{\Theta}_n^t} \mathcal{L}(\Theta_n^t, \tilde{\Theta}_n^t; h_n, \bar{C}_n)$
  end for
end for
$\mathbf{u_n^t} \leftarrow$ Generate signal vector
Upload Θ_n^t, $\tilde{\Theta}_n^t$, and $\mathbf{u_n^t}$ to the server

clients that trained $\mathcal{B}_i$ contribute to its update, with data-proportional weights. This simple design is sufficient for two reasons. First, FedAD's depth-wise block dropping preserves each retained block's original parameter structure, so the shape-misalignment issue typical of width-scaling methods does not arise. Second, the policy network (Section IV-B) is trained end-to-end to retain only the blocks most critical for each input under the resource budget, so per-block updates reaching the server are already aligned with a shared objective. A natural alternative is to replace the binary indicator u_n^i with a continuous frequency signal $\tilde{u}_n^i \in [0, 1]$ and weight aggregation as $w_n^i \propto |D_n| \cdot \tilde{u}_n^i$. We do not adopt this scheme because $\tilde{u}_n^i$ is governed primarily by a client's resource budget rather than by update informativeness; down-weighting low-frequency clients penalizes compute scarcity rather than gradient quality and compounds the disadvantage of resource-constrained clients that already train smaller sub-networks. FedAD therefore retains the simpler data-proportional weighting in equation 15, which is theoretically consistent with Corollary IV.10 and, as we verified empirically, sufficient.

D. Resources Savings Discussion

The main goal of our method FedAD is to alleviate the computational demands during training to reduce training time and mitigate the impact of stragglers on the system. When a block in the network is dropped, it does not need to perform forward and backward propagation and gradient updates during training. Consequently, FedAD performs well in reducing the MACs count. For instance, dropping a standard residual block

TABLE I: MMACs for one sample and the parameter amounts (M) for models with different drop rates. The drop rate refers to the ratio of the number of dropped blocks to the total number of blocks. Since the MACs and the number of parameters of each block is not exactly the same, the value shown here is an average.

Drop rate	Computation (MMACs)				Memory (M)			
	C100/RN18	EM/RN34	Tiny/RN34	EM/RN50	C100/RN18	EM/RN34	Tiny/RN34	EM/RN50
0.0	37.24	69.86	300.11	78.59	11.23	21.31	21.39	23.63
0.2	32.96	57.00	246.03	65.36	9.86	17.35	17.43	19.74
0.4	24.39	44.11	191.96	52.14	7.11	13.40	13.48	15.86
0.6	20.10	31.22	137.89	38.91	5.73	9.44	9.52	11.97
0.8	11.53	18.33	83.81	25.69	2.99	5.49	5.57	8.08

drop rate = # dropped blocks / # total blocks

[22] can reduce the MACs by $2\times K_{\text{in}}\times K_{\text{out}}\times 9\times H\times W$, where K_{in} and K_{out} are channels of input and output respectively, $H\times W$ is the size of the input feature. Taking ResNet18 [22] as an example, dropping a residual block on average reduces the MACs of the full model by 11.5%. Similarly, dropping a bottleneck residual block [22] in ResNet50 [22] can reduce the model's MACs by approximately 6.19% of the original. At the same time, the number of parameters in the model decreases as the model size reduces, which also reduces the memory requirements. We present in Table I the needs of MMACs ($1MMACs = 10^6 MACs$) when training a single sample and the parameter amounts of different datasets on some modular neural networks after dropping blocks.

While block dropping reduces computational demands, the policy network that enables this adaptation introduces its own overhead. As shown in the "Proportion" column of Table II, the computational cost of the policy network is minimal relative to the classifier because it is lightweight. Across all experimental configurations, the policy network accounts for only 4.64%–12.97% of the classifier's MACs. This lightweight design ensures that the adaptive decision-making cost is negligible compared to the computational savings achieved through block dropping.

Beyond client-side computation, FedAD also introduces additional aggregation logic on the server. While FedAD introduces additional aggregation logic compared to FedAvg, the server-side overhead remains negligible in practice. The binary signal vector uploaded by each client contains only B bits (e.g., 16 bits for ResNet34), which is negligible compared to transmitting millions of model parameters. The aggregation complexity increases from $O(|S|\times P)$ in FedAvg to $O(|S|\times B)+O(|S|\times P)$ in FedAD, where $|S|$ is the number of selected clients, P is the number of parameters, and B is the number of droppable blocks. Crucially, $B \ll P$ always holds regardless of model scale, as each block contains numerous parameters. Even in an extreme scenario with 10,000 clients and LLaMA-7B ($B = 32$, $P \approx 7\times 10^9$), the ratio of additional overhead is merely $320,000/7\times 10^{13} \approx 4.6\times 10^{-9}$. Moreover, the block-wise aggregation is embarrassingly parallel across both clients and blocks, with practical limits constrained by communication rather than server computation. Given that modern servers possess substantially greater computational resources than edge devices, this marginal overhead is well within practical capacity.

E. Theoretical Analysis

In this section, we provide theoretical analysis of FedAD, including convergence guarantees, training stability under dynamic partial model updates, and reliability of the training mechanism.

1) Convergence Analysis: Before the proof, we make the following assumptions first.

Assumption 1 (Standard Assumptions for Federated Optimization)**.** *We adopt standard assumptions from federated optimization literature:*

1) ***L-smooth loss:*** $\|\nabla L(\Theta_1) - \nabla L(\Theta_2)\| \le L\|\Theta_1 - \Theta_2\|$ *for some constant* $L > 0$*;*
2) ***μ-strong convexity:*** $L(\Theta_1) \ge L(\Theta_2) + \langle \nabla L(\Theta_2), \Theta_1 - \Theta_2\rangle + \frac{\mu}{2}\|\Theta_1 - \Theta_2\|^2$ *for some constant* $\mu > 0$*;*
3) ***Bounded gradients:*** $\|\nabla L_n(\Theta)\| \le G$ *for some constant* $G > 0$*, where* G *is the uniform upper bound on the gradient norm across all clients;*
4) ***Bounded data heterogeneity:*** $\mathbb{E}_n\|\nabla L_n(\Theta) - \nabla L(\Theta)\|^2 \le \sigma^2$ *for some constant* $\sigma^2 \ge 0$*, quantifying the variance due to non-i.i.d. data distribution.*

Assumption 2 (Minimum Retention Probability)**.** *Each block* B_i *has minimum expected retention probability* $p_i \ge p_{\min} > 0$ *across clients, ensured by our resour-aware sub-network extraction method (Algorithm 1).*

Theorem IV.1 (Convergence of FedAD)**.** *Under Assumptions 1 and 2 with learning rate* $\eta \le \frac{\mu}{4L^2}$ *and participation rate* ρ *per round, after* T *rounds:*

$$\mathbb{E}\left[\left\|\Theta^T - \Theta^*\right\|^2\right] \le \left(1 - \frac{\mu\eta p_{\min}}{2}\right)^T \left\|\Theta^0 - \Theta^*\right\|^2 + \frac{4\eta^2 L^2 G^2}{\mu^2 \rho p_{\min}} + \frac{2\eta\sigma^2}{\mu\rho p_{\min}} \tag{16}$$

Proof Sketch. Let S_t denote the set of selected clients at round t, and let K be the number of local mini-batches per round. For each mini-batch $b \in \{1,\dots,K\}$, the policy network determines the set of retained block indices $\mathcal{S}_n^{t,b} = \{i : d_i^{n,t,b} = 1\}$ for client n, where $d_i^{n,t,b} \in \{0,1\}$ is the binary decision variable indicating whether block i is retained. All samples within the same mini-batch share the same block selection.

The local update for client n over K mini-batches yields:

$$\Theta_n^{t+1} = \Theta_n^t - \eta \sum_{b=1}^{K} \nabla_{\mathcal{S}_n^{t,b}} L_n^{(b)}(\Theta_n^{t,b}), \tag{17}$$

where $\nabla_{\mathcal{S}_n^{t,b}} L_n^{(b)}(\cdot)$ denotes the gradient computed only over retained blocks for mini-batch b, and $\Theta_n^{t,b}$ is the intermediate model after $b-1$ local updates.

By L-smoothness and weighted aggregation $\Theta^{t+1} = \sum_{n\in S_t} w_n^t \Theta_n^{t+1}$, where $w_n^t = |D_n| / \sum_{j\in S_t} |D_j|$ is the data-proportional weight with $|D_n|$ being the local dataset size. For notational simplicity, let $g_n^{t,b} := \nabla_{\mathcal{S}_n^{t,b}} L_n^{(b)}(\Theta^{t,b})$ denote the partial gradient for client n at batch b. We obtain:

$$\begin{aligned}\mathbb{E}[L(\Theta^{t+1})] \le \mathbb{E}[L(\Theta^t)] \\ - \eta \sum_{b=1}^{K} \mathbb{E}\left[\left\langle \nabla L(\Theta^{t,b}), \sum_{n\in S_t} w_n^t g_n^{t,b} \right\rangle\right] \\ + \frac{L\eta^2}{2} \sum_{b=1}^{K} \mathbb{E}\left[\left\| \sum_{n\in S_t} w_n^t g_n^{t,b} \right\|^2\right]\end{aligned} \tag{18}$$

For each mini-batch b, decomposing the gradient over B blocks as $\nabla L_n^{(b)}(\Theta) = \sum_{i=1}^{B} d_i^{n,t,b} \nabla_{B_i} L_n^{(b)}(\Theta)$, where $\nabla_{B_i} L_n^{(b)}(\cdot)$ is the gradient w.r.t. block B_i. Since block selection is independent across mini-batches and $\mathbb{E}[d_i^{n,t,b}] = p_i \ge p_{\min}$ for all b, we have:

$$\mathbb{E}\left[\left\langle \nabla L(\Theta^{t,b}), \sum_{n\in S_t} w_n^t \nabla_{\mathcal{S}_n^{t,b}} L_n^{(b)}(\Theta^{t,b}) \right\rangle\right] \ge p_{\min} \left\| \nabla L(\Theta^{t,b}) \right\|^2 \tag{19}$$

Bounding the variance term by $\frac{2G^2}{\rho} + 2\sigma^2 + G^2 \sum_{i=1}^{B} p_i(1-p_i)$, applying strong convexity, and unrolling the recursion completes the proof. □

Remark IV.2 (Convergence Rate)**.** *Setting $\eta = O(1/\sqrt{TK})$ achieves rate*

$$O\left(\frac{1}{\sqrt{TK}}\right) + O\left(\frac{1}{p_{\min}}\right) + O\left(\sum_{i=1}^{B} p_i(1-p_i)\right), \tag{20}$$

where K is the number of local mini-batches per round. The first two terms match standard FedAvg convergence with an additional $O(1/p_{\min})$ term quantifying the cost of partial block training. The third term captures the structural variance introduced by per-batch stochastic block selection, which vanishes as the policy network converges (Theorem IV.1).

2) Training Stability: A key concern in FedAD is whether the stochastic block selection introduces excessive gradient variance that could destabilize training. We show that while block selection does introduce additional variance, thisstructural variance vanishes as training progresses, ensuring stable convergence.

Lemma IV.3 (Gradient Variance Decomposition)**.** *The gradient variance under FedAD decomposes as:*

$$\mathrm{Var}\left[g_n^{t,b}\right] \le \underbrace{\sigma_{\text{data}}^2}_{\textit{mini-batch variance}} + \underbrace{G^2 \sum_{i=1}^{B} p_i(1-p_i)}_{\textit{structural variance}} \tag{21}$$

where σ_{data}^2 is the standard mini-batch sampling variance (present in any SGD method), and the structural variance arises from stochastic block selection.

The structural variance $\sum_{i=1}^{B} p_i(1-p_i)$ is maximized when $p_i = 0.5$ for all blocks (i.e., completely random selection), yielding maximum variance $0.25B$. In contrast, when $p_i \in \{0, 1\}$ (deterministic selection), the structural variance vanishes entirely.

Theorem IV.4 (Vanishing Structural Variance)**.** *As the policy network converges, the adjustment matrix V and temperature τ in Gumbel-Softmax drive $p_i \to 0$ or $p_i \to 1$, ensuring*

$$\lim_{\text{training}} \sum_{i=1}^{B} p_i(1-p_i) = 0, \tag{22}$$

thus eliminating structural variance.

Remark IV.5. *Theorem IV.4 implies that FedAD is self-stabilizing: early in training, the policy explores different block configurations (high structural variance), but as the policy learns which blocks are important for each input, it converges to near-deterministic decisions (low structural variance).*

3) Training Mechanism Reliability: Partial block training raises three natural concerns: (1) **block starvation**, where some blocks may never be updated; (2) **learning desynchronization**, where the policy network and classifier may fail to learn synergistically; and (3) **aggregation bias**, where partial aggregation may introduce systematic errors. We address each of these concerns below.

Theorem IV.6 (Block Training Guarantee)**.** *Consider a FedAD system with N clients, B blocks, client participation rate ρ, and minimum block retention probability $p_{\min}$. After*

$$T \ge T_0 = \frac{2B \log(B/\delta)}{\rho N p_{\min}} \tag{23}$$

training rounds, all blocks receive at least one update with probability $\ge 1-\delta$.

Proof. Let U_i^T denote the number of updates for block B_i over T rounds. Since each block has retention probability $p_i \ge p_{\min}$ and each round has ρN participating clients in expectation, we have $\mathbb{E}[U_i^T] \ge T\rho N p_{\min}$. By Hoeffding's inequality and union bound over all B blocks:

$$\mathbb{P}(\exists i : U_i^T = 0) \le B \exp(-T\rho N p_{\min}), \tag{24}$$

solving for T such that this probability $\le \delta$ yields T_0. □

Theorem IV.7 (Policy Network Gradient Flow)**.** *The* `detach()` *operation in Algorithm 1 (line 15) enables gradient-based optimization of the policy network parameters $\tilde{\Theta}$.*

This allows joint optimization:

$$\min_{\Theta, \tilde{\Theta}} \mathbb{E}_{(\mathbf{x},y), \mathbf{d}\sim\pi(\cdot|\mathbf{x},\tilde{\Theta})}[L(f(\mathbf{x};\Theta,\mathbf{d}), y)] \tag{25}$$

subject to resource constraint.

Proof Sketch. The detach() operation creates a computation graph where:

$$\frac{\partial L}{\partial \tilde{\Theta}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \mathbf{d}} \cdot \frac{\partial \mathbf{d}}{\partial \mathbf{p}} \cdot \frac{\partial \mathbf{p}}{\partial \tilde{\Theta}} \tag{26}$$

The key insight: $\frac{\partial \mathbf{d}}{\partial \mathbf{p}} = \mathbf{I}$ (identity) because:

$$\mathbf{d} = (\mathbf{d} - \mathbf{p}).\text{detach}() + \mathbf{p} \tag{27}$$

$$= \text{constant} + \mathbf{p} \quad \text{(in gradient computation)} \tag{28}$$

Thus gradients bypass the discrete sampling and flow directly to the policy network via the continuous Gumbel-Softmax probabilities $\mathbf{p}$. This is mathematically equivalent to the REINFORCE gradient estimator with a baseline, but with lower variance. □

Remark IV.8 (Training Stability)**.** *The cross-entropy loss L_c (Equation 11) provides direct supervision for both networks:* ***classifier*** *learns to make accurate predictions given the blocks it receives, and* ***policy network*** *learns to drop less important blocks first.*

The resource constraint ensures the policy network learns to operate within device limits.

Theorem IV.9 (Partial Aggregation Consistency)**.** *FedAD's partial aggregation is equivalent to standard FedAvg with implicit zero weights for untrained blocks:*

$$_i\theta^t = \sum_{n \in S_t} \tilde{w}_n^i \cdot {}_i\theta_n^t \tag{29}$$

where $\tilde{w}_n^i = \frac{|D_n|}{\sum_{j \in S_t, u_j^i = 1} |D_j|} \mathbb{1}[u_n^i = 1]$.

Corollary IV.10. *Over many rounds, expected weights converge to data-proportional:*

$$\lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}[\tilde{w}_n^i] = \frac{|D_n|}{\sum_{j=1}^{N} |D_j|}. \tag{30}$$

V. Performance Evaluation

We evaluate FedAD across different datasets, models, device computing resource distribution to demonstrate its performance and generality.

A. Dataset and Models

We evaluate FedAD on three image classification datasets: Cifar100 (C100) [47], Tiny-ImageNet (Tiny) [48], and EMNIST (EM) [49]. Each dataset is split into 100 disjoint parts for 100 clients. We use the Dirichlet distribution $Dir(\beta = 0.1)$ to simulate non-IID data scenarios [50]. The smaller the value of β, the more heterogeneous the setting is. We employ ResNet18 (RN18), ResNet34 (RN34), ResNet50 (RN50) [22], RegNet-200MF (Reg) [51], and MobileNet-v2 (MN2) [23] as classifier with 8, 16, 16, 9, and 10 droppable blocks respectively.

B. Compared Methods

We compare FedAD with the following methods:

- **FedAvg (f. res.)**: Similar with authors in [18] and [21], we choose FedAvg with full resources as one baseline. It serves as a theoretical upper bound where all devices train the full model [18].
- **HeteroFL** [13]: A method for adjusting the size of the neural network by shorting the width of the hidden channels. The shrinkage ratio for each device remains constant during training.
- **FjORD** [4]: It is similar with HeteroFL, and the main difference is each device trains different drop levels (within their capabilities) during training.
- **Federated Dropout** [5]: It uses the same dropout rate for all layers to reduce the computational complexity of the model. We also extend this method to allow for different dropout rates for devices according to the resource availability.
- **FedRolex** [26]: It employs a rolling sub-model extraction scheme to allow different parts of the global model to be evenly trained. We also extended it as described in the illustration of Federated Dropout.
- **ScaleFL** [16]: It scales down the model along both width and depth via early exit classifiers, and applies self-distillation among exit predictions to enhance knowledge transfer across sub-networks.
- **NeFL** [17]: It divides the model into sub-models through widthwise and depthwise scaling, and decouples a subset of parameters for each sub-model to handle the inconsistency across heterogeneous sub-model architectures.
- **FedAvg**: In this setting, devices without full resources are dropped from the training process, and the reserved devices perform FedAvg.

C. Implementation Details

We implement FedAD using PyTorch-2.0 on NVIDIA A40 (37.13 TFLOPS) and NVIDIA RTX A6000 GPU (38.71 TFLOPS), and the source code with detailed settings are accessible at https://github.com/cassie-bwtu/FedAD. To simulate device heterogeneity, we divide device computing power levels into 1, 0.66, and 0.33, and the number of devices with different levels is 50, 30, and 20, respectively. For the policy network, we use ResNet with just one standard residual block, which is small enough compared with the classifier. Therefore, the time overhead for training the policy network is negligible. We represent the round time T as the time required to train the average number of samples per client using the complete classifier on devices with sufficient computational resources, which can be calculated by formula (6), and the exact values are shown in Table II. σ and γ in adjustment matrix $\mathbf{V}$ is set to 0.3 and -0.3, and the hyperparameter τ in Gumbel-Softmax is 0.6. The learning rates 0.5 (policy network) and 0.1 (classifier) are constant throughout training, and SGD [52] is used as the optimizer. Client participation rate is 0.1 with one local epoch per round, totaling 1000 rounds with early stopping (100 rounds without improvement). For each dataset, we combine the official train and test splits and

TABLE II: Experimental settings

Dataset	Classifier	# Droppable blocks	Proportion	Hardware	Device computing power (TFLOPS)	Round time T (second)
Cifar100	ResNet18	8	12.97%	NVIDIA A40	{37.13, 22.28, 11.14}	2.15
EMNIST	ResNet34	16	6.19%	NVIDIA A40	{37.13, 22.28, 11.14}	54.75
Tiny-Imagenet	ResNet34	16	9.65%	NVIDIA A40	{37.13, 22.28, 11.14}	31.78
EMNIST	ResNet50	16	4.64%	NVIDIA RTX A6000	{38.71, 23.23, 11.61}	59.51
Cifar100	RegNet	9	8.40%	NVIDIA RTX A6000	{38.71, 23.23, 11.61}	3.46
Tiny-Imagenet	MobileNet	10	7.41%	NVIDIA RTX A6000	{38.71, 23.23, 11.61}	2.69

Proportion = MACs of policy network / MACs of classifier

TABLE III: The test accuracy (%, ↑), time consumption (minutes, ↓), and total communication cost (G, in number of parameters, ↓) of the six experimental groups. The best results are indicated using bold typeface, and the second best are underlined.

Methods	C100/RN18			EM/RN34			Tiny/RN34			EM/RN50			C100/Reg			Tiny/MN2		
	Acc.	Time	Comm.	Acc.	Time	Comm.	Acc.	Time	Comm.	Acc.	Time	Comm.	Acc.	Time	Comm.	Acc.	Time	Comm.
FedAvg (f. res.)	**34.81** ±0.42	102.00	141.72	84.15 ±0.18	88.79	100.16	20.35 ±0.36	146.00	101.82	**84.31** ±0.21	35.95	101.65	34.86 ±0.45	257.13	22.98	**19.60** ±0.42	138.60	13.94
HeteroFL	25.41 ±1.13	**73.54**	**91.45**	80.61 ±0.45	153.22	110.87	16.23 ±0.95	218.22	107.19	78.26 ±0.58	131.09	158.45	20.07 ±1.27	**105.07**	**10.96**	14.97 ±0.88	85.88	11.18
FjORD	24.33 ±0.98	133.10	131.11	79.12 ±0.52	129.25	101.23	13.41 ±1.12	275.48	109.42	76.07 ±0.65	151.21	160.90	23.73 ±1.05	164.92	11.90	15.65 ±1.02	98.85	11.31
Federated Dropout	12.35 ±1.85	120.82	130.14	73.28 ±0.78	369.48	184.66	9.46 ±1.45	**76.79**	**39.08**	74.46 ±0.92	1335.65	386.08	17.52 ±1.63	267.83	20.77	10.87 ±1.38	88.22	11.48
FedRolex	28.60 ±0.86	91.68	98.29	**86.14** ±0.31	507.76	223.96	13.46 ±0.78	125.53	97.14	83.07 ±0.38	1231.53	381.17	28.13 ±0.82	279.21	21.22	16.09 ±0.74	171.46	15.84
ScaleFL	30.48 ±0.51	80.43	132.85	76.78 ±0.28	109.18	82.22	18.49 ±0.47	127.30	96.74	74.57 ±0.33	36.16	93.35	-	-	-	-	-	-
NeFL	31.51 ±0.45	92.79	133.91	80.56 ±0.22	94.63	105.48	19.83 ±0.39	131.74	92.02	81.37 ±0.27	65.07	98.26	32.03 ±0.52	149.75	27.96	-	-	-
FedAvg	28.29 ±0.62	231.42	102.65	81.39 ±0.27	115.64	**56.44**	15.89 ±0.69	372.62	92.49	81.19 ±0.31	53.58	**88.53**	32.20 ±0.71	203.41	11.42	12.99 ±0.65	**72.53**	**6.61**
FedAD (ours)	34.34 ±0.34	86.00	126.45	83.54 ±0.19	**75.07**	85.66	**20.73** ±0.32	123.00	88.80	83.46 ±0.23	**28.00**	89.27	**35.97** ±0.41	168.21	13.47	19.59 ±0.36	79.57	10.91

Acc. is the abbreviation of accuracy. *Comm.* denotes the total communication cost measured in number of parameters ($\times 10^9$). "-" indicates architectural incompatibility.

redistribute the samples across 100 clients via $Dir(\beta = 0.1)$ partitioning, followed by a 9:1 client-local train/test split. The test datasets from all clients are combined to form a global test dataset. All results are averaged over 3 independent runs, and accuracy is reported as mean± standard deviation. Following the protocol of DISTREAL [21] and PFLlib [53], total computation time is the cumulative local-training wall-clock time, summed across all sampled clients and rounds, up to the round at which each run reaches its individual best accuracy. Total communication cost measures the cumulative uploaded and downloaded parameters throughout training. The specific experimental settings are noted in Table II.

D. Performance Comparison and Analysis

To verify the superiority of FedAD, we compare it with baselines, evaluating their accuracy, the time required to reach the highest accuracy and total communication costs. Detailed results are provided in Table III.

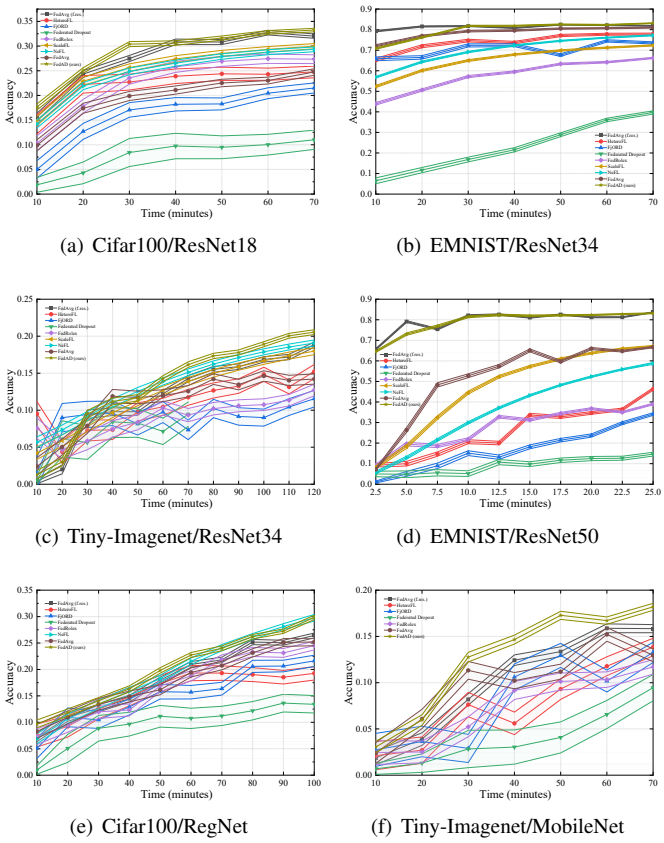
Model Accuracy. FedAD significantly outperforms heterogeneous FL methods in most cases, achieving accuracy improvements of 8.89%, 4.12%, 5.58%, 5.49%, 11.67%, and 5.20% across six experimental setups. Compared to FedAvg (f. res.), our method does not show a significant performance drop, with accuracy decreasing by only 0.08% on average. Additionally, FedAD shows a 4.28% (on average) improvement in accuracy over FedAvg by enabling resource-constrained devices to participate. Only in EM/RN34 does our method fall behind FedRolex, with a 2.6% decrease in accuracy. However, FedRolex takes approximately seven times longer than FedAD to reach this accuracy. This result reveals a trade-off between training efficiency and block training balance: FedAD's adaptive dropping may cause certain blocks to be trained less frequently, while FedRolex's rolling mechanism ensures uniform training. A promising direction is to introduce a balancing regularization term to encourage more uniform block selection while preserving input adaptivity. Another option is a hybrid strategy that uses uniform training in early rounds and switches to adaptive dropping later. Among the stronger baselines, ScaleFL and NeFL, which jointly scale along width and depth, consistently outperform single-dimension methods (HeteroFL, FjORD, Federated Dropout), with NeFL particularly competitive on EM/RN50 (81.37%) and C100/Reg (32.03%). Nevertheless, FedAD still outperforms ScaleFL and NeFL by 5.44% and 2.55% on average, indicating that input-aware adaptive block dropping yields finer-grained resource utilization than fixed 2-D splits.

Training Time. FedAD significantly reduces training duration due to its faster convergence. Even compared to FedAvg (f. res.), our method requires less time, mainly because it reduces the model size for devices with restricted computational resources, and training smaller models on the same hardware is faster. Training time sums every sampled client's local training time across all rounds and thus reflects the algorithm's intrinsic compute cost rather than the wall-clock duration of a real deployment. Because clients in a real system train synchronously, per-round latency is determined by the slowest client rather than the sum over all clients, and the absolute training time also depends on the client hardware, which is typically weaker than the server used in our simulation. A real deployment would also incur communication time and server-side aggregation time, which are orthogonal to the compute efficiency that time designed to measure. Nevertheless, the advantage FedAD shows in Table III transfers consistently to real deployment: by assigning smaller sub-networks to resource-constrained devices that would otherwise act as stragglers, FedAD simultaneously reduces both the cumulative compute cost reported here and the straggler-determined per-round latency of an actual synchronous FL system.

Total Communication Cost. It measures the cumulative uploaded and downloaded parameters throughout training. As shown in Table III, FedAD consistently reduces communication overhead compared to FedAvg (f. res.) across all settings, with reductions ranging from 10.8% (C100/RN18) to 41.4% (C100/Reg). Although FedAD introduces additional

policy networks, their lightweight design incurs negligible communication overhead. The overall reduction is primarily attributed to the accelerated convergence and compact model configurations enabled by adaptive block dropping. Compared to FedAvg, although it achieves lower communication costs in certain cases (e.g., 56.44G on EM/RN34 vs. 85.66G for FedAD), this comes at the cost of significantly degraded accuracy (81.39% vs. 83.54%). Among heterogeneous FL methods, Federated Dropout and FedRolex exhibit highly unstable communication costs across different settings, while HeteroFL and FjORD achieve comparable or lower costs in some cases, their accuracy consistently falls behind FedAD. ScaleFL and NeFL yield communication costs comparable to FedAD. ScaleFL achieves the lowest cost among heterogeneous methods on EM/RN34 (82.22G vs. 85.66G for FedAD), but at a 6.76% accuracy drop. On other settings, both incur higher costs than FedAD, as their fixed split configurations cannot exploit the per-input adaptivity of FedAD's drop policy.

Overall, FedAD achieves a favorable trade-off between model performance and communication efficiency, with reduced training time.



(a) Cifar100/ResNet18 (b) EMNIST/ResNet34

(c) Tiny-Imagenet/ResNet34 (d) EMNIST/ResNet50

(e) Cifar100/RegNet (f) Tiny-Imagenet/MobileNet

Fig. 4: Accuracy of each method under different local training times before convergence. Shaded regions indicate standard deviation. In (c), the accuracy for Federated Dropout after 70 minutes is missing because it converged around 76 minutes.

Furthermore, we recorded the global model accuracy at regular intervals of local training time, as shown in Fig. 4. Although FedAD does not always train the complete model, it still achieves accuracy comparable to or higher than FedAvg (f. res.) within the same local training time, especially in Fig. 4 (c), (e), and (f). Training sub-networks reduces the time of each round, allowing FedAD to complete more rounds and reach higher accuracy within a fixed time budget. Compared with the other heterogeneous FL methods, FedAD achieves the highest accuracy at most time points and shows faster convergence. Although FedRolex attains a final accuracy close to FedAD on EMNIST, its improvement requires longer training time. Some methods such as Federated Dropout and HeteroFL converge faster in a few settings, but their accuracy remains significantly lower. In conclusion, FedAD achieves rapid accuracy improvement and superior final accuracy. This suggests that it can effectively utilize limited device resources, reach high performance in shorter time, and maintain a better balance between efficiency and accuracy.

E. Validation of the Computational Model

To validate the FLOPS/MACs abstraction in Eqs. (5)–(7), we measure the wall-clock per-batch training time across all six experimental groups under drop rates $\{0.0, 0.2, 0.4, 0.6, 0.8\}$ with 15 random block combinations per rate. As shown in Fig. 5, predicted and measured times exhibit near-perfect linear correlation in every group (Pearson $r \in [0.957, 0.997]$). The absolute offset reflects the well-known gap between peak and sustained GPU throughput, captured by a per-group utilization factor $\eta \in [0.5\%, 3.9\%]$. As analyzed in Section IV-A, FedAD's drop decisions depend only on the relative ranking of configurations, preserved with $r > 0.95$ in every case, making the method robust to this absolute offset.

F. Adaptability to Real-time System

A key feature of FedAD is adapting to dynamically changing computational resources of the device in real-time systems by generating drop policies per batch. To verify its adaptability, we dynamically change the computational resources constraints of each device per round during training. Besides, to apply static baselines to dynamic scenarios, we use: (1) ***Conservative Strategy***: set devices' computing power to minimum level during the dynamic changes in resources. (2) ***Discarding Strategy***: exclude devices when computing power drops below initial value. The experimental results are shown in Table IV.

FedAD consistently outperforms static methods in real-time systems under both strategies. Under the conservative strategy, static methods suffer significant accuracy degradation because devices must waste available resources to meet minimum computational constraints, resulting in smaller sub-networks and inadequate global model training. Under the discarding strategy, static methods show improvement but still underperform due to reduced device participation when computational resources decrease. ScaleFL achieves the highest accuracy among the static methods but still falls far behind adaptive ones. NeFL inherently supports dynamic resources like FedAD, yet FedAD still outperforms it by 1.17%–3.10% across the four settings, owing to its per-batch input-aware

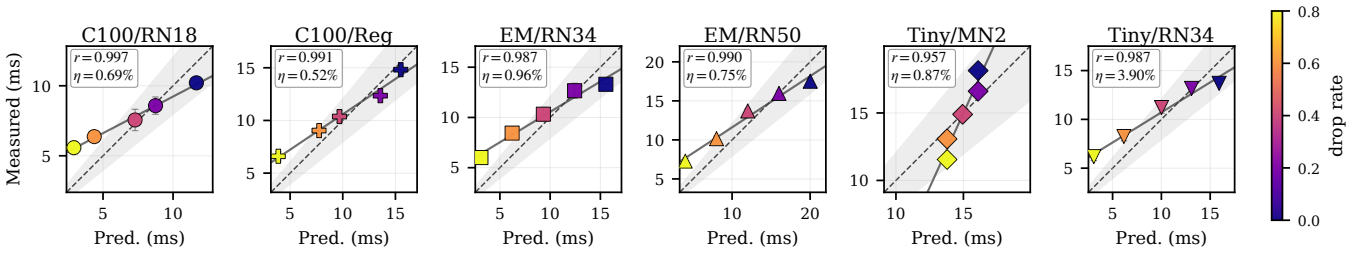


Fig. 5: Per-group validation of FedAD's computational model. Each subplot plots measured against calibrated predicted per-batch training time across five drop rates (encoded by marker colour). Solid line: least-squares fit; dashed: $y = x$; shaded: $\pm 20\%$ band. Pearson $r \in [0.957, 0.997]$ confirms strong linearity in every group.

TABLE IV: The test accuracy (%) of the four experimental groups with dynamic changes in device computational resources.

Strategies	Conservative Strategy				Discarding Strategy			
Groups	C100/RN18	EM/RN34	Tiny/RN34	EM/RN50	C100/RN18	EM/RN34	Tiny/RN34	EM/RN50
HeteroFL	3.47	27.90	9.46	10.79	22.70	75.95	13.37	70.18
FjORD	3.41	39.77	9.46	15.43	18.22	75.87	9.51	75.50
Federated Dropout	8.56	62.42	9.46	60.42	10.20	67.71	9.47	61.55
FedRolex	9.54	32.39	9.46	23.58	23.66	82.63	12.89	81.57
ScaleFL	18.27	66.31	12.28	64.37	26.56	73.17	16.34	72.09
NeFL	30.53	79.22	19.39	81.58	30.53	79.22	19.39	81.58
FedAD	**32.71**	**82.32**	**22.39**	**82.75**	**32.71**	**82.32**	**22.39**	**82.75**

Strategies are only applied to static methods. NeFL and FedAD are adaptive, so their values are identical across the two strategies.

TABLE V: The test accuracy (%) of the six experimental groups with different frequency of resource changes.

Change Frequency	C100/ RN18	EM/ RN34	Tiny/ RN34	EM/ RN50	C100/ Reg	Tiny/ MN2
1 round	32.71	82.32	**22.39**	82.75	36.32	**19.79**
3 round	33.79	83.37	22.29	82.35	35.56	18.45
5 round	33.53	83.85	21.60	82.21	36.65	18.46
10 round	33.63	**84.33**	21.53	83.32	**36.77**	18.65
Static	**34.34**	83.54	20.73	**83.46**	35.97	19.59

Static means the computing power for each device is not change.

drop policy that adapts more finely than NeFL's per-iteration submodel selection. In contrast, FedAD's adaptability enables devices to fully utilize available resources: it scales down model sizes when resources are limited to maintain participation, and scales up when resources increase to train more blocks effectively. This flexibility ensures more clients contribute to training each block, thereby enhancing global model performance.

To further validate FedAD's adaptability in real-time systems, we alter the frequency of resource changes (every 1, 3, 5, and 10 rounds). As shown in Table V, FedAD maintains comparable or even slightly higher accuracy under dynamic conditions compared to static resource constraints. Across different change frequencies and experimental groups, the average variance in our method's accuracy is only 0.39, demonstrating FedAD's adaptability and robustness. This stability stems from FedAD's ability to dynamically adjust training resources requirements by modifying the number of dropped blocks, enabling fine-grained and rapid responses to resource fluctuations while fully utilizing available resources.

G. Analysis of the Policy Network

1) Efficiency of Drop Policy: To verify the effectiveness of the drop policy generation method in FedAD, we compare it with the following heuristic methods, and summarize results in Table VI:

- **Random drop per batch (Random-Batch)**: Randomly drops blocks per batch according to computational constraints.
- **Random drop per round (Random-Round)**: Randomly drops blocks per round, keeping the sub-model fixed within each round.
- **Constant sub-model (Constant)**: Randomly drops blocks once before training, keeping the sub-model fixed throughout the entire training process.
- **Deep-First:** A training-free heuristic that always drops blocks from the shallowest end first until the resource constraint is satisfied.
- **MACs-Greedy:** A training-free heuristic that greedily drops blocks in descending order of their individual MACs.

It is evident that FedAD consistently achieves the highest accuracy across all settings. The performance of the constant sub-model is the worst, likely because the blocks that are dropped are trained by fewer times, which negatively impacted the overall model performance. Random-Round has the smallest gap with FedAD (0.92% on average). Although Random-Batch generates drop policies per batch like FedAD, its random selection yields notably lower accuracy, demonstrating that our policy network effectively generates input-aware drop policies. Among the heuristics, Deep-First is the most competitive, yet it still falls behind the policy network.

TABLE VI: The test accuracy (%) of the six experimental groups with different drop policy generation methods.

Methods	C100/ RN18	EM/ RN34	Tiny/ RN34	EM/ RN50	C100/ Reg	Tiny/ MN2
Random-Batch	31.29	83.20	19.46	82.50	30.93	19.18
Random-Round	32.46	83.09	19.85	83.13	34.30	19.29
Constant	28.59	82.56	17.05	82.17	33.17	17.81
Deep-First	32.84	81.90	19.93	82.84	33.99	18.50
MACs-Greedy	28.10	81.69	18.33	81.55	31.67	18.14
FedAD	**34.34**	**83.54**	**20.73**	**83.46**	**35.97**	**19.59**

TABLE VII: Ablation study on policy network architecture.

Policy Network Architecture	C100/RN18		EM/RN34		EM/RN50	
	Acc.	Time	Acc.	Time	Acc.	Time
2 CNNs	32.10	112.73	81.67	45.56	81.38	34.76
1 ResBlock	34.34	86.00	83.54	75.07	83.46	28.00
2 ResBlock	32.19	101.02	82.13	87.00	82.60	34.70
3 ResBlock	30.48	115.40	82.16	96.27	82.67	45.31

MACs-Greedy lags further behind, underperforming the policy network by up to 4.30% on C100/RN18. These results confirm that input-agnostic dropping rules, regardless of their design principle, cannot match the input-aware decisions of the policy network, which consistently adapts the drop policy to each batch at negligible additional cost (4.64%-12.97% extra MACs, Table II).

2) Ablation on Policy Network Architecture: To investigate the impact of policy network architecture on overall performance, we conduct ablation studies by varying the structure of the policy network. We vary its architecture from a simple two-layer CNN to residual networks with 1, 2, and 3 residual blocks. As shown in Table VII, the single residual block configuration consistently achieves the highest accuracy across all settings (34.34%, 83.54%, and 83.46%, respectively), while also maintaining competitive or superior training efficiency. Notably, increasing the network depth beyond one residual block yields no further gains and even causes slight performance degradation. This suggests that excessive parameters can lead to overfitting and unstable gradient computation. Additionally, overly simple architectures (e.g., 2CNNs) lack sufficient capacity to capture discriminative features for effective drop policy generation. These results justify our choice of a single residual block, which provides an effective trade-off between representation capacity and computational cost.

3) Routing Consistency: Here, routing consistency refers to the stability of the drop policy generated for the same input batch across consecutive training rounds. To assess the stability of the policy network's decisions during training, we analyze routing consistency by computing the Jaccard similarity between consecutive rounds. Specifically, for each batch, we measure the overlap of selected samples between round t and round $t-1$. A higher Jaccard similarity indicates more consistent routing decisions. As illustrated in Figure 6, we track five randomly sampled batches from a single client on EM/RN34 over 200 training rounds. In the early stage, the Jaccard similarity remains relatively low, reflecting the exploratory phase where the policy network is still learning to identify informative samples. As training progresses, the similarity steadily increases, exceeding 81% after round 100. In the final stage, most batches achieve near-perfect consistency, indicating that the policy network has converged to stable selection decisions. This trend demonstrates that the policy network gradually learns consistent routing patterns rather than making arbitrary selections, which is essential for reliable and reproducible sample selection in federated learning.

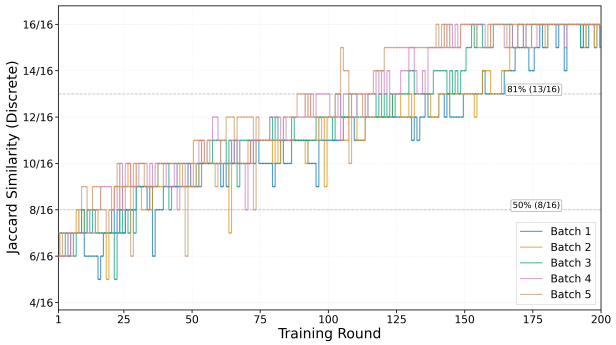


Fig. 6: The Jaccard similarity of five batch samples in the EM/RN34 experimental group.

TABLE VIII: Per-batch runtime breakdown under three computing levels (batch size 10). Values are in milliseconds.

Model	Level	PN phase	Cls fwd	Total	Overhead
C100/RN18	1.00	1.96	5.65	15.02	13.07%
	0.66	3.08	3.85	12.75	24.09%
	0.33	3.00	2.48	9.86	29.79%
EM/RN34	1.00	2.57	13.45	29.28	8.76%
	0.66	4.53	9.72	24.76	18.14%
	0.33	5.69	7.53	21.92	25.90%
Tiny/RN34	1.00	2.91	13.73	31.40	9.23%
	0.66	6.29	11.94	30.13	20.63%
	0.33	7.04	7.86	24.20	29.15%
EM/RN50	1.00	3.65	20.02	44.43	8.22%
	0.66	4.69	11.10	27.73	16.82%
	0.33	8.46	10.01	28.72	29.47%
C100/Reg	1.00	3.08	10.40	27.03	11.33%
	0.66	5.85	10.33	30.07	19.82%
	0.33	5.68	5.56	19.59	28.72%
Tiny/MN2	1.00	3.25	12.41	29.99	10.83%
	0.66	4.90	10.27	26.34	18.64%
	0.33	7.33	9.76	26.41	27.70%

4) Policy Network Overhead: Beyond the MACs proportion reported in Table II, we further quantify the policy network's (PN) actual runtime and memory through micro-benchmarks. We measure per-batch wall-clock time on the same device with batch size 10 across six experimental setups and three computing levels $\{1.0, 0.66, 0.33\}$. Each batch is decomposed into four pipelined phases: the PN phase (PN forward, Gumbel-Softmax sampling, and drop policy generation), classifier forward, joint backward, and optimizer step. Each configuration is averaged over 30 batches after 10 warm-ups.

Runtime overhead. Table VIII summarizes the PN phase time, classifier forward time, total batch time, and overhead

TABLE IX: The test accuracy (%), training round and time consumption (minutes) of the six experimental groups with a different number of client training epochs.

Epochs	C100/RN18			EM/RN34			Tiny/RN34			EM/RN50			C100/Reg			Tiny/MN2		
	Acc.	R.	Time	Acc.	R.	Time	Acc.	R.	Time	Acc.	R.	Time	Acc.	R.	Time	Acc.	R.	Time
1	**34.34**	600	**86.00**	83.54	215	**75.07**	**20.73**	236	**123.00**	**83.46**	83	**28.00**	**35.97**	495	**168.21**	**19.59**	236	**79.57**
5	32.27	421	400.09	**83.66**	83	154.96	19.62	293	718.14	82.84	62	111.05	34.19	**471**	1011.47	19.16	185	265.10
10	33.20	235	432.56	82.91	83	313.63	18.81	227	1121.99	83.08	**53**	184.47	34.05	507	1960.06	18.65	163	469.71
20	33.21	**219**	873.26	82.45	**83**	635.17	18.19	**157**	1576.28	82.71	156	1095.28	33.16	521	3580.23	17.56	**137**	637.11

R. is the shorten of round.

ratio. Three observations stand out. (i) At the full-resource level, the PN phase takes only 2.0–3.7 ms per batch, yielding an overhead of 8.2%–13.1%, consistent with the MACs proportion in Table II. (ii) Our Gumbel noise sampling runs directly on GPU and takes only 0.2–0.7 ms per batch, avoiding any CPU-to-GPU bottleneck. (iii) Block-dropping savings far exceed PN overhead: on Cifar100/RN18, the total batch time drops from 15.0 to 9.9 ms as the level decreases from 1.0 to 0.33, where the PN's $\sim$1 ms absolute increase is far exceeded by the $\sim$5.9 ms saved by block dropping. This net gain holds across all configurations; the rising overhead ratio at lower levels (up to 29.79%) simply reflects a shrinking denominator rather than a PN slowdown.

Peak memory. Per-batch GPU peak memory ranges from 57 to 232 MB and decreases at lower computing levels (e.g., 121 $\rightarrow$ 92 MB on Cifar100/RN18), aligning with block-dropping savings. Overall, the PN introduces negligible runtime and memory overhead, and FedAD's training-time savings consistently outweigh the PN cost.

H. Hyperparameter Study

1) Local Training Epochs: In FL, clients can reduce communication overhead by performing multiple local training epochs before uploading updates [46]. As shown in Table IX, increasing local epochs generally reduces communication rounds but extends total training time. Notably, RegNet's convergence is less sensitive to local epochs, possibly because it relies more on global updates. Regarding accuracy, setting local epochs to 1 or 5 yields better results, while higher values lead to degradation. This is likely due to overfitting to local data distributions, which harms generalization during global aggregation. Overall, FedAD maintains stable performance across different local training epochs.

2) Join Ratio: Join ratio denotes the proportion of clients participating in each round. As shown in Fig. 7, on EMNIST and CIFAR-100 datasets, accuracy slightly decreases as the join ratio increases, likely because greater client participation introduces more data heterogeneity, making generalization more challenging. In contrast, FedAD shows stable performance on Tiny-ImageNet with minimal fluctuations. Overall, although higher join ratios may slightly affect accuracy, FedAD maintains high robustness across different settings.

The accuracy decline at higher join ratios is a common challenge in FL with non-IID data, as increased client participation introduces more heterogeneous local updates that may conflict during aggregation. To mitigate this issue, future work could explore: (1) client clustering strategies that group clients with similar data distributions; (2) regularization techniques that constrain local updates to remain closer to the global model; and (3) similarity-aware aggregation weights that consider both data distribution and drop policy similarity among clients.

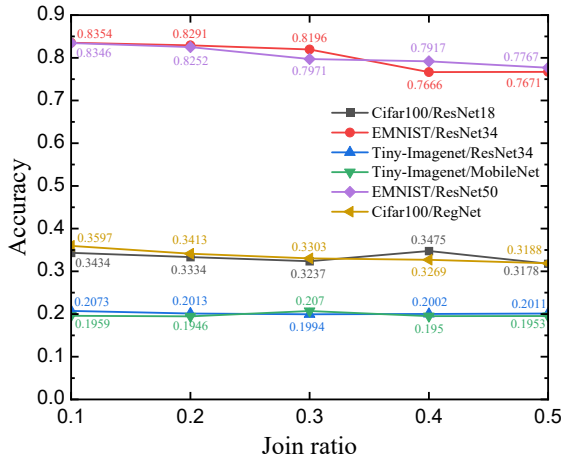


Fig. 7: The test accuracy (%) of the six experimental groups with different join ratios.

TABLE X: The test accuracy (%) of the six experimental groups with different value of τ.

τ	C100/ RN18	EM/ RN34	Tiny/ RN34	EM/ RN50	C100/ Reg	Tiny/ MN2
0.1	31.23	82.58	20.02	81.93	34.20	19.02
0.6	**34.34**	**83.54**	**20.73**	**83.46**	**35.97**	**18.59**
0.8	31.66	82.63	20.38	82.69	34.84	18.51
1.0	32.21	81.21	18.86	82.27	34.34	19.04
10.0	29.43	82.34	19.94	81.49	34.23	18.73

3) Temperature Parameter τ: In the Gumbel Softmax trick, τ controls the discreteness of the output. As shown in Table X, when tau is too small (e.g., 0.1), the model's accuracy is relatively low. This could be because the Gumbel Softmax distribution is closer to a hard one-hot encoding, causing the model to favor hard selections of certain route while ignoring other potential model structures. Conversely, when τ is too large (e.g., 10), the distribution becomes overly smooth, making route selection nearly random and preventing effective learning. The results suggest that in FedAD, $\tau = 0.6$ might be an optimal parameter choice.

I. Deployment Flexibility

To verify FedAD's flexibility in practical deployment, we explore two aspects: computing power level distribution and

TABLE XI: The test accuracy (%) of the four experimental groups with different computing power level distribution.

Computing Power Level Distribution	C100/ RN18	EM/ RN34	Tiny/ RN34	EM/ RN50
{1, 0.66, 0.33} {50, 30, 20}	**34.34**	83.54	**20.73**	83.46
{1, 0.75, 0.5, 0.25} {50, 20, 20, 10}	33.10	**84.45**	19.89	**83.57**
{1, 0.8, 0.6, 0.4, 0.2} {50, 20, 15, 10, 5}	33.59	83.73	20.53	83.02

In the first column, the first row represents the computing power levels, and the second row shows the corresponding number of clients.

TABLE XII: The test accuracy (%) of the four experimental groups with different device computing power distribution.

Device Number	C100/ RN18	EM/ RN34	Tiny/ RN34	EM/ RN50
{0, 60, 40}	29.51	78.12	17.46	80.43
{20, 48, 32}	32.04	82.05	19.57	81.30
{50, 30, 20}	34.34	83.54	**20.73**	83.46
{80, 12, 8}	**34.96**	**84.81**	20.51	**84.44**
{100, 0, 0}	34.81	84.15	20.35	84.31

Device number refers to the number of devices corresponding to computing power levels 1.0, 0.66, and 0.33.

device computing power distribution.

1) Computing Power Level Distribution: As described in Section V-B, our default setting divides device computing power levels into {1, 0.66, 0.33} with corresponding counts of {50, 30, 20}. We conduct additional experiments with different distributions as shown in Table XI. The rest of the experimental settings are consistent with section V-B. The experimental results (Table XI) show that FedAD's accuracy fluctuates slightly across datasets under different device computing power levels and counts, but remains within a stable range. This indicates strong flexibility in adapting to various hardware environments while maintaining robustness against device heterogeneity and resource constraints.

2) Device Computing Power Distribution: In this section, we explore FedAD's performance under different device computing power distributions at given computational levels {1.0, 0.66, 0.33}. We vary the number of devices at each computing power level while keeping other settings consistent with Section V-B. The detailed setup and results are shown in Table XII.

Overall, accuracy improves as the number of high computational resource devices increases. However, when the system consists only of low computing power devices (first row of Table XII), the accuracy of FedAD decreases. This likely occurs because the round time T is not adjusted, preventing any device from training a complete ResNet. This highlights a future direction: dynamically adjusting T based on system capacity and network conditions, as communication constraints (e.g. bandwidth, latency) also affect effective training time in real-world deployments. A joint optimization approach considering both computational and communication constraints would enhance FedAD's robustness in diverse deployment environments. Despite significant variations in device distribution, FedAD's accuracy generally remains stable at a high level, especially when sufficient high-resource devices are present. This demonstrates FedAD's strong adaptability and robustness across different hardware environments.

J. Applicability to Other Architectures

While our experiments focus on modular CNNs for image classification, FedAD is applicable to any neural network architecture composed of stackable and composable blocks. This includes modular CNNs (e.g., ResNet, MobileNet), transformer-based architectures (e.g., ViT, BERT, GPT, LLaMA), and autoencoders with modular encoder/decoder blocks. With the rapid development of transformers, block-based structures have become the dominant paradigm in modern deep learning. Large language models such as LLaMA-7B (32 blocks) and GPT-3 (96 blocks) are inherently compatible with FedAD's block-dropping mechanism. However, FedAD is not directly applicable to architectures with sequential dependencies (e.g., RNNs/LSTMs) or non-modular structures (e.g., GNNs). Despite these limitations, the broad adoption of block-based architectures ensures FedAD's wide applicability across diverse domains.

VI. Conclusions and Future Work

In this paper, we propose FedAD, an adaptive block-dropping approach for federated learning on resource-constrained devices, which can adapt well to real-time systems. FedAD dynamically generates input-specific drop policies in real-time, adjusting the local training model to account for heterogeneous and time-varying computational resources of devices. At its core, FedAD employs a lightweight policy network that outputs drop probabilities for each block. Combined with time constraints and device capacity, these probabilities form a drop policy that defines the classifier's structure and guides local training. During training, the policy network and classifier are jointly optimized. This enables each device to independently generate fine-grained drop policies tailored to both input and available resources, ensuring strong adaptability to resource fluctuations across devices. Extensive experiments validate that FedAD converges faster and outperforms state-of-the-art methods in resource-limited and heterogeneous environments, while achieving comparable performance in resource-rich settings. Experiments also emphasize FedAD's strong adaptability to dynamic resource changes across devices in real-time system.

As part of our future work, we will investigate balancing mechanisms to ensure more uniform block training, similarity-aware aggregation techniques to handle increased client heterogeneity, and joint optimization approaches that consider both computational and communication constraints for diverse deployment scenarios. We also plan to explore lightweight feature statistic calibration (e.g., normalization-aware aggregation) to align dynamic sub-networks without extra communication cost, and Byzantine-robust aggregation methods (e.g., Krum or coordinate-wise median) to strengthen the policy network's robustness against adversarial manipulation.

ACKNOWLEDGMENTS

This work is supported by the Program of Technology Innovation of the Science and Technology Commission of Shanghai Municipality (Granted No. 21511104700) and the Interdisciplinary Program of Shanghai Jiao Tong University (project number YG2024QNB05).

REFERENCES

[1] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao, "A survey on federated learning," Knowledge-Based Systems, vol. 216, p. 106775, 2021.

[2] C. J. Hoofnagle, B. Van Der Sloot, and F. Z. Borgesius, "The european union general data protection regulation: what it is and what it means," Information & Communications Technology Law, vol. 28, no. 1, pp. 65–98, 2019.

[3] R. Bonta, "California consumer privacy act (ccpa)," Retrieved from State of California Department of Justice: https://oag. ca. gov/privacy/ccpa, 2022.

[4] S. Horvath, S. Laskaridis, M. Almeida, I. Leontiadis, S. Venieris, and N. Lane, "Fjord: Fair and accurate federated learning under heterogeneous targets with ordered dropout," Advances in Neural Information Processing Systems, vol. 34, pp. 12 876–12 889, 2021.

[5] D. Wen, K.-J. Jeon, and K. Huang, "Federated dropout—a simple approach for enabling federated learning on resource constrained devices," IEEE wireless communications letters, vol. 11, no. 5, pp. 923–927, 2022.

[6] S. Okuno, M. Miwa, and N. Fukumoto, "Towards straggler-tolerant and accuracy-aware distributed dnn training in clouds," in 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid). IEEE, 2021, pp. 614–617.

[7] Y. Xu, Y. Liao, H. Xu, Z. Ma, L. Wang, and J. Liu, "Adaptive control of local updating and model compression for efficient federated learning," IEEE Transactions on Mobile Computing, vol. 22, no. 10, pp. 5675–5689, 2023.

[8] J. Zhang, S. Guo, Z. Qu, D. Zeng, Y. Zhan, Q. Liu, and R. A. Akerkar, "Adaptive federated learning on non-iid data with resource constraint," IEEE Transactions on Computers, vol. 71, no. 7, pp. 1655–1667, 2022.

[9] J. Nguyen, K. Malik, H. Zhan, A. Yousefpour, M. Rabbat, M. Malek, and D. Huba, "Federated learning with buffered asynchronous aggregation," in International Conference on Artificial Intelligence and Statistics. PMLR, 2022, pp. 3581–3607.

[10] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in ICC 2019-2019 IEEE international conference on communications (ICC). IEEE, 2019, pp. 1–7.

[11] Y. G. Kim and C.-J. Wu, "Autofl: Enabling heterogeneity-aware energy efficient federated learning," in MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture, 2021, pp. 183–198.

[12] Z. Liu, D. Li, J. Fernandez-Marques, S. Laskaridis, Y. Gao, S. Z. Li, S. X. Hu, T. Hospedales et al., "Federated learning for inference at anytime and anywhere," arXiv preprint arXiv:2212.04084, 2022.

[13] E. Diao, J. Ding, and V. Tarokh, "Heterofl: Computation and communication efficient federated learning for heterogeneous clients," arXiv preprint arXiv:2010.01264, 2020.

[14] M. Kim, S. Yu, S. Kim, and S.-M. Moon, "Depthfl: Depthwise federated learning for heterogeneous clients," in The Eleventh International Conference on Learning Representations, 2023.

[15] J. Shin, J. Ahn, H. Kang, and J. Kang, "Fedsplitx: Federated split learning for computationally-constrained heterogeneous clients," arXiv preprint arXiv:2310.14579, 2023.

[16] F. Ilhan, G. Su, and L. Liu, "Scalefl: Resource-adaptive federated learning with heterogeneous clients," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2023, pp. 24 532–24 541.

[17] H. Kang, S. Cha, J. Shin, J. Lee, and J. Kang, "Nefl: Nested model scaling for federated learning with system heterogeneous clients," IEEE Transactions on Mobile Computing, vol. 24, no. 8, pp. 6734–6746, 2025.

[18] K. Pfeiffer, M. Rapp, R. Khalili, and J. Henkel, "Cocofl: Communication-and computation-aware federated learning via partial nn freezing and quantization," arXiv preprint arXiv:2203.05468, 2022.

[19] S. A. Reveliotis, Real-time management of resource allocation systems: A discrete event systems approach. Springer Science & Business Media, 2005, vol. 79.

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

[21] M. Rapp, R. Khalili, K. Pfeiffer, and J. Henkel, "Distreal: Distributed resource-aware learning in heterogeneous systems," in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, no. 7, 2022, pp. 8062–8071.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[23] A. G. Howard, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," arXiv preprint arXiv:1704.04861, 2017.

[24] L. Liu and J. Deng, "Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution," in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, no. 1, 2018.

[25] Z. Xu, Z. Yang, J. Xiong, J. Yang, and X. Chen, "Elfish: Resource-aware federated learning on heterogeneous edge devices," Ratio, vol. 2, no. r1, p. r2, 2019.

[26] S. Alam, L. Liu, M. Yan, and M. Zhang, "Fedrolex: Model-heterogeneous federated learning with rolling sub-model extraction," Advances in neural information processing systems, vol. 35, pp. 29 677–29 690, 2022.

[27] D. Wu, R. Ullah, P. Harvey, P. Kilpatrick, I. Spence, and B. Varghese, "Fedadapt: Adaptive offloading for iot devices in federated learning," IEEE Internet of Things Journal, vol. 9, no. 21, pp. 20 889–20 901, 2022.

[28] Y. Wu, L. Li, C. Tian, and C. Xu, "Breaking the memory wall for heterogeneous federated learning with progressive training," arXiv preprint arXiv:2404.13349, 2024.

[29] X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, and J. E. Gonzalez, "Skipnet: Learning dynamic routing in convolutional networks," in Proceedings of the European conference on computer vision (ECCV), 2018, pp. 409–424.

[30] Y. Guo, H. Shi, A. Kumar, K. Grauman, T. Rosing, and R. Feris, "Spottune: transfer learning through adaptive fine-tuning," in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2019, pp. 4805–4814.

[31] A. Veit and S. Belongie, "Convolutional networks with adaptive inference graphs," in Proceedings of the European conference on computer vision (ECCV), 2018, pp. 3–18.

[32] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," arXiv preprint arXiv:1611.01144, 2016.

[33] Z. Wu, T. Nagarajan, A. Kumar, S. Rennie, L. S. Davis, K. Grauman, and R. Feris, "Blockdrop: Dynamic inference paths in residual networks," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 8817–8826.

[34] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in 2016 23rd international conference on pattern recognition (ICPR). IEEE, 2016, pp. 2464–2469.

[35] J. Zhang, Y. Hua, H. Wang, T. Song, Z. Xue, R. Ma, and H. Guan, "Fedcp: Separating feature information for personalized federated learning via conditional policy," in Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2023, pp. 3249–3261.

[36] K. Yin, X. Ji, Y. Wang, and Z. Wang, "Fedclcc: A personalized federated learning algorithm for edge cloud collaboration based on contrastive learning and conditional computing," Defence Technology, 2024.

[37] Q. Yang, J. Zhang, W. Hao, G. P. Spell, and L. Carin, "Flop: Federated learning on medical datasets using partial networks," in Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, 2021, pp. 3845–3853.

[38] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," Advances in neural information processing systems, vol. 25, 2012.

[39] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," IEEE Transactions on Wireless Communications, vol. 16, no. 3, pp. 1397–1411, 2016.

[40] S. Lie, "Cerebras architecture deep dive: First look inside the hardware/software co-design for deep learning," IEEE Micro, vol. 43, no. 3, pp. 18–30, 2023.

[41] H. Mostafa, B. Pedroni, S. Sheik, and G. Cauwenberghs, "Hardware-efficient on-line learning through pipelined truncated-error backpropagation in binary-state networks," Frontiers in neuroscience, vol. 11, p. 496, 2017.

[42] M. Wang, S. Lu, D. Zhu, J. Lin, and Z. Wang, "A high-speed and low-complexity architecture for softmax function in deep learning," in 2018

IEEE asia pacific conference on circuits and systems (APCCAS). IEEE, 2018, pp. 223–226.
[43] S. S. Basha, S. R. Dubey, V. Pulabaigari, and S. Mukherjee, "Impact of fully connected layers on performance of convolutional neural networks for image classification," Neurocomputing, vol. 378, pp. 112–119, 2020.
[44] C. Z. Basha, B. L. Pravallika, and E. B. Shankar, "An efficient face mask detector with pytorch and deep learning," EAI Endorsed Transactions on Pervasive Health and Technology, vol. 7, no. 25, pp. e4–e4, 2021.
[45] A. Mao, M. Mohri, and Y. Zhong, "Cross-entropy loss functions: Theoretical analysis and applications," in International conference on Machine learning. PMLR, 2023, pp. 23 803–23 828.
[46] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in Artificial intelligence and statistics. PMLR, 2017, pp. 1273–1282.
[47] A. Krizhevsky and H. Geoffrey, "Learning Multiple Layers of Features From Tiny Images," Technical Report, 2009.
[48] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "A Downsampled Variant of Imagenet as an Alternative to the Cifar Datasets," arXiv preprint arXiv:1707.08819, 2017.
[49] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik, "Emnist: Extending mnist to handwritten letters," in 2017 international joint conference on neural networks (IJCNN). IEEE, 2017, pp. 2921–2926.
[50] T. Lin, L. Kong, S. U. Stich, and M. Jaggi, "Ensemble distillation for robust model fusion in federated learning," Advances in neural information processing systems, vol. 33, pp. 2351–2363, 2020.
[51] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár, "Designing network design spaces," in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2020, pp. 10 428–10 436.
[52] N. S. Keskar and R. Socher, "Improving generalization performance by switching from adam to sgd," arXiv preprint arXiv:1712.07628, 2017.
[53] J. Zhang, Y. Liu, Y. Hua, H. Wang, T. Song, Z. Xue, R. Ma, and J. Cao, "Pfllib: A beginner-friendly and comprehensive personalized federated learning library and benchmark," Journal of Machine Learning Research, vol. 26, no. 50, pp. 1–10, 2025.

Qiqi Cai is currently a Ph.D student in the Department of Computer Science and Engineering, Shanghai Jiao Tong University. Her research interests include distributed AI, machine learning operations, and federated learning.

Jian Cao is currently a tenured professor with the Department of Computer Science and Engineering, Shanghai Jiao Tong University. He is also the director of research institute of network computing and service computing. Dr. Cao's research interests include intelligent data analytics, service computing and distributed AI. He has published more than 300 research papers in prestigious journals and conferences. Currently, he is the distinguished member of CCF and the senior member of IEEE.

Jianqing Zhang is presently a PhD student in the Department of Computer Science and Engineering at Shanghai Jiao Tong University. He is the founder and primary contributor of the well-known open-source project PFLlib[1]. His research interests encompass synthetic data generation, collaboration between large and small models, and federated learning. He has had six papers accepted by top-tier conferences.

Wei Guan is currently a Ph.D student in the Department of Computer Science and Engineering, Shanghai Jiao Tong University. His research interests including predictive monitoring, anomaly detection and machine learning.

Shiyou Qian received a PhD degree in computer science from Shanghai Jiao Tong University, China, in 2015. He is currently an associate researcher with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. His research interests include event matching for content-based publish/subscribe systems, resource scheduling for Hybrid-Cloud, and driving recommendation with vehicular networks.

Rajkumar Buyya is a Redmond Barry Distinguished Professor and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He has authored over 850 publications and seven text books including "Mastering Cloud Computing" published by McGraw Hill, China Machine Press, and Morgan Kaufmann for Indian, Chinese and international markets respectively. He served as the founding Editor-in-Chief of the IEEE Transactions on Cloud Computing. He is currently serving as Co-Editor-in-Chief of Journal of Software: Practice and Experience, which was established 50+ years ago. For further information on Dr.Buyya, please visit his cyberhome: www.buyya.com.

[1]https://github.com/TsingZ0/PFLlib