CaBaFL: Asynchronous Federated Learning via Hierarchical Cache and Feature Balance

Zeke Xia, *Graduate Student Member, IEEE*, Ming Hu, *Member, IEEE*, Dengke Yan, Xiaofei Xie, *Member, IEEE*, Tianlin Li, *Member, IEEE*, Anran Li, *Member, IEEE*, Junlong Zhou, *Member, IEEE*, and Mingsong Chen, *Senior Member, IEEE*

***Abstract*—Federated learning (FL) as a promising distributed machine learning paradigm has been widely adopted in Artificial Intelligence of Things (AIoT) applications. However, the efficiency and inference capability of FL is seriously limited due to the presence of stragglers and data imbalance across massive AIoT devices, respectively. To address the above challenges, we present a novel asynchronous FL approach named CaBaFL, which includes a hierarchical cache-based aggregation mechanism and a feature balance-guided device selection strategy. CaBaFL maintains multiple intermediate models simultaneously for local training. The hierarchical cache-based aggregation mechanism enables each intermediate model to be trained on multiple devices to align the training time and mitigate the straggler issue. In specific, each intermediate model is stored in a low-level cache for local training and when it is trained by sufficient local devices, it will be stored in a high-level cache for aggregation. To address the problem of imbalanced data, the feature balance-guided device selection strategy in CaBaFL adopts the activation distribution as a metric, which enables each intermediate model to be trained across devices with totally balanced data distributions before aggregation. Experimental results show that compared to the state-of-the-art FL methods, CaBaFL achieves up to 9.26X training acceleration and 19.71% accuracy improvements.**

***Index Terms*—Artificial Intelligence of Things (AIoT), asynchronous federated learning (FL), data/device heterogeneity, feature balance.**

I. Introduction

With the prosperity of artificial intelligence (AI) and the Internet of Things (IoT), AI of Things (AIoT) is becoming the mainstream paradigm for the design of large-scale distributed systems [1], [2], [3], [4]. Federated learning (FL) [5], [6], [7], [8], [9], [10] as an important distributed machine learning paradigm has been widely used in AIoT-based applications, e.g., mobile edge computing [11], healthcare systems [12], and autonomous driving [13]. Typically, AIoT-based FL consists of a central server and a set of AIoT devices. The cloud server maintains a global model and dispatches it to multiple AIoT devices for training. Each AIoT device trains its received global model using its local data and then uploads the local model to the server. By aggregating all the local models, the server can achieve collaboratively global model training without leaking the raw data of any devices.

However, due to the heterogeneity of devices and data, AIoT-based FL still encounters two main challenges. The first challenge is the straggler problem. The heterogeneous nature of AIoT devices (e.g., varying computing and wireless network communication capacities), can result in significant differences in the training time for each device [14], [15]. Aggregating all the local models, including those from devices with poor computation capabilities, can lead to longer training time. The second challenge is that the data among AIoT devices are not independent-and-identically distributed (non-IID) [16]. Such a data imbalance issue among AIoT devices can lead to the problem of "weight divergence" [17] and results in the inference accuracy degradation of the global model [18], [19]. To address the above challenges, the existing solutions can be mainly classified into three schemes, i.e., synchronous [20], [21], [22], asynchronous [23], [24], and semi-asynchronous [25], [26], [27]. In synchronous FL methods [5], the cloud server generates the global model after receiving all the local models. The non-IID problem could be alleviated with some well-designed training and client selection strategies [20], [28], [29], while the straggler problems cannot be well addressed. Asynchronous FL methods [23] directly aggregate the uploaded local model to update the global model without waiting for other local models. By a timeout strategy, asynchronous FL methods could discard stragglers, thereby avoiding the inefficient update in the global model. However, non-IID scenarios still seriously limit the performance of existing asynchronous FL methods. Semi-asynchronous FL methods [25], [26] maintain a buffer to store uploaded local models, when the stored models reach a certain number, the server performs an aggregation operation and

Manuscript received 11 August 2024; accepted 12 August 2024. This work was supported in part by the Natural Science Foundation of China under Grant 62272170; in part by the "Digital Silk Road" Shanghai International Joint Lab of Trustworthy Intelligent Software under Grant 22510750100; in part by the National Research Foundation, Singapore; and in part by the Cyber Security Agency through its National Cybersecurity Research and Development Programme under Grant NCRP25-P04-TAICeN. This article was presented at the International Conference on Embedded Software (EMSOFT) 2024 and appeared as part of the ESWEEK-TCAD special issue. This article was recommended by Associate Editor S. Dailey. *(Corresponding author: Mingsong Chen.)*

Zeke Xia, Dengke Yan, and Mingsong Chen are with the MoE Engineering Research Center of HW/SW Co-Design Technology and Application, East China Normal University, Shanghai, China (e-mail: mschen@sei.ecnu.edu.cn).

Ming Hu and Xiaofei Xie are with the School of Computing and Information Systems, Singapore Management University, Singapore.

Tianlin Li is with the College of Computing and Data Science, Nanyang Technological University, Singapore.

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

Junlong Zhou is with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China.

Digital Object Identifier 10.1109/TCAD.2024.3446881

clears the buffer. Although semi-asynchronous FL methods can alleviate the straggler problems, they still encounter the problems of non-IID data. Due to adopting different training mechanisms, synchronous methods are often difficult to combine directly with asynchronous methods. Therefore, how to ensure performance in scenarios of data imbalance while solving the straggler problem is a serious challenge.

To address both the straggler and data imbalance challenges, this article presents a novel asynchronous FL approach named CaBaFL. It maintains a hierarchical cache structure to allow each intermediate model to be asynchronously trained by multiple clients before aggregation and uses a feature balance-guided client selection strategy to enable each model to eventually be trained by totally balanced data. To address the challenge of stragglers, we use a hierarchical cache-based aggregation mechanism to achieve asynchronous training. Specifically, we use multiple intermediate models for local training and to guarantee the number of activated devices, we enable each intermediate model participant in the aggregation after multiple times of local training. To facilitate the asynchronous aggregation, each intermediate model is stored in the low-level cache named L2 cache. When trained with sufficient devices, a model can be stored in the high-level cache named L1 cache. While a certain number of devices trains a model, it will be updated with the global model aggregated with all the models in the L1 cache. Based on our asynchronous mechanism, the feature balance-guided device selection strategy wisely selects a device for each intermediate model, aiming to make the total data used to train the model balanced before aggregation. However, gaining direct access to the data distribution of each device could potentially compromise users' privacy. To address this concern, we are inspired by the observation that the middle-layer features strongly reflect the underlying data distributions and propose a method to select devices based on their middle-layer activation patterns. This article has three major contributions.

1) We propose a novel asynchronous FL framework named CaBaFL, which enables multiple intermediate models for collaborative training asynchronously using a hierarchical cache-based aggregation mechanism.
2) We present a feature balance-guided device selection strategy to wisely select devices according to the activation distribution to make each intermediate model be trained with totally balanced data, which alleviates the performance deterioration caused by data imbalance.
3) We conduct comprehensive experiments on both well-known datasets and models to show the superiority of CaBaFL over state-of-the-art (SOTA) FL methods for both IID and non-IID scenarios.

II. Preliminary and Related Work

A. Preliminary of Federated Learning

In general, an FL system consists of one cloud server and multiple dispersed clients. In each round of training in FL, the cloud server first selects a subset of devices to distribute the global model. After receiving the model, the devices conduct local training and upload the model to the cloud server. Finally, the cloud server aggregates the received models and obtains a new global model. The learning objective of FL is to minimize the loss function over the collection of training data at N clients, i.e.,

$$\min_{w} F(w) = \sum_{k=1}^{N} \frac{|D_k|}{|D|} F_k(w) \tag{1}$$

where N is the number of clients that participate in local training, w is the global model parameters, D_k is the k^{th} client, $|D_k|$ represents the training data size on D_k, and $F_k(w) = (1/|D_k|)\sum_{j \in D_k} f_j(w)$ is the loss empirical objective over the data samples at client k.

B. Related Work

Asynchronous FL has a natural advantage in solving the straggler effect, where the server can aggregate without waiting for stragglers. Xie et al. [23] developed a FedAsync algorithm, which combines a function of staleness with asynchronous update protocol. In FedASync, whenever a model is uploaded, the server directly aggregates it. Although FedASync can solve the straggler problem, some stragglers may become stale models, thereby reducing the accuracy of the global model. In addition, the client in FedASync will send a large number of models to the server, causing a lot of communication overhead. In terms of reducing data transmission, Wu et al. [25] proposed a SAFA protocol, in which asynchronous clients continuously perform local updates until the difference between the local update version and the global model version reaches tolerance. Although SAFA considers model staleness, the server needs to wait for the asynchronous clients. Moreover, SAFA needs to maintain a bigger buffer compared to FL, which can cause more memory costs and thus lead to high complexity and low scalability. Similarly, Ma et al. [26] set a model buffer for model aggregation to achieve semi-asynchronous FL and dynamically adjust the learning rate and local training epochs to mitigate the impact of stragglers and data heterogeneity. However, none of the above methods can solve the problem of data heterogeneity well.

To address the data heterogeneity problem, Zhou et al. [30] proposed the WKAFL protocol, which leverages the stale models of stragglers by maintaining a globally unbiased gradient and mitigates the impact of data heterogeneity through gradient clipping. Hu et al. [31] used the semi-asynchronous FL mechanism, which maintains a buffer in the cloud server to store the local models. The server attempts to alleviate the impact of data heterogeneity by minimizing the variance of hard labels in the buffer. However, this method requires the clients to send the hard labels to the server. Unfortunately, in real-world scenarios, hard labels of data often contain sensitive information and cannot be obtained by the server. As an alternative, FedAC [32] employs a momentum aggregation strategy for updating the global model and incorporates fine-grained correction to adjust client gradients, effectively mitigating the challenges posed by data heterogeneity. FedLC [33] deals with the non-IID problem by enabling local collaboration among edge devices and solves the stale model problem through

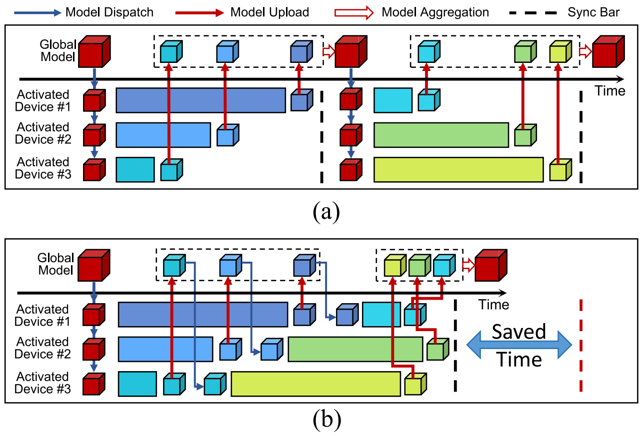


Fig. 1. Motivating example of our asynchronous FL method. (a) Conventional synchronous FL. (b) Intuition of our asynchronous FL.

dynamic learning rates. However, these methods optimize the aggregation strategy without using a wise device selection strategy, which still seriously limits their performance.

To the best of our knowledge, CaBaFL is the first attempt to employ collaborative training and feature balance-guided device selection strategy in asynchronous FL to improve both model accuracy and training stability.

III. Motivation

A. *Intuition of Our Asynchronous FL Mechanism*

Fig. 1 presents the intuition of our asynchronous FL mechanism. Assume that one FL training involves the local training of three models. Fig. 1(a) presents a training snapshot of some conventional synchronous FL methods, where local models are aggregated after they finish training on one activated device. In this case, the FL method performs two aggregation operations without taking the straggler problem into account. From this subfigure, we can clearly observe that some models become idle in between two aggregations. Unlike conventional FL methods, our approach always pushes models to conduct local training synchronously in a balanced way. Specifically, our approach considers the straggler problem and allows a model to be trained on multiple devices before an aggregation operation. When an activated device finishes its local training, it will immediately forward its hosting model to another device with a balanced training overhead. As an example shown in Fig. 1(b), a model can traverse two devices before one aggregation operation, and all three models have similar total training time spent on their two traversed devices. In this way, the straggler problem can be mitigated, since a compact training scheme can not only enable stragglers to be chosen for local training more often in a fair manner but also accelerate the training convergence processes.

B. *Correlation Between Data and Activation Distributions*

To mitigate the challenge posed by data imbalance, our objective is to carefully choose devices for each intermediate model and ensure that the total data used for training the model is balanced before aggregation. Accomplishing this task necessitates having knowledge of the data distribution associated with each device. Due to the risk of privacy leakage, it is difficult for the server to obtain the data distribution of each device directly. Therefore, selecting an easily obtainable metric that does not compromise privacy to guide device selection, which ensures that each intermediate model is trained by balanced data is a key challenge for our asynchronous FL approach in dealing with the non-IID problem. We have made the following important observations that demonstrate the ability of middle-layer activation patterns to reflect the input data distribution of each device. Moreover, we have found that the activation distributions (i.e., feature distributions) of some model middle-layer can provide a finer-grained representation of input data distributions than the one relying solely on input data labels. These observations motivate us to select devices based on middle-layer activation distributions.

Observation 1: To explore the connection between the activation distribution and the data distribution, we divide CIFAR-10 into six subdatasets (i.e., D_0–D_5), in which the data in D_0 is balanced, i.e., IID, and the other five data are divided according to the Diricht distribution [34] $Dir(\beta)$, where a small value of β indicates a more seriously data imbalance among subdatasets. We select ResNet-18 for model training and computing the activation distribution of a specific layer on the whole CIFAR-10 dataset and six subdatasets, respectively.

Fig. 2(a)–(c) present the cosine similarities of the activation distribution using the whole CIFAR-10 dataset with that using six subdatasets, with $\beta = 0.1, 0.5$, and 1.0, respectively. We can observe that the D_0 achieves the highest-cosine similarity and the subdatasets divided with a smaller β achieves lower-cosine similarity. Therefore, if the data distribution of a subdataset is more similar to that of the whole dataset, it achieves a higher-cosine similarity of their activation distribution. Fig. 2(d)–(f) present the cosine similarities of the activation distribution using the CIFAR-10 dataset with that using the combinations of five imbalance subdatasets. Note that since the data in CIFAR-10 dataset and D_0 is balanced, data of the combination of all the five imbalance subdatasets is balanced. From Fig. 2(d)–(f), we can observe that with the data distribution of the combination dataset close to the whole dataset, the cosine similarity of their activation distribution becomes higher. Because the activation amount is obtained by counting the number of times the neuron is activated, the activation amount of the combination dataset is equal to the sum of that of all the combined datasets.

Observation 2: Due to the preference difference of devices, even if they have the same data label, the features of their data may be different. For example, for the “Dog” category, some users prefer Husky dogs, and some users prefer Teddy dogs. Therefore, the data category distribution sometimes ignores the differences between the same category data.

To explore whether differences between the same category data can lead to different activation distributions, we select the CIFAR-100 dataset, which contains 100 fine-grained and 20 coarse-grained classifications. We divide the CIFAR-100 dataset into six subdatasets (i.e., D_0–D_5), in which the data in D_0 is balanced on fine-grained classifications and the other five data are balanced on course-gained classifications but imbalanced on fine-grained classifications. We conduct model

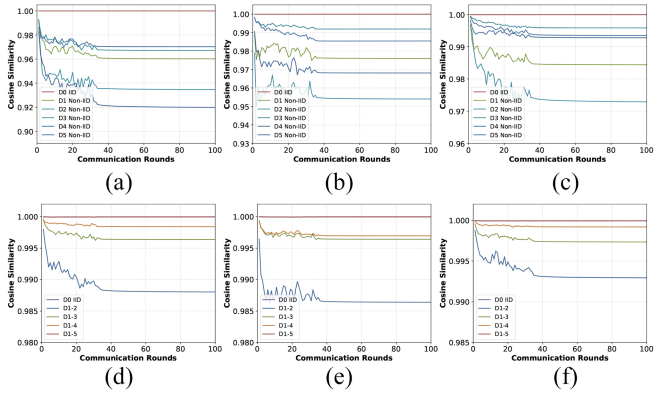


Fig. 2. Cosine similarity comparison of activation distributions. (a) $\beta = 0.1$. (b) $\beta = 0.5$. (c) $\beta = 1.0$. (d) Combination for (a). (e) Combination for (b). (f) Combination for (c).

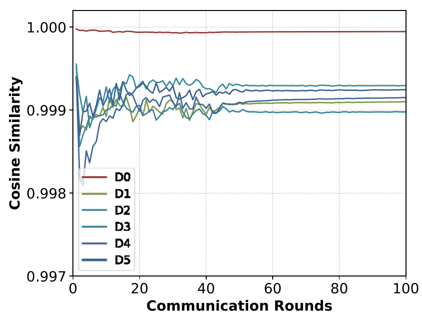


Fig. 3. Cosine similarity comparison on CIFAR-100 dataset.

training on the task of the coarse-grained classification and computing the activation distribution of a specific layer on the whole CIFAR-100 dataset and six subdatasets, respectively.

Fig. 3 presents the cosine similarities of the activation distribution using CIFAR-100 with six subdatasets. We can observe that D_0 achieves the highest-cosine similarity and the difference of data with the same category can decrease the similarity of activation distribution. Therefore, compared to the data distribution, the activation distribution can express the differences between data in a more fine-grained manner.

The above observations indicate that the higher the cosine similarity between the activation distributions of training data and the global activation distribution, the more balanced the training data are. Therefore, selecting a device that enables high-cosine similarity can lead the training data distribution to be IID. In our approach, we use activation distributions as a metric to guide the server to select devices, ensuring that each intermediate model is trained using more balanced data.

IV. Our CaBaFL Approach

A. Overview

Fig. 4 illustrates the framework and workflow of our CaBaFL approach, which consists of a central cloud server and multiple AIoT devices, i.e., $D_1 - D_N$, where the cloud server includes two crucial components, i.e., a hierarchical cache-based model aggregation controller and a feature balance-guided device selector, respectively. The hierarchical cache-based model aggregation controller performs the model aggregation and updating according to the number of training times. The feature balance-guided device selector chooses clients for local training based on the cosine similarity between the model's feature in the L2 cache and the global feature. Note that, according to the observations in Section III, we select the activation of a specific layer as the metric to calculate the feature. Assume that l is a layer of some model m. From the perspective of m, the feature distribution of a device is the activation distribution of l using raw device data as inputs of m. The feature distribution of an intermediate model indicates the accumulative feature distributions of l on all its traversed devices since the last aggregation. For example, assume that an intermediate model m_k is continuously trained by devices D_i and D_j, its feature distribution can be calculated as $f_{m_k} = f_{D_i} + f_{D_j}$, where f_{D_i} and f_{D_j} are the feature distribution on D_i and D_j, respectively. Note that, the global feature distribution is the sum of all the device feature distributions. Assume there are N devices, the global feature can be calculated with $f_g = \sum_{i=1}^{N} f_{D_i}$.

Inspired by the concept of cache in the computer architecture domain, we developed a novel two-level cache-like data structure that is hosted in the memory. In CaBaFL, the hierarchical cache-based model aggregation controller consists of a 2-level cache data structure to store intermediate models and a cache update controller to control model updating in L1 cache. The feature balance-guided device selector consists of a model feature distribution cache and a device feature distribution cache, where the model feature distribution cache records the feature distribution of each intermediate model and the device feature distribution cache records the global feature distribution and feature distributions of each device. When an intermediate model completes a local training session, the device selector updates its feature distribution by adding the feature distribution of its latest dispatched device to its original feature distribution. In addition, CaBaFL periodically broadcasts the global model to all devices to collect their feature distributions. Upon receiving the global model, each device calculates the device feature distribution using its raw data. After the feature distribution collection, the server updates the device feature distribution cache and model feature distribution cache. Note that the feature distribution collection process operates asynchronously with the FL training process. As shown in Fig. 4, the FL training process for each intermediate model in CaBaFL includes five steps as follows.

1) *Step 1 (Device Selection):* For an intermediate model, the device selector selects a device according to the feature distribution and training time of the intermediate model and the feature distributions of candidate devices.
2) *Step 2 (Model Dispatching):* The server dispatches intermediate models to selected devices for local training.
3) *Step 3 (Model Uploading):* The device uploads the model to the cloud server after local training is completed.
4) *Step 4 (Cache Update):* The cache update controller stores the received model in the L2 cache and decides if it updates the model to the L1 cache according to its training times and features.
5) *Step 5 (Cache Aggregation):* If an intermediate model reaches the specified training times, the cloud server

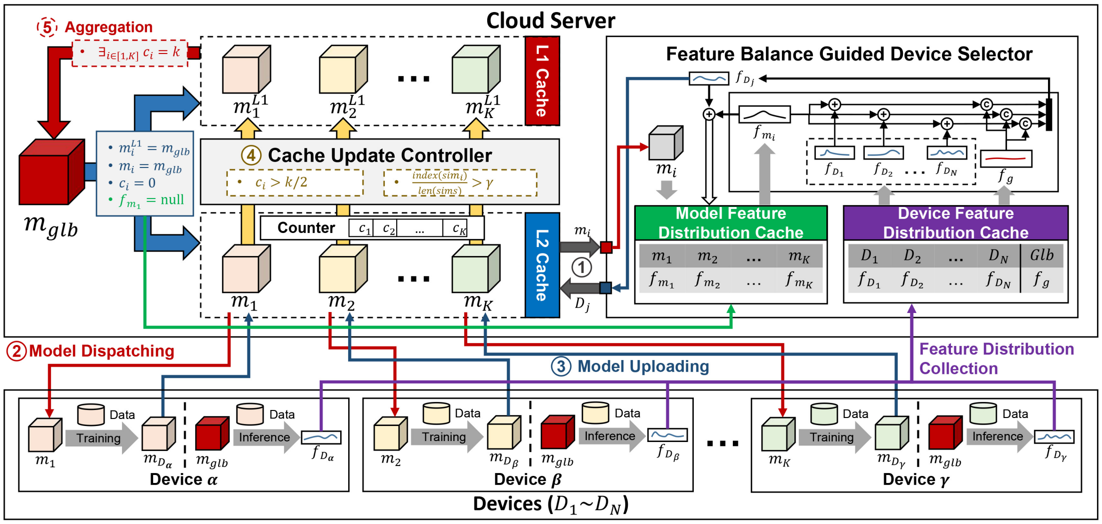


Fig. 4. Framework and workflow of CaBaFL.

aggregates all the models in the L1 cache to generate a new global model and update the intermediate model using the global model.

Our approach repeats steps 1–5 until a given time threshold is reached. Throughout the training process, none of the intermediate models need to wait for other intermediate models, i.e., all the intermediate models are trained asynchronously.

Algorithm 1 details the implementation of our CaBaFL approach. Assume that there are at most K activated clients participating in local training at the same time. Line 1 initializes the 2-level caches and other parameters we need. Note that the size of the L1 and L2 cache is K intermediate models. Line 2 initializes the vector *sims*. Lines 3–28 present the FL training process of models in the 2-level cache, where the "for" loop is a parallel loop. Line 6 represents that the model m_i is uploaded to the server after training on the device D_j. Line 7 represents the training times of m_i plus one. Line 8 represents the server storing the model m_i in the L2 cache. In line 9, the server calculates the cosine similarity sim_i between the model feature distribution f_{m_i} and the global feature distribution f_g as sim_i. After that, the server adds sim_i to *sims* and sorts *sims* (lines 10 and 11). In lines 12–16, the server updates m_i^{L1} using the eligible m_i. Line 13 represents that the server updates the model in the L2 cache to the corresponding L1 cache. In lines 15 and 16, the server updates the feature distribution $f_{m_i^{L1}}$ and the training data size for m_i^{L1} after the L1 cache is updated. Lines 18–23 present the details of the model aggregation process. In line 18, model aggregation is triggered when c_i, the training times of m_i, achieves k. In line 19, the server aggregates models in the L1 cache to obtain a new global model m_{glb}. In lines 20–23, the server replaces m_i and $cache[1][i-1]$ with the m_{glb} and resets the training times and model feature of m_i. In line 25, *DevSel*() selects a device D_j for m_i for model dispatching.

B. CaBaFL Implementation

1) Hierarchical Cache-Based Asynchronous Aggregation: *Hierarchical Cache Updating:* CaBaFL maintains a 2-level cache in the cloud server to screen models in the L2 cache.

Algorithm 1 Implementation of CaBaFL

Input: i) f_g, global feature distribution; ii) f_D, feature distributions of devices; iii) f_m, feature distributions of models; iv) T, time threshold.
Output: m_{glb}, the global model;
CaBaFL(f_g,f_D,f_m,T)

1: $Init(cache, c, DS, DS^{L1}, S)$;
2: $VecInit(sims)$;
3: **while** time threshold T is not reached **do**
4: /* Parallel for */
5: **for** $i \leftarrow 1, \ldots, K$ **do**
6: $m_i \leftarrow ReceiveModel(i)$;
7: $c_i \leftarrow c_i + 1$;
8: $cache[0][i-1] \leftarrow m_i$;
9: $sim_i \leftarrow cossim(f_g, f_{m_i})$;
10: $add(sims, sim_i)$;
11: $sort(sims)$;
12: **if** $c_i > \frac{k}{2}$ or $\frac{index(sim_i)}{len(sims)} > \gamma$ **then**
13: $cache[1][i-1] \leftarrow cache[0][i-1]$;
14: $m_i^{L1} \leftarrow cache[1][i-1]$;
15: $f_{m_i^{L1}} \leftarrow f_{m_i}$;
16: $DS_i^{L1} \leftarrow DS_i$;
17: **end if**
18: **if** $c_i = k$ **then**
19: $m_{glb} \leftarrow Aggr(m^{L1}, DS^{L1})$;
20: $cache[1][i-1] \leftarrow m_{glb}$;
21: $m_i \leftarrow m_{glb}$;
22: $c_i \leftarrow 0$;
23: $f_{m_i} \leftarrow null$;
24: **end if**
25: $D_j \leftarrow DevSel(f_g, f_D, DS, S, c_i, f_{m_i})$;
26: **end for**
27: **end while**
28: **return** m_{glb};

When a model completes local training and is uploaded to the cloud server, the cloud server first saves the model in the L2 cache. If the model meets certain conditions, the server updates the model to the L1 cache. The server calculate the cosine similarity sim_i between the model feature distribution f_{m_i} and the global feature distribution f_g as $sim_i = cossim(f_g, f_{m_i}) = [(f_g \cdot f_{m_i})/(\|f_g\| \|f_{m_i}\|)]$. If sim_i is low, aggregating these models hurts the performance of the global model. To perform effective model screening, the cloud server maintains a sorted list *sims*, which records the similarity between the features of all models trained and f_g. The server adds sim_i to *sims*.

m_i^{L1} will be updated to m_i, if $([index(sim_i)]/[len(sims)]) > \gamma$, where $index(sim_i)$ represents the index of sim_i in $sims$ and $len(sims)$ represents the length of $sims$. In addition, if the number of model training times exceeds half of the specified times, i.e., $c_i > (k/2)$, the cloud server will also update m_i^{L1} to m_i because the model has learned enough knowledge to participate in model aggregation.

Model Aggregation Strategy ($Aggr(\cdot)$): In cache aggregation, our approach calculates a weight for the model in the L1 cache based on the training data size and the similarity between the model features and global features. The aggregation process is formulated as follows:

$$Aggregation\left(m^{L1}, DS^{L1}\right) = \frac{\sum_{i=1}^{K} m_i^{L1} \times \frac{(DS_i^{L1})^{\alpha}}{1-CS_i}}{\sum_{i=1}^{K} \frac{(DS_i^{L1})^{\alpha}}{1-CS_i}} \tag{2}$$

where m_i^{L1} represents the model in L1 cache, DS_i^{L1} represents the size of training data of m_i^{L1}, $CS_i = cossim(f_{m_i^{L1}}, f_g)$ represents the similarities between the feature distribution of the model in L1 cache $f_{m_i^{L1}}$ and the global feature distribution f_g, K represents the size of the cache, and α is a hyperparameter that is less than 1.

To calculate the weight, we first calculate the model weights based on the model's training data size. Due to the possibility of different numbers of training times of the model in the L1 cache, there may be significant differences in the training data size of the model. Therefore, if we directly use the training data size as the weight, it may harm the performance of the global model when data heterogeneity is high. Therefore, we use the hyperparameter α to mitigate this effect. After that, we calculate the weights based on the similarity $CS_i = cossim(f_g, f_{m_i^{L1}})$. The higher the CS_i, the more balanced the model feature distribution is, and the higher the weight is assigned to the model. Since the similarity between models is very close to 1, to achieve small differences between them, we use $1 - CS_i$ to amplify these differences.

2) Feature Balance Guided Device Selection: Algorithm 2 presents the device selection strategy of CaBaFL. When selecting devices, greedily choosing the device with the greatest similarity between model features and global features can lead to fairness issues by causing some devices to be selected repeatedly while others are rarely selected. Therefore, CaBaFL sets a hyperparameter σ. When the variance of the number of times devices are selected exceeds σ, the server selects devices from those idle devices with the fewest selections; otherwise, the selection range is all idle devices (lines 1–4). Note that we normalize before calculating the variance. If the model has just started a new training round, i.e., $c_i = 0$, the server randomly selects a device for the model to start a new training round (lines 6–9). Otherwise, for each device D_j, CaBaFL calculates the cosine similarity w_1 between the model feature distribution when D_j is selected and the global feature distribution (line 11). CaBaFL also considers balancing the total training time of the models in the L2 cache when selecting devices. Since the training time of a device is often directly proportional to the size of its data, balancing the model training data size can indirectly balance training time between models in the L2 cache. More balanced training times between models can alleviate the stale model problem caused by the stragglers and allow for faster training. Therefore, CaBaFL calculates the variance of the training data size of the model in the L2 cache when D_j is selected (lines 12–14), where $|D_j|$ represents the training data size on D_j. w_1 is then subtracted by the variance to obtain the weight of the device (line 15), and the device with the highest weight is selected for model dispatching (line 17). Finally, lines 18–20 update S_{res}, f_{m_i} and DS_i.

Algorithm 2 Device Selection Procedure

Input: i) f_g, global feature distribution; ii) f_D, feature distributions of devices; iii) DS, data size of L2 cache; iv) S, selected times of devices; v) c_i, training times of m_i; vi) f_{m_i}, feature distribution of m_i.
Output: D_j, selected device.
DevSel(f_g,f_D,DS,S,c_i,f_{m_i})

1: $sr \leftarrow GetIdleDevice()$;
2: $S^{idle} \leftarrow \{S_{D_i} \mid D_i \in sr\}$
3: **if** $\mathrm{var}(S) > \sigma$ **then**
4:     $sr \leftarrow \{D_j \mid S_{D_j}^{idle} = \min(S^{idle})\}$;
5: **end if**
6: **if** $c_i = 0$ **then**
7:     $D_j \leftarrow RandomlySelect(sr)$;
8:     **return** D_j
9: **end if**
10: **for** $D_j \in sr$ **do**
11:     $w_1 \leftarrow cossim(f_g, f_{m_i} + f_{D_j})$;
12:     $DS' \leftarrow DS$;
13:     $DS'_i \leftarrow DS'_i + |D_j|$;
14:     $w_2 \leftarrow \mathrm{var}(DS')$;
15:     $w_{D_j} \leftarrow w_1 - w_2$;
16: **end for**
17: $res \leftarrow \arg\max_{D_j} w_{D_j}$;
18: $S_{res} \leftarrow S_{res} + 1$;
19: $f_{m_i} \leftarrow f_{m_i} + f_{D_{res}}$;
20: $DS_i \leftarrow DS_i + |D_{res}|$;
21: **return** res

V. Performance Evaluation

A. Experimental Setup

To demonstrate the effectiveness of our approach, we implemented CaBaFL using the PyTorch framework [35]. All experimental results were obtained from a Ubuntu workstation equipped with an Intel i9 CPU, 64GB of memory, and an NVIDIA RTX 4090 GPU.

Settings of Baselines: We compared our approach with six baselines, including the classical FedAvg [5] and five SOTA FL methods (i.e., FedProx [20], FedAsync [23], SAFA [25], FedSA [26], and WKAFL [30]), which aim to solve similar problem. Note that FedAvg and FedProx are synchronous FL methods, FedAsync is an asynchronous FL method, and the other three are semi-asynchronous FL methods. For SAFA, FedSA, and WKAFL, we set the model buffer size to $K/2$. To ensure a fair comparison, we used an SGD optimizer with a learning rate of 0.01 and a momentum of 0.5 for all baselines and CaBaFL. Each client was trained locally for five epochs with a batch size of 50. Note that all these experimental settings are widely used in the evaluation of baselines.

Settings of Datasets and Models: Our experiments were conducted on three well-known datasets, i.e., CIFAR-10, CIFAR-100 [36], and FEMNIST [37], which are widely used in evaluating the performance of the above baselines. To create

TABLE I
COMPARISON OF TEST ACCURACY FOR BOTH IID AND NON-IID SCENARIOS

Dataset	Model	Heterogeneity Settings	Test Accuracy (%)						
			FedAvg [5]	FedProx [22]	FedASync [25]	FedSA [30]	SAFA [29]	WKAFL [34]	CaBaFL
CIFAR-10	ResNet-18	$\beta = 0.1$	45.65 ± 3.52	45.57 ± 1.42	47.34 ± 0.72	47.26 ± 1.61	42.20 ± 2.63	40.86 ± 5.98	$\mathbf{55.46 \pm 0.67}$
		$\beta = 0.5$	60.34 ± 0.37	58.66 ± 0.27	60.70 ± 0.28	60.59 ± 0.28	57.69 ± 0.71	67.84 ± 0.78	$\mathbf{69.31 \pm 0.18}$
		$\beta = 1.0$	65.79 ± 0.34	63.55 ± 0.11	66.61 ± 0.22	65.11 ± 0.27	61.92 ± 0.44	71.19 ± 0.49	$\mathbf{72.84 \pm 0.36}$
		IID	64.89 ± 0.17	63.78 ± 0.14	64.76 ± 0.10	64.98 ± 0.12	64.28 ± 0.16	73.50 ± 0.18	$\mathbf{74.83 \pm 0.07}$
	CNN	$\beta = 0.1$	46.58 ± 1.29	46.59 ± 1.06	48.90 ± 0.68	48.58 ± 1.92	42.66 ± 2.23	41.03 ± 3.91	$\mathbf{52.92 \pm 0.62}$
		$\beta = 0.5$	54.77 ± 0.60	54.33 ± 0.22	55.97 ± 0.66	55.44 ± 0.37	52.63 ± 1.30	50.25 ± 3.15	$\mathbf{57.89 \pm 0.59}$
		$\beta = 1.0$	55.88 ± 0.53	55.72 ± 0.28	56.00 ± 0.31	56.45 ± 0.28	54.42 ± 1.02	53.88 ± 1.78	$\mathbf{58.58 \pm 0.72}$
		IID	57.87 ± 0.19	57.76 ± 0.21	57.82 ± 0.09	57.94 ± 0.14	55.88 ± 0.17	58.15 ± 0.38	$\mathbf{61.75 \pm 0.30}$
	VGG-16	$\beta = 0.1$	62.67 ± 5.15	63.31 ± 3.28	66.26 ± 1.24	65.26 ± 2.13	56.37 ± 3.95	55.09 ± 3.92	$\mathbf{70.68 \pm 1.49}$
		$\beta = 0.5$	77.94 ± 0.46	77.15 ± 0.09	78.26 ± 0.20	78.02 ± 0.26	75.40 ± 0.87	78.78 ± 0.20	$\mathbf{82.36 \pm 0.25}$
		$\beta = 1.0$	79.45 ± 0.41	78.58 ± 0.14	79.69 ± 0.17	78.64 ± 0.33	77.46 ± 0.75	80.19 ± 1.02	$\mathbf{83.81 \pm 0.27}$
		IID	80.35 ± 0.05	79.22 ± 0.08	80.30 ± 0.06	80.42 ± 0.04	79.25 ± 0.14	82.89 ± 0.24	$\mathbf{85.12 \pm 0.05}$
CIFAR-100	ResNet-18	$\beta = 0.1$	33.64 ± 0.52	33.24 ± 0.38	35.05 ± 0.41	33.61 ± 0.49	31.01 ± 1.04	27.65 ± 2.22	$\mathbf{37.77 \pm 0.35}$
		$\beta = 0.5$	41.31 ± 0.19	40.08 ± 0.18	42.75 ± 0.37	41.96 ± 0.20	39.44 ± 0.44	44.78 ± 1.48	$\mathbf{47.30 \pm 0.28}$
		$\beta = 1.0$	43.33 ± 0.19	41.76 ± 0.11	44.85 ± 0.23	42.75 ± 0.23	40.92 ± 0.23	47.41 ± 0.76	$\mathbf{48.10 \pm 0.44}$
		IID	42.76 ± 0.09	42.63 ± 0.15	42.20 ± 0.18	43.18 ± 0.15	42.07 ± 0.16	49.10 ± 0.20	$\mathbf{49.64 \pm 0.06}$
	CNN	$\beta = 0.1$	29.44 ± 0.65	30.64 ± 0.83	31.81 ± 0.57	30.52 ± 0.49	29.44 ± 1.44	19.18 ± 2.59	$\mathbf{33.22 \pm 0.48}$
		$\beta = 0.5$	34.08 ± 0.64	34.73 ± 0.45	32.99 ± 0.30	34.70 ± 0.49	33.98 ± 0.95	28.88 ± 2.25	$\mathbf{36.44 \pm 0.39}$
		$\beta = 1.0$	32.43 ± 0.48	32.74 ± 0.38	32.84 ± 0.27	34.27 ± 0.30	33.55 ± 0.35	31.35 ± 1.35	$\mathbf{37.31 \pm 0.49}$
		IID	33.04 ± 0.24	33.55 ± 0.20	32.60 ± 0.22	33.23 ± 0.15	31.73 ± 0.37	32.99 ± 0.68	$\mathbf{36.88 \pm 0.33}$
	VGG-16	$\beta = 0.1$	47.30 ± 1.27	47.29 ± 0.45	49.69 ± 0.68	46.91 ± 1.41	42.34 ± 1.66	30.82 ± 3.78	$\mathbf{50.53 \pm 0.49}$
		$\beta = 0.5$	54.83 ± 0.59	53.80 ± 0.59	56.39 ± 0.47	54.55 ± 0.40	50.61 ± 0.66	53.08 ± 2.40	$\mathbf{57.37 \pm 0.37}$
		$\beta = 1.0$	56.49 ± 0.21	54.50 ± 0.35	57.30 ± 0.35	55.24 ± 0.31	53.65 ± 0.40	56.71 ± 1.07	$\mathbf{59.53 \pm 0.17}$
		IID	57.56 ± 0.05	56.30 ± 0.23	57.91 ± 0.13	56.39 ± 0.14	55.33 ± 0.24	61.16 ± 0.38	$\mathbf{64.96 \pm 0.04}$
FEMNIST	ResNet-18	-	82.25 ± 0.43	81.57 ± 0.30	82.87 ± 0.35	82.07 ± 0.43	76.85 ± 0.35	74.68 ± 0.60	$\mathbf{84.08 \pm 0.17}$
	CNN	-	81.29 ± 0.38	81.57 ± 0.29	82.68 ± 0.23	81.99 ± 0.46	76.82 ± 0.38	73.81 ± 0.58	$\mathbf{84.00 \pm 0.10}$
	VGG-16	-	82.34 ± 0.38	81.38 ± 0.21	82.73 ± 0.26	82.00 ± 0.47	76.87 ± 0.39	74.62 ± 0.47	$\mathbf{84.30 \pm 0.13}$

the heterogeneity of device data for CIFAR-10 and CIFAR-100, we employed the Dirichlet distribution $Dir(\beta)$ [34], where smaller values of β indicate greater data heterogeneity. As FEMNIST already possesses a non-IID distribution, we did not require the use of the Dirichlet distribution. Moreover, to show the pervasiveness of our approach, we conducted experiments on three networks, i.e., CNN, ResNet-18, and VGG-16, which have different structures and depths.

Settings of System Heterogeneity: To simulate system heterogeneity, for the experiments on datasets CIFAR-10 and CIFAR-100, we assumed each of them involved 100 AIoT devices with varying computing power. However, for dataset FEMNIST, there are a total of 180 devices. First, we simulate different computing power based on the processing speed of true devices. Since the NVIDIA Jetson AGX Xavier device can be powered by different performance modes that can provide different computing power, we measured the processing speed under different performance modes and used it as the basis for simulating the computing power of our devices. To generate the simulated computing power, we used a Gaussian distribution with the mean and variance obtained from actual time consumption data of training models on Jetson AGX Xavier. Specifically, we assume that the training performance (i.e., the training time of one data sample) of a device follows the Gaussian distribution $N(0.03, 0.01)$, as measured in seconds. Second, we simulated a fixed network bandwidth for each device to calculate the communication time required with the cloud server. In addition, we assume only 10% of devices can participate in training at the same time.

B. Performance Comparison

1) Comparison of Accuracy: Table I compares the test accuracy between CaBaFL and all the baselines on three datasets with different non-IID and IID settings using ResNet-18, CNN, and VGG-16 models. From Table I, it can be observed that CaBaFL can achieve the highest-test accuracy in all the scenarios regardless of model type, dataset type, and data heterogeneity. For example, CaBaFL improves test accuracy by 19.71% over WKAFL in CIFAR-100 dataset with VGG-16 model when $\beta = 0.1$. Fig. 5 shows the learning curves of CaBaFL and all baseline methods on CIFAR-10 and ResNet-18. As an example of dataset CIFAR-10, when $\beta = 0.1$ and the target accuracy is 45%, CaBaFL outperforms SAFA by 9.26X in terms of training time. Moreover, We can observe that CaBaFL achieves the highest accuracy and exhibits good stability in its learning curve. Furthermore, we can find that our method can perform better than the baselines even in the IID scenario. This is mainly because in CaBaFL, a model can be trained on multiple devices before being aggregated, i.e., it can perform more times of stochastic gradient descent.

TABLE II
COMPARISON OF COMMUNICATION OVERHEAD

Heter. Settings	Acc. (%)	Communication Overhead						
		FedAvg	FedProx	FedASync	FedSA	SAFA	WKAFL	CaBaFL
$\beta = 0.1$	40	1620	1620	**600**	1510	4700	7930	1360
	45	2300	2300	4220	2380	14640	7940	**2120**
$\beta = 0.5$	58	2760	2760	3160	3160	9550	2680	**1811**
	60	7860	NA	11060	11040	NA	3380	**2259**
$\beta = 1.0$	63	2920	5200	2580	4560	NA	2960	**2085**
	65	7480	NA	5760	10270	NA	3270	**2279**
IID	63	1200	1220	**940**	1770	5800	1580	1593
	70	NA	NA	NA	NA	NA	4010	**2441**

2) Comparison of Communication Overhead: To evaluate the communication overhead caused by CaBaFL, we conducted experiments on the CIFAR-10 dataset using the ResNet-18 model. Table II compares the communication overheads between CaBaFL and the baselines to achieve specified inference accuracy for the global model. We can find CaBaFL leads to the least communication overhead within six out of the eight cases. For example, when the target accuracy is 65% and $\beta = 1.0$, the communication overhead of FedASync is

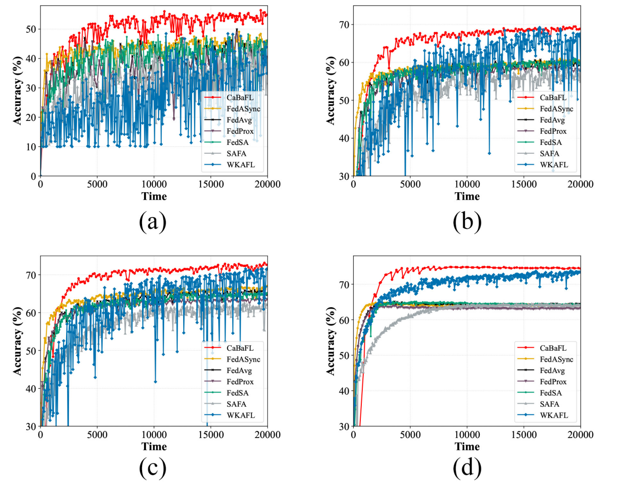


Fig. 5. Learning curves of training CIFAR-10 with ResNet-18. (a) $\beta = 0.1$. (b) $\beta = 0.5$. (c) $\beta = 1.0$. (d) IID.

TABLE III
STABILITY COMPARISON USING THE STANDARD DEVIATION OF MA

FedAvg	FedProx	FedASync	FedSA	SAFA	WKAFL	CaBaFL
0.72	0.41	0.38	0.56	1.47	4.13	**0.37**

5760, while our approach is 2279. Note that both FedProx and SAFA cannot achieve the target in this case.

3) Comparison of Stability and Fairness: We conducted experiments to quantify the stability and fairness of models using the CIFAR-10 dataset and ResNet-18 model with $\beta = 0.5$. For stability analysis, we used the standard deviation of the moving average (MA) to quantify model stability. We calculated the performance of our method and all baselines on this metric, and the results are shown in Table III. We can find that CaBaFL performs best. Meanwhile, our approach considers the fairness of device selection. By introducing the hyperparameter σ (see line 3 of Algorithm 2), our device selection strategy can ensure that all devices are selected with a similar number of times. We normalized the number of times a device is selected and used the variance of the normalized values of devices to quantify such fairness. We compared our device selection strategy with the classic random device selection strategy. We found that the fairness of using our device selection strategy is 8.7×10^{-7}, while the fairness of using the random strategy is 1×10^{-6}, indicating the fairness of CaBaFL is similar to the one of FedAvg.

C. Impacts of Different Configuration

1) Impacts of Different Features: We investigated the impact of selecting features from different model layers on accuracy. Since ResNet-18 has 4 blocks, we performed pooling operations on the feature outputs of each block separately and used them as features when selecting the device. The feature dimensions of Blocks 1 to 4 are 64, 128, 256, and 512, respectively. Fig. 6 shows all experiment results conducted on CIFAR-10 with IID distribution and Dirichlet distribution where $\beta = 0.5$. We can observe that selecting the features from Block 4 performs the best. We can conclude that because

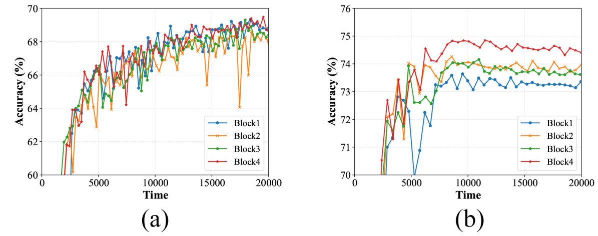


Fig. 6. Impact of different features on test accuracy. (a) $\beta = 0.5$. (b) IID.

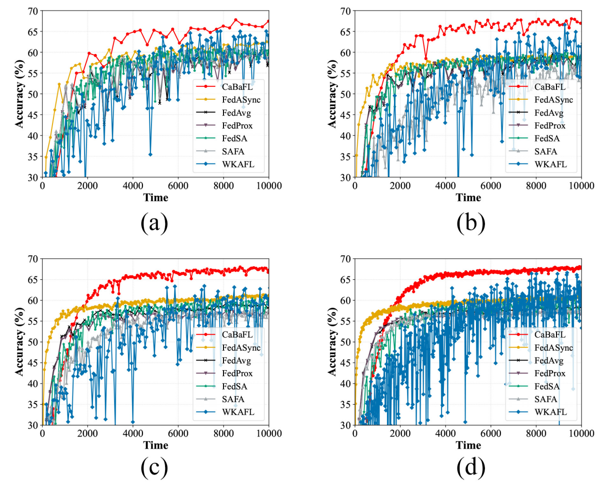


Fig. 7. Learning curves for different ratios of simultaneously training clients. (a) 5%. (b) 10%. (c) 20%. (d) 50%.

of the highest dimensionality of Block 4, the features from Block 4 can represent data distribution in a more fine-grained manner, thus achieving the highest accuracy.

2) Impacts of Different Number of Simultaneously Training Devices: By default, we assumed that 10% of devices were selected to participate in the training simultaneously. To investigate the impact of different numbers of devices trained simultaneously, we also considered four different ratios (i.e., 5%, 10%, 20%, and 50%) for simultaneously training devices, and conducted experiments using ResNet-18 on CIFAR-10 with $\beta = 0.5$. Note that in the experiments, we did not specify the percentages of stragglers using the above ratios. Instead, we assumed that the performance of devices within an experiment follows some Gaussian distribution, where stragglers only denote weak devices with low-training speed. From Fig. 7, we can find that more selected devices will lead to more stable training convergence. This is because more selected devices lead to more training data, thus alleviating the weight divergence problem during the global model training.

3) Impacts of Different Device Performance: We conducted an experiment to assess the generalization ability of CaBaFL. Our experiment considered the impact of network conditions on delay and the varying computing capabilities of devices on local training time. To facilitate the evaluation, we combined the delay and local training time, assuming that their combinations adhere to Gaussian distributions. We consider five kinds of device conditions, i.e., excellent, high, medium, low, and critical, as illustrated in Table IV.

TABLE IV
TRAINING PERFORMANCE FOR DEVICES

Quality	Excellent	High	Medium	Low	Critical
Device Settings	N(10,1)	N(15,2)	N(20,2)	N(30,3)	N(50,5)

TABLE V
CONFIGURATIONS OF DIFFERENT DEVICE COMPOSITIONS

Config	# of Clients				
	Excellent	High	Medium	Low	Critical
Config1	40	30	10	10	10
Config2	10	10	10	30	40
Config3	10	20	40	20	10
Config4	20	20	20	20	20

TABLE VI
TEST ACCURACY COMPARISON FOR DIFFERENT DEVICE CONFIGURATIONS

Configration		Config1	Config2	Config3	Config4
Test Acuracy (%)	FedAvg	44.91 ± 2.63	44.78 ± 2.57	45.45 ± 2.83	45.32 ± 2.99
	FedProx	46.34 ± 2.55	44.59 ± 2.86	46.14 ± 3.11	43.79 ± 2.63
	FedASync	50.31 ± 1.78	48.67 ± 1.65	50.12 ± 2.17	49.16 ± 1.47
	FedSA	47.75 ± 2.32	46.32 ± 1.21	46.51 ± 2.54	47.04 ± 2.51
	SAFA	41.68 ± 2.97	40.14 ± 2.51	41.32 ± 3.02	40.44 ± 3.88
	WKAFL	49.06 ± 6.67	39.30 ± 6.65	46.79 ± 6.16	39.55 ± 5.12
	CaBaFL	$\mathbf{56.96 \pm 0.58}$	$\mathbf{56.06 \pm 0.50}$	$\mathbf{56.13 \pm 0.54}$	$\mathbf{56.74 \pm 0.60}$

Based on Table IV, we considered four configurations as shown in Table V that have different device compositions to evaluate the generalization ability of CaBaFL.

We conducted our experiments using the ResNet-18 model on the CIFAR-10 dataset with $\beta = 0.1$. Table VI compares the accuracy between CaBaFL and all baselines.

From this table, we can find that CaBaFL can achieve the highest and stablest accuracy in all four cases. Meanwhile, we can observe that the accuracy of baselines decreases significantly when the numbers of stragglers increase, while the accuracy differences of our approach are small. This is mainly because both synchronous and semi-asynchronous methods must wait for a certain number of models before aggregating. Therefore, their accuracy is affected by stragglers. Moreover, the number of stale models increased due to the increase in the number of stragglers, leading to a decrease in accuracy.

4) Impacts of Different Model Training Times: To investigate the impact of model training times k, we set up five different model training times, respectively, 10, 15, 20, 25, and 30, and conducted experiments using ResNet-18 model on CIFAR-10 dataset with IID distribution. Fig. 8 exhibits the experiment result. We can find that as the model training times k increase, the accuracy of the global model improves, but too high a number of model training times leads to a decrease in global model accuracy and instability of the learning curve.

5) Impacts of Task Types: To evaluate the generalization ability of our approach to different types of tasks, we conducted experiments on two well-known nonimage-based datasets, i.e., the text type dataset Shakespeare [37] and the table type dataset Activity [38], using the LSTM model and MLP model, respectively. From Table VII, we can find that CaBaFL can achieve the best results in all two cases, indicating the generalization ability of CaBaFL on different tasks.

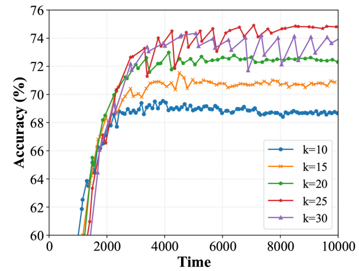
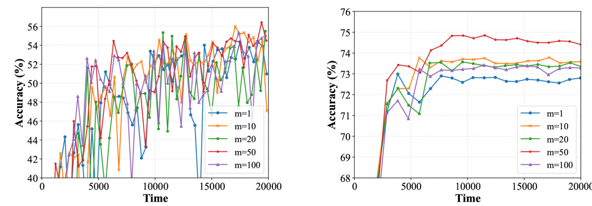


Fig. 8. Impact of training times on test accuracy.



(a) (b)

Fig. 9. Impact of the feature collection cycle. (a) $\beta = 0.1$. (b) IID.

6) Impacts of Feature Collection Cycles: We assumed that feature collection is performed after every m cache aggregation. We set m to be 1, 10, 20, 50, and 100, respectively, and conduct experiments on the CIFAR-10 dataset and ResNet-18 model with both the IID distribution and the Dirichlet distribution (with $\beta = 0.1$). Fig. 9 exhibits the experiment results. From the results, we can find that both too-fast and too-slow feature collection cycles result in lower-global model accuracy. In addition, a feature collection cycle that is too fast will significantly increase the communication overhead, since the server needs to frequently send the global model to all devices for feature collection.

D. Ablation Study

1) Key Components of CaBaFL: To demonstrate the effectiveness of CaBaFL, we investigated five variants of CaBaFL: 1) *Conf1* represents selecting devices-based only on the similarity between model feature and global feature; 2) *Conf2* denotes selecting devices-based only on data size; 3) *Conf3* indicates randomly selecting devices; 4) *Conf4* represents aggregating models in the L2 Cache, which means only L2 Cache is available in the server; and 5) *Conf5* denotes averaging aggregation during model aggregation.

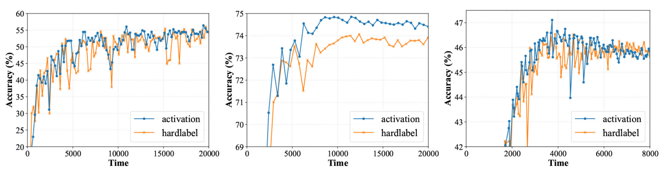
We conducted experiments using the ResNet-18 model on CIFAR-10 with both non-IID and IID settings, and the results are presented in Table VIII. Table VIII shows that CaBaFL achieves the highest-test accuracy among all six designs. By comparing the performance of *Conf4* and CaBaFL, we can observe that our 2-level cache structure significantly improves the stability of model training and enhances model accuracy in scenarios with high levels of data heterogeneity. By comparing the performance of *Conf5* and CaBaFL, we observed that our cache aggregation strategy significantly improves the model accuracy. Furthermore, by comparing the performance of *Conf1* to *Conf3* with that of CaBaFL, we observed that our feature balance-guided device selection strategy is effective

TABLE VII
TEST ACCURACY COMPARISON FOR DIFFERENT TASK TYPES

Dataset	Test Accuracy (%)						
	FedAvg	FedProx	FedASync	FedSA	SAFA	WKAFL	CaBaFL
Shakespeare	51.61 ± 0.07	51.63 ± 0.12	50.83 ± 0.19	51.89 ± 0.05	51.13 ± 0.23	51.78 ± 0.32	$\mathbf{52.69 \pm 0.10}$
Activity	95.10 ± 0.16	95.23 ± 0.12	95.06 ± 0.08	95.27 ± 0.09	93.47 ± 0.45	95.17 ± 0.76	$\mathbf{95.43 \pm 0.23}$

TABLE VIII
COMPARISON OF TEST ACCURACY AMONG CABAFL AND ITS VARIANTS

Settings	Test Accuracy(%)			
	$\beta = 0.1$	$\beta = 0.5$	$\beta = 1.0$	IID
CaBaFL	55.46 ± 0.67	69.31 ± 0.18	72.84 ± 0.36	74.83 ± 0.07
Conf1	52.54 ± 0.95	69.05 ± 0.23	72.73 ± 0.23	73.99 ± 0.18
Conf2	54.16 ± 0.65	68.04 ± 0.20	71.59 ± 0.35	72.60 ± 0.18
Conf3	53.50 ± 1.23	69.16 ± 0.19	72.80 ± 0.34	72.67 ± 0.26
Conf4	53.29 ± 2.12	69.27 ± 0.42	72.49 ± 0.26	73.91 ± 0.09
Conf5	53.08 ± 0.49	68.30 ± 0.25	71.47 ± 0.12	72.99 ± 0.18



(a) (b) (c)

Fig. 10. Comparison of accuracy based on activation amounts and hard labels. (a) CIFAR-10, $\beta = 0.1$. (b) CIFAR-10, IID. (c) CIFAR-100, special.

in different scenarios of data heterogeneity. Especially under the extreme data heterogeneity condition where $\beta = 0.1$, our device selection strategy can significantly improve the accuracy of the global model compared to *Conf3*, which is random device selection.

2) Activation Amounts Versus Hard Labels: We evaluated the impact of activation amounts and hard labels on the training performance of CaBaFL. Fig. 10(a) and (b) show the results conducted on the combination of CIFAR-10 and ResNet-18 within IID and non-IID ($\beta = 0.1$), respectively. We can find activation amount-based CaBaFL outperforms hard label-based CaBaFL since activation distributions provide a fine-grained representation for data distributions. Note that the CIFAR-100 dataset consists of 20 coarse-grained categories, which can be refined into 100 fine-grained categories. To reproduce Observation 2 in Section III, we construct a special data distribution for devices based on CIFAR-100, where the device data are IID according to coarse-grained categories but non-IID according to fine-grained categories. Fig. 10(c) shows the training processes using ResNet-18, where activation amount-based CaBaFL outperforms hard label-based CaBaFL due to its fine-grained representation of data distributions.

E. Real Test-Bed Evaluation

To evaluate the effectiveness of our approach in practical scenarios, we conducted experiments on real devices using CNN models and the CIFAR-10 dataset, considering both IID and non-IID ($\beta = 1.0$) scenarios. Fig. 11 shows our test-bed platform, which uses: 1) an Ubuntu-based cloud server equipped with an Intel i9 CPU, 32-GB memory, and an NVIDIA RTX 3090Ti GPU and 2) four NVIDIA Jetson Nano boards and six Raspberry Pi boards as heterogeneous clients.

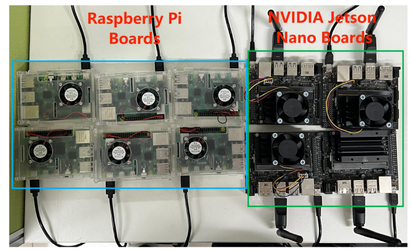
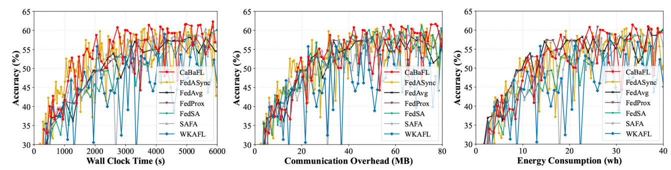
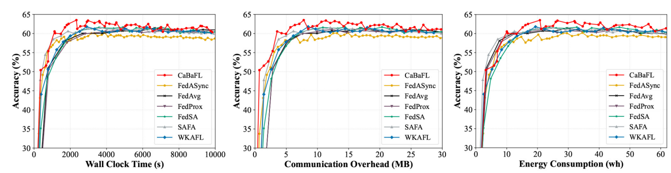


Fig. 11. Our real test-bed platform.



(a) (b) (c)

Fig. 12. Comparison of accuracy for different system metrics ($\beta = 1.0$). (a) Wall clock time. (b) Communication overhead. (c) Energy consumption.



(a) (b) (c)

Fig. 13. Comparison of accuracy for different system metrics (IID). (a) Wall clock time. (b) Communication overhead. (c) Energy consumption.

Figs. 12 and 13 present a comparison of accuracy for different system metrics on our test-bed platform. We can find that CaBaFL can also achieve the best performance compared to all the baselines. Please note that the number of samples per device in the real test-bed platform is much bigger than the number of samples per device in the simulation experiment. As a result, the experiment in Fig. 10 is more likely to cause much more severe catastrophic forgetting even when $\beta = 1.0$, resulting in large amplitudes of the learning curves. Note that since the local epoch is smaller with $\beta = 1.0$ than with IID distribution, the communication overhead is greater with $\beta = 1.0$ than with IID distribution.

F. Impact of Hyperparameters

To evaluate the impact of hyperparameters in CaBaFL, we set different configurations for α, γ, and σ, respectively, and performed experiments in the CIFAR-10 dataset using

TABLE IX
IMPACT OF α ON TEST ACCURACY

Hetero. Settings	Value of α 0.5	0.7	0.9	1
$\beta = 0.1$	55.46 ± 0.67	54.63 ± 0.46	55.14 ± 0.79	53.70 ± 0.79
IID	73.07 ± 0.09	73.31 ± 0.11	73.48 ± 0.20	74.83 ± 0.07

TABLE X
IMPACT OF σ ON TEST ACCURACY

Hetero. Settings	Value of σ 1×10^{-6}	2×10^{-6}	3×10^{-6}	4×10^{-6}
$\beta = 0.1$	55.19 ± 0.27	53.60 ± 0.33	55.46 ± 0.67	54.44 ± 1.07
IID	74.83 ± 0.07	72.80 ± 0.10	73.17 ± 0.15	73.72 ± 0.09

TABLE XI
IMPACT OF γ ON TEST ACCURACY

Hetero. Settings	Value of γ 0.4	0.3	0.2	0.01
$\beta = 0.1$	54.59 ± 1.62	55.56 ± 0.77	54.08 ± 1.04	53.26 ± 1.95
IID	73.19 ± 0.10	73.06 ± 0.14	73.11 ± 0.11	74.83 ± 0.07

ResNet-18 model with IID and Non-IID ($\beta = 0.1$) scenarios. Tables IX–XI exhibit the experiment results.

Table IX shows the inference accuracy with different settings of α. From Table IX, we can find that when the non-IID degree is high, a smaller α can improve the accuracy of the model, while under the IID distribution, a larger α can improve the accuracy of the model. This is mainly because when the degree of non-IID is high, the local data size of the device varies greatly, so a smaller α is needed to balance this difference, while the opposite is true in the case of IID.

Table X shows the impact of σ. We can find that when the non-IID degree is high, a larger σ can improve the accuracy of the model, while under the IID distribution, a smaller σ can improve the accuracy of the model. Please note that CaBaFL normalizes the number of times a device is selected before calculating the variance of the number of times the device is selected, so the value of σ is less than 0.

Table XI shows the impact of γ. We can find that when the value of γ is appropriate, CaBaFL can achieve better-global model accuracy. When the non-IID degree is high, a larger γ, i.e., filtering more models to make the cumulative training data of the model in the L1 cache more balanced, is required to achieve higher-global model accuracy, while under the IID distribution, a smaller γ, i.e., filtering few models, allows CaBaFL to achieve higher-model accuracy.

VI. DISCUSSION

A. Fairness and Workload Balance

We strive to address the fairness and workload balance issues in our device selection mechanism, where we use the hyperparameter σ to control the fairness of device selection (see line 3 of Algorithm 2). During the FL training, if the variance of the number of device selections exceeds σ, CaBaFL will select idle devices that have been touched least frequently, ensuring the fairness of device selection. We recorded the number of times a device was selected using our device selection strategy as well as the random selection strategy. From Fig. 14, we can find that compared to the random

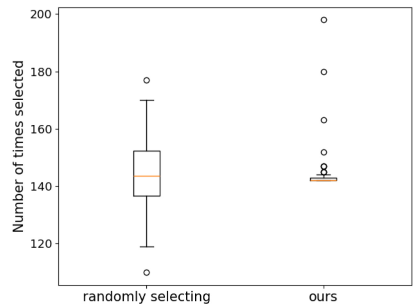


Fig. 14. Comparison between random selection strategy and our feature balance-guided device selection strategy.

selection strategy, except for a few outliers, the number of selected devices for our strategy is very close, demonstrating that the workload between devices is balanced.

B. Complexity and Scalability

Assume that the dimension size of flattened feature distributions is d, the current FL training round index is k, and the number of activated devices in a round is n. Let D be the number of all devices, and m be the total number of model parameters. The memory and time complexity analysis of our approach is as follows.

Time Complexity: The time complexity of our method mainly comes from Cache Update and Device Selection.

1) *Cache Update:* From lines 9–12 of Algorithm 1, we can figure out that the time complexity of Cache Update is $O(k \log k + d)$.
2) *Device Selection:* From lines 10–16 of Algorithm 2, we can determine the time complexity of device selection is $O(n(d + n))$.

Above all, the overall time complexity of our method is $O(k \log k + d + n(d + n))$. Even for a FL system with 1000 devices, one round of Cache Update and Device Selection costs less than 2ms, which is negligible compared with local training time.

Memory Complexity: The memory complexity of our method mainly also comes from cache update and device selection.

1) *Cache Update:* Although there are two caches in our method, we can only maintain the L1 cache, and the L2 cache can only exist logically. Because the model in the L2 cache can be deleted from memory after being sent to the device for training. Therefore, the memory complexity is $O(nm)$. In addition, our method also maintains the model feature distribution and device feature distribution, and its memory complexity is $O((D+n)d)$.
2) *Device Selection:* From Algorithm 2, we can determine that the memory complexity of device selection is $O(Dd + n)$. Since $nm \gg Dd + dn + n$, the overall memory complexity of our method is approximately $O(nm)$, which is the same as the traditional FL.

VII. CONCLUSION

To improve the inference performance of FL in large-scale AIoT applications, this article introduced a new asynchronous

FL method named CaBaFL, which can effectively mitigate the notorious straggler and data imbalance problems caused by device and data heterogeneity. Specifically, CaBaFL maintains a hierarchical cache data structure on the server that can: 1) mitigate the straggler problem caused by device heterogeneity using our proposed hierarchical cache-based aggregation mechanism and 2) achieve stable training convergence and high-global model accuracy by properly placing models in different hierarchies of the cache. Moreover, by using our feature balance-guided device selection strategy, CaBaFL can alleviate the performance deterioration caused by data imbalance. Comprehensive experimental results demonstrate that CaBaFL can achieve much better-inference performance compared with SOTA heterogeneous FL methods within both IID and non-IID scenarios.

Acknowledgment

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore and Cyber Security Agency of Singapore.

References

[1] J. Xiong and H. Chen, “Challenges for building a cloud native scalable and trustable multi-tenant AIoT platform,” in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, 2020, p. 26.

[2] X. Han et al., “ADS-lead: Lifelong anomaly detection in autonomous driving systems,” *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 1, pp. 1039–1051, Jan. 2022.

[3] M. Hu, E. Cao, H. Huang, M. Zhang, X. Chen, and M. Chen, “AIoTML: A unified modeling language for AIoT-based cyber-physical systems,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. (TCAD)*, vol. 40, no. 11, pp. 3545–3558, Nov. 2023.

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

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

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

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

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

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

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

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

[12] Q. Liu, C. Chen, J. Qin, Q. Dou, and P.-A. Heng, “FedDG: Federated domain generalization on medical image segmentation via episodic learning in continuous frequency space,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2021, pp. 1013–1023.

[13] S. Safavat and D. B. Rawat, “Asynchronous federated learning for intrusion detection in vehicular cyber-physical systems,” in *Proc. IEEE Conf. Comput. Commun. (INFOCOM) Workshops*, 2023, pp. 1–6.

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

[15] C. Yu et al., “Distributed learning over unreliable networks,” in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2019, pp. 7202–7212.

[16] F. Sattler, S. Wiedemann, K.-R. Mller, and W. Samek, “Robust and communication-efficient federated learning from non-i.i.d. data,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 9, pp. 3400–3413, Sep. 2020.

[17] P. Kairouz et al., “Advances and open problems in federated learning,” *Found. Trends® Mach. Learn.*, vol. 14, nos. 1–2, pp. 1–210, 2021.

[18] S. Khodadadian, P. Sharma, G. Joshi, and S. T. Maguluri, “Federated reinforcement learning: Linear speedup under Markovian sampling,” in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2022, pp. 10997–11057.

[19] A. Li, L. Zhang, J. Tan, Y. Qin, J. Wang, and X.-Y. Li, “Sample-level data selection for federated learning,” in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2021, pp. 1–10.

[20] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, “Federated optimization in heterogeneous networks,” in *Proc. Mach. Learn. Syst. (MLSys)*, vol. 2, 2020, pp. 429–450.

[21] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, “SCAFFOLD: Stochastic controlled averaging for federated learning,” in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2020, pp. 5132–5143.

[22] Z. Zhang, Y. Zhang, D. Guo, S. Zhao, and X. Zhu, “Communication-efficient federated continual learning for distributed learning system with non-IID data,” *Sci. China Inf. Sci.*, vol. 66, no. 2, 2023, Art. no. 122102.

[23] C. Xie, S. Koyejo, and I. Gupta, “Asynchronous federated optimization,” 2019, *arXiv:1903.03934*.

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

[25] W. Wu, L. He, W. Lin, R. Mao, C. Maple, and S. Jarvis, “SAFA: A semi-asynchronous protocol for fast federated learning with low overhead,” *IEEE Trans. Comput.*, vol. 70, no. 5, pp. 655–668, May 2021.

[26] Q. Ma, Y. Xu, H. Xu, Z. Jiang, L. Huang, and H. Huang, “FedSA: A semi-asynchronous federated learning mechanism in heterogeneous edge computing,” *IEEE J. Sel. Areas Commun.*, vol. 39, no. 12, pp. 3654–3672, Dec. 2021.

[27] Y. Zhang et al., “FedMDS: An efficient model discrepancy-aware semi-asynchronous clustered federated learning framework,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 3, pp. 1007–1019, Mar. 2023.

[28] C. Chen, Z. Chen, Y. Zhou, and B. Kailkhura, “FedCluster: Boosting the convergence of federated learning via cluster-cycling,” in *Proc. IEEE Int. Conf. Big Data*, 2020, pp. 5017–5026.

[29] H. Jin, D. Bai, Y. Dai, L. Gu, C. Yu, and L. Sun, “Personalized edge intelligence via federated self-knowledge distillation,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 2, pp. 567–580, Feb. 2023.

[30] Z. Zhou, Y. Li, X. Ren, and S. Yang, “Towards efficient and stable K-asynchronous federated learning with unbounded stale gradients on non-IID data,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, pp. 3291–3305, Dec. 2022.

[31] C.-H. Hu, Z. Chen, and E. G. Larsson, “Scheduling and aggregation design for asynchronous federated learning over wireless networks,” *IEEE J. Sel. Areas Commun.*, vol. 41, no. 4, pp. 874–886, Apr. 2023.

[32] Y. Zang, Z. Xue, S. Ou, L. Chu, J. Du, and Y. Long, “Efficient asynchronous federated learning with prospective momentum aggregation and fine-grained correction,” in *Proc. AAAI Conf. Artif. Intell. (AAAI)*, 2024, pp. 16642–16650.

[33] Y. Xu, Z. Ma, H. Xu, S. Chen, J. Liu, and Y. Xue, “FedLC: Accelerating asynchronous federated learning in edge computing,” *IEEE Trans. Mobile Comput.*, vol. 23, no. 5, pp. 5327–5343, May 2024.

[34] T.-M. H. Hsu, H. Qi, and M. Brown, “Measuring the effects of non-identical data distribution for federated visual classification,” 2019, *arXiv:1909.06335*.

[35] A. Paszke et al., “PyTorch: An imperative style, high-performance deep learning library,” in *Proc. Annu. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2019, pp. 1–12.

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

[37] S. Caldas et al., “LEAF: A benchmark for federated settings,” 2018, *arXiv:1812.01097*.

[38] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. Reyes-Ortiz, “A public domain dataset for human activity recognition using smartphones,” in *Proc. Eur. Symp. Artif. Neural Netw. (ESANN)*, 2013, pp. 437–442.