ShuffleFL
Addressing Heterogeneity in Multi-Device Federated Learning

Zhu, Ran; Yang, Mingkun; Wang, Qing

DOI
10.1145/3659621

Publication date
2024

Document Version
Final published version

Published in
Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies

ShuffleFL: Addressing Heterogeneity in Multi-Device Federated Learning

RAN ZHU, Delft University of Technology, The Netherlands
MINGKUN YANG, Delft University of Technology, The Netherlands
QING WANG, Delft University of Technology, The Netherlands

Federated Learning (FL) has emerged as a privacy-preserving paradigm for collaborative deep learning model training across distributed data silos. Despite its importance, FL faces challenges such as high latency and less effective global models. In this paper, we propose **ShuffleFL**, an innovative framework stemming from the hierarchical FL, which introduces a *user layer* between the FL devices and the FL server. ShuffleFL naturally groups devices based on their affiliations, e.g., belonging to the same user, to *ease* the strict privacy restriction–*"data at the FL devices cannot be shared with others"*, thereby enabling the exchange of local samples among them. The user layer assumes a multi-faceted role, not just aggregating local updates but also coordinating data shuffling within affiliated devices. We formulate this **data shuffling** as an optimization problem, detailing our objectives to align local data closely with device computing capabilities and to ensure a more balanced data distribution at the intra-user devices. Through extensive experiments using realistic device profiles and five non-IID datasets, we demonstrate that ShuffleFL can improve inference accuracy by 2.81% to 7.85% and speed up the convergence by 4.11× to 36.56× when reaching the target accuracy.

CCS Concepts: • **Computing methodologies** → **Distributed computing methodologies**; • **Human-centered computing** → **Ubiquitous and mobile computing**.

Additional Key Words and Phrases: Federated Learning, IoT, Data Heterogeneity, System Heterogeneity, Data Shuffling

ACM Reference Format:
Ran Zhu, Mingkun Yang, and Qing Wang. 2024. ShuffleFL: Addressing Heterogeneity in Multi-Device Federated Learning. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 8, 2, Article 85 (June 2024), 34 pages. https://doi.org/10.1145/3659621

1 INTRODUCTION

With the proliferation of mobile devices augmented with advanced sensing capabilities [11, 13], rich data generated at the edge presents numerous possibilities for carrying out complicated Internet of Things (IoT) applications. Nonetheless, data silos resulting from the data privacy concern hinder the potential of such valuable data. As a remedy to this issue, Federated Learning (FL) [21, 41] has emerged as a privacy-preserving Machine Learning (ML) paradigm that shifts the cloud data centers towards the edge. Specifically, FL enables collaborative training of ML models among mobile users located in different geographical regions, all while maintaining their own data stored on the devices. This feature makes FL exploit the device-side computing resources while preserving the data privacy, offering a wide range of edge intelligence services such as health monitoring [43, 51], typing assisting [14], and home automation [60].

Authors' Contact Information: Ran Zhu, r.zhu-1@tudelft.nl, Delft University of Technology, Delft, The Netherlands; Mingkun Yang, m.yang-3@tudelft.nl, Delft University of Technology, Delft, The Netherlands; Qing Wang, qing.wang@tudelft.nl, Delft University of Technology, Delft, The Netherlands.

ACM 2474-9567/2024/6-ART85
https://doi.org/10.1145/3659621

FL typically involves a vast number of devices, ranging from hundreds to even millions. A key challenge of deploying FL in real-world scenarios lies in the significant heterogeneity of these devices [15, 31]. Unbalanced communication or hardware capabilities (i.e., system heterogeneity) and non-IID training data (i.e., data heterogeneity) across devices undermine the performance and efficiency of FL. Substantial efforts have been devoted to alleviating this issue, including modifying local training objectives [12, 22, 30–33, 57], improving weights aggregation at the central server [16, 44, 47, 54, 55], clustering devices according to the similarity of their model parameters [4, 34, 43, 49], adapting the model to devices with varying capabilities [10, 15, 27, 28], and implementing device selection strategies [8, 9, 17, 26, 29]. However, all these endeavors encounter limitations imposed by data privacy concerns among devices, resulting in efforts that only scratch the surface when coping with the issue of heterogeneity. Indeed, numerous devices in the wild display inherent affiliations that can be grouped by users (which can refer to any entity with the required capability and authorization to engage with affiliated devices), with some devices originating from the same user. In this context, we naturally introduce the hierarchical *device-user-server* structure and partially relax the data privacy constraints by shuffling data among devices belonging to the same user, thereby addressing the heterogeneity problem. For example, a user may employ multiple IMU-enabled devices with a body area network during activities such as walking, along with other local sensor networks like IoT devices in smart homes. These scenarios make it feasible to challenge the *device data cannot be shared* constraint within the context of conventional federated learning.

Differences in user habits and device hardware capabilities like data generation ability and computational resources often lead to a substantial mismatch between the data volume used for local training and the computational capacity of each device. Smartwatches excel at collecting human activity data but are equipped with limited computing power, inevitably making them act as stragglers in FL. However, each connection from these devices plays a pivotal role in model training. The judicious utilization of device resources to augment their contribution to the global model is crucial. Data shuffling among devices can effectively allocate and utilize computational capabilities while mitigating data heterogeneity, as consistently corroborated in distributed computing [20, 53, 56, 61]. To this end, achieving resource balance in a heterogeneous device-user-server FL system will significantly enhance model performance, alleviating the straggler issue and leading to superior accuracy.

To deliver a functioning and practical system, we need to address three key challenges:

Challenge 1: How to design an effective data shuffling method to alleviate device stragglers while balancing the data distribution among devices? In the device-user, data heterogeneity and system heterogeneity are intricately intertwined. The orchestration of data exchange among devices, involving which devices receive data, how much data to receive, and what data to receive, necessitates an elaborate approach.

Challenge 2: How to add global guidance that can alleviate user stragglers to further improve the FL efficiency, without compromising privacy? In the user-server, time consumed by different users also varies dramatically. The straggler issue in the user-server and the device-user should be carefully integrated to avoid overfitting in terms of time reduction in the device-user for front-runner users while underfitting for user stragglers.

Challenge 3: How to ensure the additional time overhead introduced by data shuffling is elastic to various applications? In our practical testing, users with computing capabilities like a Raspberry Pi require an unbearable data shuffling time for an 80-class next-character prediction task [5], far exceeding the reduction in device local training time achieved through data exchange. A lightweight data shuffling scheme that can be applied to different levels of task complexity to preserve the advantages of data shuffling is crucial.

To tackle the above challenges, in this work, we propose ShuffleFL in device-user-server structured FL, a data shuffling scheme that enhances the training efficiency. ShuffleFL orchestrates data, communication, and computation resources across devices to address the inherent issues posed by data imbalance and system latency in FL. To this end, devices with substantial computational resources receive a larger allocation of data samples, mitigating the intra-user device stragglers issue and ensuring system efficiency. Selective exchange of data categories can promote a more balanced data distribution, enhancing data efficiency, and inference accuracy

(Section 4.2). Besides, we also take into account the diverse capacities of inter-users. By incorporating guidance from the global view, we adaptively scale the optimization focus between data and system heterogeneity. This prevents an overemphasis on system latency optimization for front-runner users and ensures sufficient attention toward reducing system latency for user stragglers, thereby alleviating the user stragglers issue (Section 4.3). Obtaining such data shuffling results introduces additional time overhead, primarily determined by the number of data categories involved in tasks. To adapt the ShuffleFL for different complexity levels of tasks, we design an optimization complexity reduction strategy to mitigate the associated time cost (Section 4.4).

We have evaluated the performance of ShuffleFL with four representative FL tasks (image classification, natural language processing, speech recognition, and human activity recognition) on five public datasets. We compare ShuffleFL with six baselines, and the experiments show that ShuffleFL outperforms SOTA in terms of inference accuracy (up to 7.85% improvement), and time-to-accuracy (up to 36.56× speedup while achieving the target accuracy). Overall, we make the following key contributions in this paper:

- To the best of our knowledge, ShuffleFL is the first-of-its-type work that proposes to challenge the constraint of *device data cannot be shared* in the device-user-server FL framework. This approach mitigates data and system heterogeneity by harmonizing the data, computation, and communication resources among devices, taking into account both intra-user (i.e., device-user) and inter-user (i.e., user-server) perspectives. Importantly, it achieves this without revealing any private information to the server.
- We devise an optimization complexity reduction strategy that clusters data categories with similar features to alleviate the computational burden introduced by ShuffleFL, thereby enabling ShuffleFL to be elastic to varying-scale FL scenarios.
- We conduct comprehensive evaluations to assess the performance of ShuffleFL in a dynamic and heterogeneous environment. Extensive experimental results consistently demonstrate the superior performance of ShuffleFL, showcasing advancements in both final inference accuracy and overall training efficiency.

2 BACKGROUND ON FL FRAMEWORKS

Traditional device-server FL frameworks. Traditional FL frameworks are illustrated in Figure 1(left). In these traditional two-layer device-server FL frameworks, a deep learning model is collaboratively trained by harnessing computational resources and data instances residing on distributed devices (i.e., clients), without sharing data between the server and devices, to maintain stringent privacy constraints. Within this device-server architecture, devices periodically offload locally trained models to the central server where local updates are aggregated for the global model. However, inherent disparities in computing resources and local data distributions amongst devices present formidable challenges to FL in obtaining a qualified global model within a confined time scope, especially with the increasingly involved edge devices. Specifically, the incongruity in computing resources and local data volume raises some straggler devices in the training process, impeding the global model from achieving a high *time-to-accuracy* performance. Devices endowed with superior sensing capacity amass copious data for model training, yet they may have tight resource budgets to process all the collected data locally. Furthermore, devices possess non-IID data, reflective of the unique ambient environments in which they are deployed. Consequently, training in a federation way encounters two inconsistencies: 1) the *intra-device* inconsistency between computing resources and sensing capacity, and 2) the *inter-device* data distribution inconsistency.

Hierarchical device-edge-server FL frameworks. In hierarchical device-edge-server architecture [38], as illustrated in Figure 1(middle), a two-step aggregation process is employed. Local updates are initially aggregated at the edge servers at a minimal cost of communication overheads, followed by a cloud-side aggregation on models uploaded from edge servers with low latency over the backbone network. This hierarchical architecture yields a notable reduction in communication costs compared to the two-layer FL frameworks. Although hierarchical frameworks reduce the training latency from the communication perspective, the predominant bottleneck in

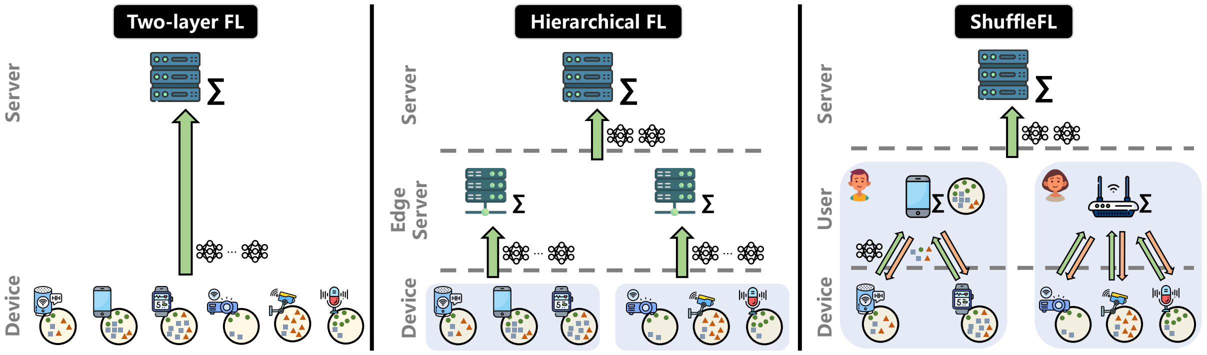


Fig. 1. The illustration/comparison of three FL frameworks: *(left)* traditional two-layer device-server FL; *(middle)* hierarchical device-edge-server FL; *(right)* our proposed device-'user'-server **ShuffleFL** framework, where devices belonging to same user opportunistically share data and computing resources to improve the overall performance without compromising privacy.

achieving high time-to-accuracy performance remains the local on-device training phase, which consumes a substantial proportion of the overall time in the FL process. Besides, the degraded model performance due to the non-IID nature of training data across devices remains unsolved in hierarchical frameworks. Therefore, it is imperative and necessary to address the aforementioned two inconsistency issues in hierarchical FL frameworks.

3 ShuffleFL OVERVIEW

In the hierarchical device-edge-server FL frameworks, the selection of devices to the proper edge servers is primarily based on communication bandwidth considerations, with emphasis on geographic location and time windows. However, they overlook intrinsic relationships between devices, specifically, the affiliation of a group of devices to an entity. The advantages of such a collective perspective compared to individual devices are twofold: 1) the overall data instances of devices in a group present a relatively balanced distribution, and 2) the overall computational resources are capable of completing the training task within a confined time interval.

Motivated by these facts, we propose an approach in the context of hierarchical FL frameworks. *By the possibility of relaxing the rigorous privacy issues among the devices belonging to the same user or entity, we shuffle the data of devices within each user to balance the local data distribution, as well as the alignment to their computational resources.* The hierarchical framework of our proposed ShuffleFL is shown in Figure 1(right). ShuffleFL trains the task-specific model in a federation manner using computing resources and data residing on the edge devices. ShuffleFL's hierarchical structure is composed of three layers, with models being locally trained on the **device** layer, aggregated in the **user** layer, and followed by a final model aggregation in the **server** layer for the global model. More specifically, we define the main components in the framework of ShuffleFL as follows:

- **Server:** The cloud server refers to the service provider, as well as the coordinator of the federated training task. The cloud server engages in direct communication with users, while all devices are transparent from the perspective of the cloud server.
- **User.** The broad-term user pertains to an entity with the capability and authorization to interact with affiliated devices and supports executing non-computationally intensive algorithms. For instance, in the context of a smart home, the user can be instantiated as the gateway, akin to the router, which exchanges data between connected devices, and can also serve as the platform for delivering IoT services.
- **Devices.** Each device is endowed with computing power parameterized by the inference time and communication capacity to its user parameterized. Devices are inherently partitioned according to their owners

(a.k.a, users). The devices can share data with each other to have a better system performance. Note that in this work, we assume that following data transmission, each device discards the transmitted data to prevent any additional burden on user storage. This assumption aligns with practical considerations. In scenarios involving multiple mobile devices owned by a user participating in an image recognition task, the repetitive copying of the same image across various devices is often considered impractical and unacceptable.

The unique feature of our ShuffleFL is data shuffling, which distinguishes it from other FL frameworks, such as the traditional FL or the hierarchical FL illustrated in Figure 1. The data shuffling in ShuffleFL has two primary goals: 1) the reallocation of the training task aligns with the computational capabilities of each device. Devices with substantial computational resources are assigned a greater number of samples, reducing the total wall-clock time required for training progress with high *system efficiency*; and 2) balancing the local data residing in devices, thereby improving the training accuracy and overall *data efficiency*. In this way, ShuffleFL could improve the time-to-accuracy performance of the FL system significantly.

Relaxation of privacy constraints. The proposed relaxation of privacy constraints is reasonable and implementable in specific scenarios, such as smart homes, smart farming, smart factories, and other similar environments, where deployed sensors generate data about target objects. For instance, smart home kits, typically including certain kinds of sensors, and wearable devices (e.g., smartwatch, smartphone, earphones, etc.) monitor the user's daily activities. The data exchange among devices belonging to a user does not compromise privacy in such contexts. In this way, breaking redundant privacy constraints and permitting devices within a defined range to exchange local samples, yields tangible benefits in terms of system efficiency (i.e., training latency) and data efficiency (i.e., model performance). Specifically, sample reallocation raises the consistency of training data volumes and computational capacities across devices, resulting in a significant reduction in aggregation latency for a global model in each communication round. Furthermore, data distribution of devices in the same group achieves equilibrium post-shuffling, mitigating biases stemming from inherent device properties such as deployment environments, and working status. This balanced distribution facilitates the aggregated model's ascent to high accuracy.

4 SYSTEM DESIGN

In this section, we detail the design of ShuffleFL, placing a focal point on the key role of data shuffling. Section 4.1 presents the system model and the objective for data shuffling, employing transition matrices, and specifies the challenges of deriving the optimal data shuffling. The optimal transition matrix is approached from two sides, concentrating not only on devices within a user (Section 4.2, *intra-user data shuffling analysis*) but also from a broader user-level perspective (Section 4.3, *inter-user optimization*). Additionally, Section 4.4 introduces a dimensionality reduction strategy, aimed at mitigating the practical implementation challenge of ShuffleFL in complex tasks with a multitude of possible classes. We conclude the overall training pipeline in Section 4.5.

4.1 System Model

We consider an FL system with a Server denoted as S, a set of Users $\mathcal{U}$, and a number of Devices $\mathcal{V}$. For each device $v \in \mathcal{V}$, we model its computing capability using the inference time p^v (in s/frame), and its communication capacity –to the user it belongs to– with the uplink data rate $b^v_\uparrow$ and the downlink data rate $b^v_\downarrow$ (both in b/s). We denote the local data distribution $\mathcal{D}^v$ residing at the device v as a vector $\mathbf{D}^v = [d_{1,v}, \cdots, d_{c,v}] \in \mathbb{R}^c$, where c is the number of classes of the dataset for a specific classification task. For example, in FEMNIST [5] where the collection of images is commonly used to train deep learning models for computer vision, $c = 62$. Besides, we assume each device can only belong to a single user, that is $\mathcal{V}^i \cap \mathcal{V}^j = \emptyset, \forall i, j \in \mathcal{U}$, and we assume $\mathcal{V} = \mathcal{V}^1 \cup \cdots \cup \mathcal{V}^{|\mathcal{U}|}$.

Intra-user data shuffling matrix. Shuffling the data among the devices of a user is key in our proposed ShuffleFL to improve the system performance in terms of inference accuracy as well as time-to-accuracy. In the n-th communication round, affiliated devices $\mathcal{V}^u$ of user $u \in \mathcal{U}_n$ exchange local data according to the their respective transition matrix $\mathbf{T}_n^v \in \mathbf{R}^{c\times|\mathcal{V}^u|}$ for $v \in \mathcal{V}^u$. In such a matrix, the entry $\mathbf{T}_{i,j} \in \mathbb{Z}_0^+$ in the i-th row and j-th column represents the number of i-class samples to be transferred to device j. Following the transition matrices, devices send the transferred samples to the user, and the user relays to the target devices in parallel. The original local data $\mathcal{D}_n^v$ (in distribution $\mathbf{D}_n^v$) undergoes transformation to $\hat{\mathcal{D}}_n^v$ (in distribution $\hat{\mathbf{D}}_n^v$) after data shuffling. The transformation of local data distribution after data shuffling can be formulated as

$$\hat{\mathbf{D}}_n^{v\top} = \sum_{v' \in \mathcal{V}^u} \mathbf{T}_{n(:v)}^{v'}. \tag{1}$$

The transition matrix of device v satisfies $\sum_j^{|\mathcal{V}^u|} \mathbf{T}_{i,j} = d_{i,v}, \forall i \in \{1, \cdots, c\}$, indicating that the i-class samples hold by device v should be either transferred to other devices or remain locally (i.e., $\mathbf{T}_{i,v}$). This is due to the stringent storage budget of mobile devices, which necessitates deleting samples after they are transferred to other devices. In this way, the overall samples across devices remain unchanged: $\bigcup_{v\in\mathcal{V}} \mathcal{D}_{n-1}^v = \bigcup_{v\in\mathcal{V}} \mathcal{D}_n^v$, and nonoverlapping $\mathcal{D}_n^i \cap \mathcal{D}_n^j = \emptyset, \forall n, i, j$.

Objective: To optimize the intra-user data shuffling in order to improve the system efficiency, we rely on two key aspects: *Data Imbalance (DI)* and *System Latency (SL)*. The intuitive optimization objective is to minimize the maximum values of both *DI* and *SL* among the devices of a user:

$$\min_{\mathcal{T}_n^u} \left\{ \max\{SL_n^v\}_{v\in\mathcal{V}^u} + \alpha \max\{DI_n^v\}_{v\in\mathcal{V}^u} \right\}, \tag{2}$$

where α scales the *DI* term, ensuring it carries the same importance in the objective as *SL*. This is necessary because *DI* and *SL* differ significantly in magnitude. Optimizing $\mathcal{T}_n^u$ according to Equation (2) could improve both wall-clock training time and global model performance compared to training without the proposed intra-user data shuffling, as we will show later in Section 6. However, obtaining the optimal transition matrix presents non-trivial challenges in three folds:

- **Coupled intra-user DI and SL.** Achieving balanced local data (low *DI*) typically entails the communication cost of transferring samples across devices, which increases the system latency (high *SL*), rendering an underlying trade-off between two tightly coupled terms. This necessitates the careful consideration of gains from data balancing and the overhead it introduces. We will analyze this in Section 4.2.
- **Inter-user disparity.** Improving the time-to-accuracy of the global model is not straightforward, even when each user can coordinate their devices to shuffle the local data samples properly. The potential disparities in the average device capabilities among users cannot be disregarded, as the time required to obtain the global model, in each communication round, depends on the latency of the last user to offload its local aggregated model to the server. We will present our method to tackle this problem in Section 4.3.
- **Computing complexity.** The dimensions of the transition matrices increase with the number of classes in the classification task, leading to an exponential rise in computational complexity when optimizing the transition matrices $\mathbf{T}$. Considering the limited computational capability on the user side, we must simplify the problem complexity of identifying the best transition matrices, particularly when the classification task has a large possible classes. This will be tackled in Section 4.4.

4.2 Preliminary Analysis of Intra-User Data Shuffling

In the n-th communication round, a user $u \in \mathcal{U}_n$ guides the data shuffling among its devices $\mathcal{V}^u$ through a set of transition matrices $\mathcal{T}_n^u = \{\mathbf{T}_n^v\}_{v\in\mathcal{V}^u}$. This guidance is based on the 4-tuple $\{\mathbf{D}_n^v, b_\downarrow^v, b_\uparrow^v, p^v\}_{v\in\mathcal{V}^u}$, covering the factors of local data distribution, communication bandwidth, and computing capability of devices. Establishing a clear relationship between the action of exchanging local samples and the goal of improving data and system efficiency is crucial for a quantitative evaluation of the intra-user data shuffling, thereby helping to find the optimal $\mathcal{T}_n^u$. Here, we detail how we formulate the intra-user data imbalance and system latency.

Data Imbalance (DI). Given the total number of local samples, devices with higher data efficiency in model training are those having a more balanced local data distribution. In line with this idea, we define a reference distribution as $\tilde{\mathbf{D}}_n^v = [\lfloor \frac{|\hat{\mathcal{D}}_n^v|}{c} \rfloor, \cdots, \lfloor \frac{|\hat{\mathcal{D}}_n^v|}{c} \rfloor] \in \mathbb{R}^c$. We choose a balanced data distribution as a reference mainly because all the task-related classes matter to the model performance. Besides, since the data distribution in the real world is agnostic, an unbiased model could mitigate the fairness issues upon deployment in reality, which performs fairly across different classes. We use the Jensen–Shannon (JS) divergence between the shuffled data distribution and the reference distribution to measure the data imbalance as follows:

$$DI_n^v = JS\left(\hat{\mathbf{D}}_n^v \| \tilde{\mathbf{D}}_n^v\right) = \frac{1}{2}KL\left(\hat{\mathbf{D}}_n^v \| \bar{\mathbf{D}}_n^v\right) + \frac{1}{2}KL\left(\tilde{\mathbf{D}}_n^v \| \bar{\mathbf{D}}_n^v\right), \tag{3}$$

where $KL(\cdot)$ is the Kullback–Leibler (KL) divergence of two vectors and the mixture distribution $\bar{\mathbf{D}}_n^v = \frac{\hat{\mathbf{D}}_n^v + \tilde{\mathbf{D}}_n^v}{2}$. Devices have higher data efficiency when local distribution tends to be less diverse from the balanced reference distribution.

System Latency (SL). We quantify the system efficiency of a device by considering the wall-clock time of training latency, which comprises the time consumed in communication, denoted as t_{comm}^v, for intra-user data shuffling, and on-device computation, denoted as t_{comp}^v. We have the system latency expressed below:

$$SL_n^v = t_{\text{comm}}^{v,n} + t_{\text{comp}}^{v,n}. \tag{4}$$

For a device v, sample exchange with other devices involves two steps: uploading transferred samples to the target device and downloading samples from other devices. Each step requires establishing a communication channel, relayed by the user (e.g., a gateway), connecting device v with the target device. Thus, the communication cost is

$$t_{\text{comm}}^{v,n} = \textstyle\sum_{i=1}^{c} \sum_{\substack{v' \in \mathcal{V}^u \\ v' \neq v}} \Bigg[\underbrace{I\big(\mathbf{T}_{n\,(i,v')}^{v}\big) \times \left(\frac{1}{b_{\uparrow}^{v}} + \frac{1}{b_{\downarrow}^{v'}}\right)}_{\text{Transmitting}} + \underbrace{I\big(\mathbf{T}_{n\;(i,v)}^{v'}\big) \times \left(\frac{1}{b_{\downarrow}^{v}} + \frac{1}{b_{\uparrow}^{v'}}\right)}_{\text{Receiving}} \Bigg], \tag{5}$$

where $I(\cdot)$ calculates the size of samples in bits. It is worth noting that we omit the time related to the device reporting updated models to the user (i.e., local offloading) in the calculation of t_{comm}^v, as this time component cannot be reduced by data shuffling with matrix $\mathbf{T}_n^v$.

The time consumption of on-device computation on the shuffled local data is

$$t_{\text{comp}}^{v,n} = E \ \times \ (B \times \lfloor \frac{|\hat{\mathcal{D}}_n^v|}{B} \rfloor) \ \times \ (3 \times p^v), \tag{6}$$

where $E \in \mathbb{Z}^+$ denotes the local training epochs of devices, B is the batch size, and $\lfloor \frac{|\hat{\mathcal{D}}_n^v|}{B} \rfloor$ is the number of iterations. The factor 3× is introduced based on the assumption that the backward pass is estimated to be 2 times the forward pass (i.e., the model inference) [1, 25].

So far, we have established the quantitative effects of data shuffling on a single device in both data and system efficiency by introducing the concepts of data imbalance and system latency. The problem in ShuffleFL is formulated as, for user $u \in \mathcal{U}_n$, how to obtain optimal transition matrices $\mathcal{T}_n^v$, making devices $\mathcal{V}^u$ having low data imbalance and system latency after the data shuffling, thereby ShuffleFL takes less wall-clock time consumption while achieving a better global model performance.

Mitigating the time variance of the devices within a user. We utilize Sequential Least Squares Programming (SLSQP) [24] to solve the formulated nonlinear optimization function (i.e., Equation (2)). We present a toy example in Figure 3, tracking two users, each including four devices, to showcase how different optimization objectives in

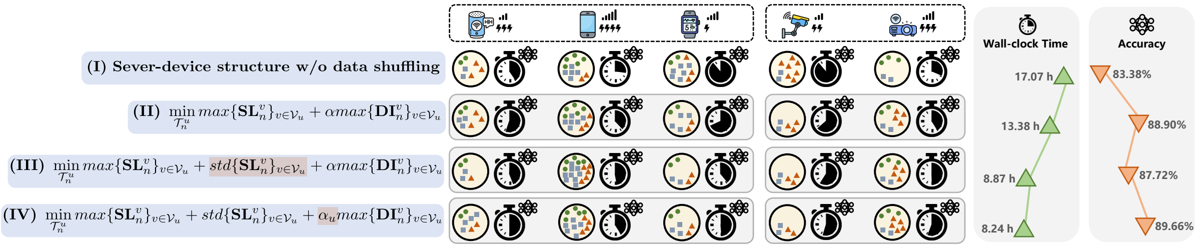

(I) Sever-device structure w/o data shuffling
(II) $\min_{\mathcal{T}_n^u} max\{\mathbf{SL}_n^v\}_{v\in\mathcal{V}_u} + \alpha max\{\mathbf{DI}_n^v\}_{v\in\mathcal{V}_u}$
(III) $\min_{\mathcal{T}_n^u} max\{\mathbf{SL}_n^v\}_{v\in\mathcal{V}_u} + std\{\mathbf{SL}_n^v\}_{v\in\mathcal{V}_u} + \alpha max\{\mathbf{DI}_n^v\}_{v\in\mathcal{V}_u}$
(IV) $\min_{\mathcal{T}_n^u} max\{\mathbf{SL}_n^v\}_{v\in\mathcal{V}_u} + std\{\mathbf{SL}_n^v\}_{v\in\mathcal{V}_u} + \alpha_u max\{\mathbf{DI}_n^v\}_{v\in\mathcal{V}_u}$
Wall-clock Time
17.07 h
13.38 h
8.87 h
8.24 h
Accuracy
83.38%
88.90%
87.72%
89.66%

Fig. 2. Data shuffling in ShuffleFL following $\mathcal{T}$, obtained by optimizing different objectives. The first row represents the conventional server-device FL frameworks without data shuffling between devices, which suffers from high training latency, as well as imbalanced local data distribution among devices. From the second row to the bottom, devices within a user exchange local samples, but the transition matrices $\mathcal{T}_n^u$ differ in the optimization objective. Compared to the original objective (II; Section 4.2), variance in training time between devices is reduced by introducing an additional standard deviation term into the objective (III; Section 4.2). At the same time, ShuffleFL also applies the adaptive scaling factor in the objective (IV; Section 4.3) for each user, considering the disparity of average device capabilities exists among participating users. In this way, users in ShuffleFL optimize the objective in the bottom row to obtain the transition matrices, resulting in lower wall-clock time and higher global model accuracy in given communication rounds.

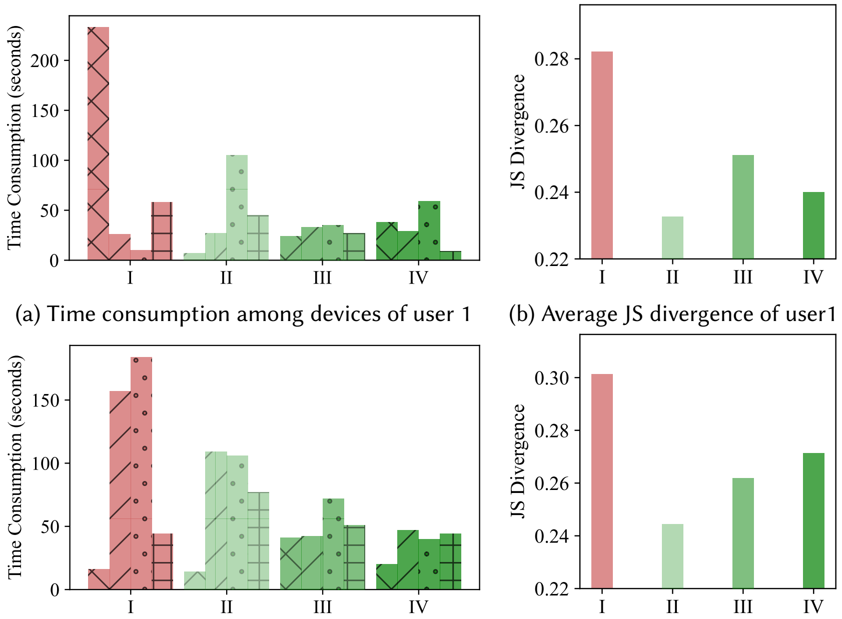
Figure 2 influence data shuffling. The benefits of data shuffling, following matrices $\mathcal{T}$ in Equation (2) (Figure 3 II), are evident compared to the performance of devices without data shuffling (Figure 3 I), reflected in significant relief in training latency and local data imbalance. Yet, the time consumption varies a lot across devices within a user. To mitigate this variance, an additional term is introduced into the objective (Figure 2 III), resulting in further reductions in wall-clock time, as depicted in Figure 3 III. This improvement stems from a more balanced time distribution among devices. However, this enhancement entails a slight decrease in model accuracy. This occurs because data shuffling by $\mathcal{T}$ tends to optimize the SL-related terms more than the DI term, causing a less balanced local data distribution compared to the previous objective (Equation (2)).

4.3 Inter-User Optimization

A gap persists between achieving intra-user optimal data shuffling and the overarching goal of attaining high time-to-accuracy performance for the global model, primarily attributed to the presence of user stragglers. We observe that the average device time consumption varies a lot between users due to the dynamic device capabilities (including communication bandwidth, computing power, and local sample volume). As a result, applying the same trade-off strategy between DI and SL, controlled by coefficient α, to all users is ill-suited. Therefore, we upgrade the optimization objective by personalizing the scaling factor α_u for individual users (Figure 2 IV). At the beginning of each communication round n, the server sends all participating users $\mathcal{U}^n$ adaptive α_u to guide the importance of DI and SL terms they should focus on during the optimization. In this way, the optimization objective, $\forall u \in \mathcal{U}^n$, becomes

$$\mathcal{T}_n^{u*} =_{\mathcal{T}_n^u} \left\{ \max\{SL_n^v\}_{v\in\mathcal{V}^u} + \operatorname{std}\{SL_n^v\}_{v\in\mathcal{V}^u} + \alpha_u \max\{DI_n^v\}_{v\in\mathcal{V}^u} \right\}. \tag{7}$$

To obtain the suitable factor $\{\alpha_u\}_{u\in\mathcal{U}_n}$ without violating the privacy constraints, the server retrieves the history wall-clock training time of users in current communication round, denoted as $\mathbf{H}_n = [h_1, \cdots, h_{|\mathcal{U}_n|}] \in \mathbb{R}^{|\mathcal{U}_n|}$. Each element h_u represents the time interval from when user u downloaded the global model from the server to the time it offloaded the locally aggregated model to the server during its last participation in the FL task. However, the performance of the last participant may be stale for predicting user performance in the current round, as the computing power and communication bandwidth of devices may have changed during this duration. To cope with this, users are also required to report the difference in average device capabilities between the last



(a) Time consumption among devices of user 1

(b) Average JS divergence of user1

(c) Time consumption among devices of user 2

(d) Average JS divergence of user 2

Fig. 3. The time consumption and average JS divergence of devices belonging to two users in a communication round of FL. I, II, III, and IV denote the different optimization objectives in Figure 2, respectively. ***Distinct patterns of bars represent the different devices in the corresponding user.***

participating stratus and the current status, that is,

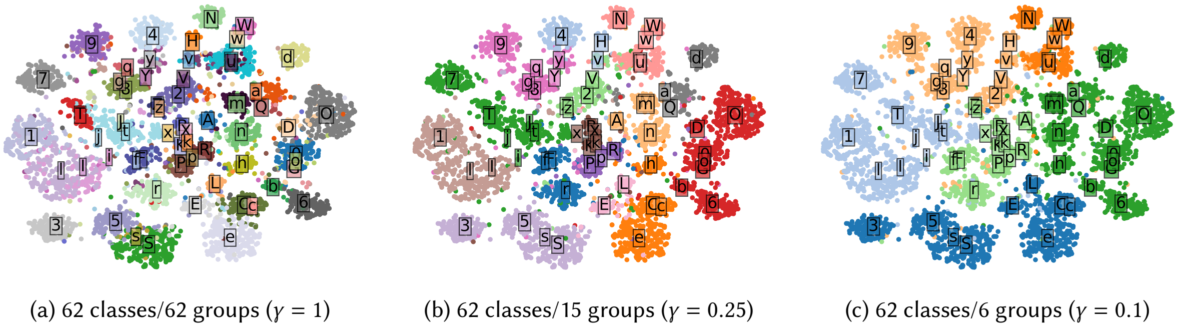
$$\beta_u = \kappa_n \times \frac{\bar{p}^u_{n'}}{\bar{p}^u_n} + (1-\kappa_n) \times \frac{\bar{b}^u_n}{\bar{b}^u_{n'}}, \tag{8}$$

where n' denotes the last communication round in which user u participated. The average computation and communication capabilities across devices within user u are denoted by $\bar{p}^u = \frac{1}{|\mathcal{V}^u|}\sum_{v\in\mathcal{V}^u} p^v$ and $\bar{b}^u = \frac{1}{|\mathcal{V}^u|}\sum_{v\in\mathcal{V}^u}\frac{b^v_\uparrow + b^v_\downarrow}{2}$. From Equations (4), (5), and (6), the time latency $t \propto \bar{p}$ and $t \propto \frac{1}{\bar{b}}$ with distinct coefficients. The coefficients κ and $1-\kappa$ scale the impact of computation and communication capabilities on time latency, depending on the training and transferred samples respectively, which can be approximated to

$$\kappa_u = \frac{3\times\sum_{v\in\mathcal{V}^u}|\mathcal{D}^v_n|}{3\times\sum_{v\in\mathcal{V}^u}|\mathcal{D}^v_n| + \sum_{i=1}^{c}\sum_{v\in\mathcal{V}^u}\sum_{\substack{v'\in\mathcal{V}^u\\ v'\neq v}}\mathbf{T}^v_{n'(i,v')}}. \tag{9}$$

The estimated performance of users in the current round is denoted as $\tilde{\mathbf{H}}_n = [\tilde{h}_1, \cdots, \tilde{h}_{|\mathcal{U}_n|}] \in \mathbb{R}^{|\mathcal{U}_n|}$, where $\tilde{h}_u = \beta_u h_u$. Leveraging $\tilde{\mathbf{H}}_n$, the server executes user-specific balancing through an adaptive scaling factor

$$\alpha_u = \frac{\bar{\mathbf{H}}_n}{\tilde{h}_u} \times \alpha, \tag{10}$$


(a) 62 classes/62 groups ($\gamma = 1$)

(b) 62 classes/15 groups ($\gamma = 0.25$)

(c) 62 classes/6 groups ($\gamma = 0.1$)

Fig. 4. The t-SNE embedding of FEMNIST data instances encompassing 62 classes: (a) each colored cluster represents the latent features of data in the same class mapped to a 2-dimensional space; (b) and (c) apply the dimension reduction on transition matrix **T** at different shrinkage ratio γ, merging the adjacent clusters into $\lfloor 62\gamma \rfloor$ groups (each represented by a unique color). In practice, devices within a user shuffle the local data by groups, which brings the benefit of reducing the computational complexity when optimizing **T**.

where $\bar{\mathbf{H}}_n = \frac{1}{|\mathcal{U}_n|} \times \sum_{u' \in \mathcal{U}_n} \tilde{h}_{u'}$ is the average time latency across users in current round, and α is a constant scaling factor. Users with lower training latency $\tilde{h}_u$ are assigned larger scaling factors α_u, signifying a greater emphasis on the *DI* term in the optimization objective.

By substituting Equation (10) into Equation (7), we can upgrade the final optimization objective to incorporate both intra-user and inter-user perspectives. As shown in Figure 3a, 3c III, discernible disparities exist in the time consumption between two users. Specifically, user 1 acts as the front-runner, while user 2 exhibits characteristics akin to a straggler. Following the incorporation of global inter-user guidance to calibrate optimization emphasis between *DI* and *SL* terms, user 1 prominently directs efforts toward optimizing data imbalances (Figure 3b IV), while user 2 allocates considerable attention to mitigate the system latency (Figure 3c IV).

4.4 Reducing the Optimization Complexity

However, *the computational complexity of solving the optimization problem increases along with the dimensions of the variable* $\mathcal{T}$ where each element **T** is shaped by the number of classes c in the task. In certain classification tasks, such as image classification and speech recognition, the value of c may be notably large. For example, the FEMNIST dataset comprises 62 classes, including 26 lowercase letters, 26 uppercase letters, and 10 digits. In such cases, the computational time required for optimizing $\mathcal{T}$ for each user in each communication round becomes impractical and resource-intensive. We will present how to solve this problem in the next subsection.

To reduce the complexity of finding the optimal transition matrix for the inter-user data shuffling, we further design in ShuffleFL a dimension reduction strategy parameterized by shrinkage ratio γ to alleviate the computational burden on users with varying computational resources. To achieve this, we employ t-SNE [52] to embed the hidden features of the data instance in each class into a 2-dimensional space as shown in Figure 4 (a). The rationale behind the dimension reduction is that we can class-wise merge adjacent clusters into the same group because the distance between each cluster indicates the similarity of classes. We then use k-mean to aggregate the original cluster from c to $k = \lfloor c\gamma \rfloor$ groups, as shown in Figure 4 (b) and (c). The reduction ratio can be adjusted with the varying computing power of users. In this way, the shape of the transition matrix $T_n^v \in \mathbb{R}^{c \times |\mathcal{V}^n|}$ is squeezed to $\tilde{T}_n^v \in \mathbb{R}^{\lfloor c\gamma \rfloor \times |\mathcal{V}^n|}$. When performing the data shuffling, devices exchange samples by randomly selecting data instances in the same group.

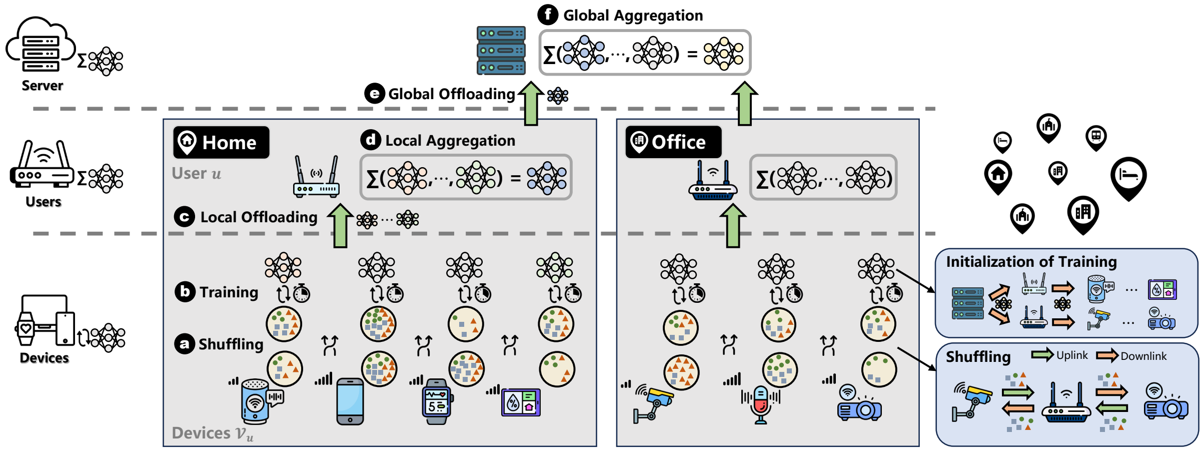


Fig. 5. ShuffleFL hierarchical architecture and training pipeline: it first bootstraps the learning process by exchanging the local samples among devices owned by the same user. This step aims to balance the training time and local data distribution, considering the varying computing power and the non-IID nature of local data across these devices; each device then executes local SGD on the shuffled data for updating the local models, periodically offloading the local model weight parameters to its respective user for local aggregation. The cloud server takes a further aggregation to generate a global model, which serves as the initialization for the local models of devices in the next communication round.

However, the utilization of class-wise merging inherently introduces bias when evaluating the balance level of device data, inevitably influencing the model performance. To analyze the trade-off between the optimization time cost and model performance, we conduct a case study to investigate the impact of optimization complexity reduction, as detailed in Section 6.3. The detailed optimization time, whether using optimization complexity reduction or not, is provided in Appendix A.3.

4.5 Final Training Pipeline of ShuffleFL

Now we introduce the final training pipeline of ShuffleFL. The detailed system architecture of our ShuffleFL is shown in Figure 5. In round n, the cloud server randomly selects a subset of active users $\mathcal{U}_n \subseteq \mathcal{U}$ as in traditional FL frameworks. Then, the participating users first coordinate the data shuffling amongst devices in the same group (Figure 5 ⓐ). After the data shuffling, users assign the current global model (downloaded from the cloud server) to the afflicted devices. Devices perform local stochastic gradient descent (SGD) to update the local model f_{w^v} with learnable parameters w^v in parallel (Figure 5 ⓑ):

$$\mathcal{L}(f_{w^v_\tau}, \xi^v_\tau) = \frac{1}{|\xi^v_\tau|} \sum_{x \in \xi^v_\tau} l(f_{w^v_\tau}; x), \tag{11}$$

$$\hat{w}^v_{\tau+1} = w^v_\tau - \eta_\tau \nabla \mathcal{L}(f_{w^v_\tau}, \xi^v_\tau), \tag{12}$$

where τ indexes the local SGD iteration on the objective $\mathcal{L}(\cdot)$ regarding the loss function $l(\cdot)$ (e.g., cross-entropy loss). ξ^v_τ is a batch of samples randomly selected from local data $\mathcal{D}^v$ at iteration τ. We obtain the updated local model weights $\hat{w}^v_{\tau+1}$ after one-step gradient descent with the learning rate η_τ. Devices update the local model iteratively, obtaining the updated local model $\hat{w}^v_E$ after $E \in \mathbb{Z}^+$ local epochs, where each epoch comprises several steps of local SGD.

Algorithm 1 ShuffleFL.

```
1:  Require: Sever $S$; Users $\mathcal{U}$; Devices $\mathcal{V}$; Device computing power $p^v$, uplink bandwidth $b_\uparrow^v$, downlink bandwidth $b_\downarrow^v$, local data $\mathcal{D}$ in the distribution $\mathbf{D}^v$, $\forall v \in \mathcal{V}$; shrinkage ratios $\Gamma$.
2:  for each round $n = 1, \cdots, N$ do
3:      $\mathcal{U}_n \leftarrow$ randomly select participating users from $\mathcal{U}$
4:      $\{\kappa_u\}_{u \in \mathcal{U}_n} \leftarrow$ users report the staleness factor to server $S$ following Equation (9)
5:      $\{\alpha_u\}_{u \in \mathcal{U}_n} \leftarrow$ server $S$ assigns scaling factor to users following Equation (10)   ▷ Adaptive Scaling Factor
6:      for $u \in \mathcal{U}_n$ in parallel do
7:          $\gamma_u \leftarrow$ select shrinkage ratio from $\Gamma$
8:          $\mathcal{T}_n^u \in \mathbb{R}^{\lfloor c\gamma_u \rfloor \times |\mathcal{V}^n|} \leftarrow$ initialize transition matrices   ▷ Dimension Reduction
9:          $\mathcal{T}_n^{u*} \leftarrow$ user obtain transition matrices by optimizing Equation (7)
10:         $\{\hat{\mathbf{D}}_n^v\}_{v \in \mathcal{V}^u} \leftarrow$ devices exchange data   ▷ Data Shuffling
11:         for $v \in \mathcal{V}^u$ in parallel do
12:             for each epoch $e = 1, \cdots, E$ do   ▷ Local Training
13:                 $\hat{w}_e^v \leftarrow$ local SGD following Equation (12)
14:             end for
15:         end for
16:         $\bar{w}_n^u \leftarrow \sum_{v \in \mathcal{V}^u} s^v \hat{w}_E^v$   ▷ Local Aggregation
17:     end for
18:     $\bar{w}_n \leftarrow \sum_{u \in \mathcal{U}_n} s^u \bar{w}_n^u$   ▷ Global Aggregation
19: end for
```

The device-user and user-server apply the same model aggregation mechanism (e.g., FedAvg). More specifically, user $u \in \mathcal{U}_n$ aggregates the updated models from devices (Figure 5 ⓒ and ⓓ) and the central server periodically receives the aggregated model weight parameters from the participating users $\mathcal{U}_n$ and takes a further aggregation for a global model (Figure 5 ⓔ and ⓕ). At the end of each communication round, the server generates the new global model $\bar{w}_n$:

$$\bar{w}_n^u = \sum_{v \in \mathcal{V}^u} s^v \hat{w}_E^v, \tag{13}$$

$$\bar{w}_n = \sum_{u \in \mathcal{U}_n} s^u \bar{w}_n^u, \tag{14}$$

where s^v and s^u are the weight of the model update during local aggregation and global aggregation, respectively, satisfying $\sum_{v \in \mathcal{V}^u} s^v = \sum_{u \in \mathcal{U}^n} s^u = 1$.

Algorithm 1 describes the entire process of ShuffleFL. In each communication round, the server receives the staleness factor from all participating users, which indicates the changes in the average capabilities of devices between the last joined round and the current round (row 4). The cloud server, considering the reported staleness combined with the retrieval history wall-click time of users, provides each user with a suitable scaling factor to have a trade-off between data distribution and training efficiency during optimization, thereby balancing the user-wise training latency from a global view (row 5). Stepping into procedures for users, each user first decides the reduction ratio for the dimensions of the transition matrix, matching the user's computing power. Devices under the user first shuffle the local samples following the optimal transition matrices (rows 7-10), and then go through multiple iterations to update the model on the shuffled local dataset in parallel (rows 11-16). The updated model weight parameters are aggregated layer by layer in a synchronous way, and we obtain the global model on the cloud server (row 18).

Table 1. A summary of the datasets used in our experiments.

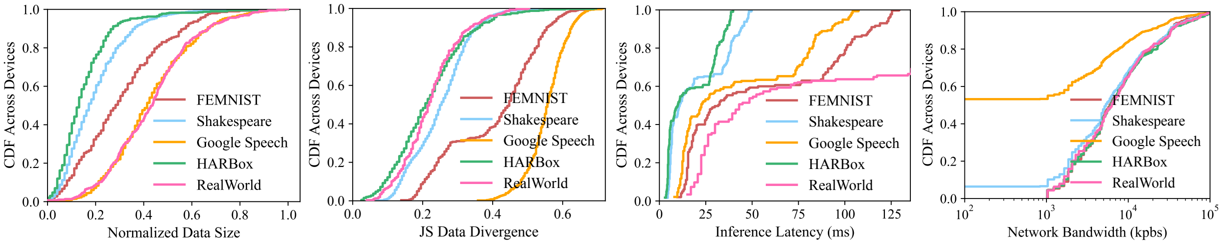
Dataset	Modality	Task	#Classes	#Devices	#Users	#Samples
#1 FEMNIST [5]	Image	Image Classification	62	300	100	34483
#2 Shakespeare [41]	Text	Next Character Prediction	80	500	100	311339
#3 Google Speech [58]	Audio	Key Word Sensing	35	1000	250	32038
#4 HARBox [43]	IMU	Human Activity Recognition	5	400	121	34115
#4 RealWorld [50]	IMU	Human Activity Recognition	8	252	45	215038

5 EXPERIMENTAL SETUP

5.1 Tasks, Datasets, and Models

We consider four IoT applications with different scales to evaluate the performance and generality of our proposed ShuffleFL across different tasks, datasets, and ML models.

- **Application #1: Image classification.** With the increasing computation capabilities on devices, image classification has become a typical task on mobile devices. FEMNIST [5] is a handwritten character recognition dataset grouped by writers. Hence, we can naturally distribute one writer's image to a user. In this application, we sample 100 writers and each user corresponds to a character writer. Here, we utilize a lightweight ConvNet [6, 36, 40, 46] consisting of two convolutional layers following with two fully connected layers as the task-specific model.
- **Application #2: Natural language processing.** Shakespeare [41] dataset is a next-character prediction task, commonly applied in mobile devices, e.g., text auto-completion in the virtual keyboard. Shakespeare dataset is built from *The Complete Works of William Shakespeare* by each speaking role's dialogues in each play. We sample dialogues from 100 speaking roles for the model training. An RNN constructed by an 8-D encoder, including two LSTM layers and one fully connected layer, is deployed in the FL processing.
- **Application #3: Speech recognition.** We use the Google Speech dataset [58], an audio dataset for spotting command keywords in IoT applications. The dataset contains extensive utterances of 1-second duration each, corresponding to 35 command keywords (e.g., numbers from *zero* to *nine, up, down, stop, go*, etc.). We sample audio data from 250 speakers for training. Then, we also evaluate the dataset with a lightweight model [63, 64] for a 35-class keyword spotting task.
- **Application #4: Human activity recognition.** Human Activity Recognition (HAR) offers appealing features for smart service using the data collected from various onboard sensors. This application is developed to provide various activity predictions for users based on data generated from their multiple devices. In this work, we use the HARBox [43] dataset for human activity recognition. HARBox collects 9-axis IMU data from 121 individuals, to distinguish daily activities: *walking, hopping, phone calls, waving, and typing*. A sliding window of 2 seconds at 50 Hz is used to deliver a 900-dimension feature for all 34115 data samples [29, 43]. We use data from 100 users for training, with the remaining 21 user data used for testing. Besides, we also validate ShuffleFL in a real-world dataset. RealWorld [50] includes data from 15 participants performing 8 activities: jumping, lying, standing, sitting, running, walking, climbing down, and climbing up. 7 IMU-enabled devices, strategically placed on the user's body—comprising the head, chest, upper arm, waist, forearm, thigh, and shin—simultaneously capture accelerometer and gyroscope data at a sampling rate of 50 Hz. We partition the data of 15 users into 45 users. Among them, 36 users are allocated for training, and 9 users are designated for testing. Detailed data processing procedures and rationale can be found in Appendix A.2. Considering the simplicity of the dataset and task, we employ a shallow CNN [28, 29, 43] model to recognize human activities.



(a) Unbalanced data size (b) Heterogeneous distribution (c) Diverse computation (d) Diverse communication

Fig. 6. Distribution of the heterogeneous data and system capacities across the devices.

The detailed information of these datasets is summarized in Table 1. We divide each dataset mentioned above into two parts: server-side test data for evaluating the global model performance and user-side data for local training. Regarding the training data, despite some datasets (i.e., FEMNIST, Shakespeare, and Google Speech) having a larger number of participants, we sample 100-250 individuals with sufficient data as users. Detailed justifications can be found in Section 8. For the RealWorld dataset, each user possesses 7 devices, corresponding to the placement of 7 devices on the user's body. To simulate the multi-device setting for users in all datasets, we further partition and assign the user data across 3-6 devices as device local data for various applications, except for the image classification and human activity recognition tasks, we use 2-5 devices for the user data partition.

5.2 Heterogeneity

Data heterogeneity. We employ Dirichlet distribution [16, 30, 64] parameterized by a vector $q \in \mathbb{R}^c$, where $q \backsim Dir(\mu q)$ to partition user data across client devices. Specifically, users first select the number of devices to partition within a specified range (as mentioned above), and then the user data can be distributed to these data following a Dirichlet distribution. For the inherent multi-device RealWorld dataset, we directly utilize its device data as local data. Detailed processing can be found in Appendix A.2. A smaller concentration parameter μ leads to a more diverse distribution from the prior class distribution q. We profile the non-IID data across devices after tightly coupled with inherent non-IID from the user level. In each dataset, we see a high statistical deviation across devices not only in the number of samples (Figure 6a) but also in the data distribution (Figure 6b). Note that a smaller JS divergence indicates a greater imbalance in the data distribution. In this paper, we set μ=0.5.

System heterogeneity. To approach the real-world heterogeneous system in emulation, we acquire the local computation time of deep learning models across hundreds of device types from AI benchmark [19]. Figure 6c illustrates the distribution of computation efficiency across devices. The slowest device would take around 13× computational time than the fastest for the same task. For real mobility traces, we use Tracer dataset [17], which captures the daily smartphone resource conditions of 100 participants over 6 weeks. In this work, we assign the value from the AI benchmark as base computation time to devices to emulate real computing capacities [25, 26, 29]. Then we use the normalized computing capabilities of the Tracer as coefficients to achieve a real mobility pattern.

Settings of the communication links. MobiPerf [18] is a public dataset for measuring network performance on mobile devices, which collects the available cloud-to-edge network throughput of over 100k worldwide mobile clients. As shown in Figure 6d, the best communication channel can be 200× better than the worst one. To approach the dynamic communication ability of devices, we design a coefficient ω as follows, with $x \sim N(1, 0.2)$:

$$\omega = \begin{cases} 1, & x \leq 0 \\ x, & 1 \leq x \leq 2 \\ 2, & x \geq 2 \end{cases} . \tag{15}$$

Table 2. Embedded processor profiles with average optimization time (in seconds) per user across five datasets.

User Type		#1 FEMNIST		#2 Shakespeare		#3 Google Speech		#4 HARBox	#4 RealWorld
		w/ reduction	w/o	w/ reduction	w/o	w/ reduction	w/o		
Raspberry Pi 4 (Model B)		28.69	4197.60	25.16	3173.12	24.01	556.57	9.89	25.26
Jetson Nano	Mode 1	32.17	6609.90	28.35	4466.70	28.91	1308.85	15.67	41.51
	Mode 2	47.78	13817.93	49.05	11409.10	39.30	2035.22	24.49	53.60
Jetson TX2	Mode 1	27.65	5955.94	29.79	4178.05	23.75	1215.29	11.76	30.05
	Mode 2	44.00	9457.70	44.59	5981.51	34.90	1559.64	19.11	49.47
Jetson Xavier NX	Mode 1	19.77	3596.98	25.46	2715.40	18.26	246.11	8.57	16.20
	Mode 2	24.73	4683.95	31.29	3315.14	26.75	723.53	10.51	35.54

To this end, we randomly allocate the communication rates for devices based on the MobiPerf dataset and then apply the small disturbance (i.e., ω) to emulate dynamic real-world connectivity for each communication round.

Computing capability at the user. To quantify the computing capability of users, we measure the optimization time –spent on calculating the suitable data transfer matrix across users– on 4 embedded devices with 7 different profiles for each dataset (see Table 2). These 7 processors in our experiments simulate various users, aiming to approach real-world scenarios. Given the simplicity of HARBox and RealWorld datasets, characterized by basic 5-class and 8-class classification tasks, we only provide optimization time results without optimization complexity reduction. In our experiments, users employ varying numbers of devices, and we report the average time consumption per user without loss of generality. Note that when evaluating the time overhead of ShuffleFL, we use the exact optimization time of each user, rather than the average time consumption. Realistic device profiles also demonstrate that implementing category dimension reduction for high-category tasks significantly reduces the time spent optimizing user heterogeneity. To balance optimization times across diverse user computational capacities and ensure the efficacy of FL, we adopt varying shrinkage ratios γ for different device profiles. More details can be found in Appendix A.3.

5.3 Baselines

To comprehensively evaluate the ShuffleFL performance, we compare ShuffleFL with six baselines. Among them, baselines (2)-(4) concentrate on developing diverse device selection policies to counter statistical and system heterogeneity and bring a large speedup in time-to-accuracy performance for FL. Baselines (5)-(6) adopt the hierarchical framework similar to the proposed method in this paper to improve the overall performance of FL.

1. **FedAvg** [41]: a classic FL method with applications in commercial products [3]. Devices conduct SGD on local data and offload model updates to the central server for aggregation to obtain a new global model.
2. **Oort** [26]: develops a device selection strategy by profiling the device utility by taking into account both statistical and system utilities.
3. **Pyramid** [29]: builds upon Oort by additionally incorporating adaptive local training and model update dropout to enable more refined device selection.
4. **FLAME** [8]: modifies the time utility of Oort and introduces an energy utility to construct device utility. To make a fair comparison, we provide FLAME results by considering its statistical and time utilities in our experiments.
5. **HierFL** [39] introduces a client-edge-cloud framework to achieve high communication efficiency by optimizing two aggregation intervals, i.e., client-edge aggregation interval and edge-cloud aggregation interval.

(6) **FedGS** [35] capitalizes on naturally clustered factory devices, selectively choosing a subset of devices within each factory. It then constructs homogeneous super nodes to establish an end-edge-cloud hierarchical FL framework.

5.4 Hyperparameter Settings

Unless specified otherwise, our experiments are conducted with the settings described as follows. For baselines and ShuffleFL, we set the number of communication rounds to 150, 300, 500, 200, and 300 for five datasets. The minibatch size for local training is 16 in speech recognition and 32 in other applications. Each device performs 5 local training epochs. Note that for the hierarchical baselines, we maintain the same number of local iterations with other methods while adjusting the frequency of both edge aggregation and server aggregation. The results we report are based on using the optimal settings. We use Adam as the optimizer for local updates with a learning rate of 0.001 and a momentum of 0.9. The user participation ratio in each round is set at 0.2. These configurations are consistent with those reported in the literature [14, 26]. In addition, we align baselines with the base model, training settings, and data configurations utilized in ShuffleFL for each application, while adhering to hyperparameter settings provided by each baseline method except FLAME. Given that FLAME is purpose-built for human activity recognition, we finetune its critical hyperparameters to ensure optimal performance when extending to other applications. Details of this fine-tuning process can be found in Appendix A.1.

6 EVALUATION

In this section, we conduct extensive experiments to evaluate ShuffleFL on four FL applications using two metrics: **1) Inference accuracy:** defined as the model convergence accuracy learned through FL, evaluated on server-side test dataset; **2) Time-to-accuracy:** defined as the time consumption for FL training to reach the target accuracy. Specifically, the target accuracy for each dataset is set to be achievable for all baselines. Our key results are:

- ShuffleFL outperforms various FL baselines, achieving an inference accuracy improvement of up to 5.28%, 4.63%, 5.75%, 2.81%, and 7.85% [1] across five datasets, respectively. Besides, ShuffleFL speeds up convergence to a target accuracy. When we define the lowest final accuracy among baselines as the target accuracy, ShuffleFL accelerates the convergence by up to 6.96×, 6.49×, 4.11×, 9.29×, and 36.56×. (Section 6.1)
- Each component of ShuffleFL matters. By jointly considering each key component, ShuffleFL can guarantee the full exploration of data and system heterogeneity for each user and adapt to the various computing abilities of gateways via complexity reduction along the data category. (Section 6.2)
- ShuffleFL achieves superior performance while maintaining robustness to varying sensitivity checking, including the impact of the number of users selected in each round, the impact of different communication capacities, and the impact of optimization complexity reduction along the data category. (Section 6.3)

6.1 Overall Performance

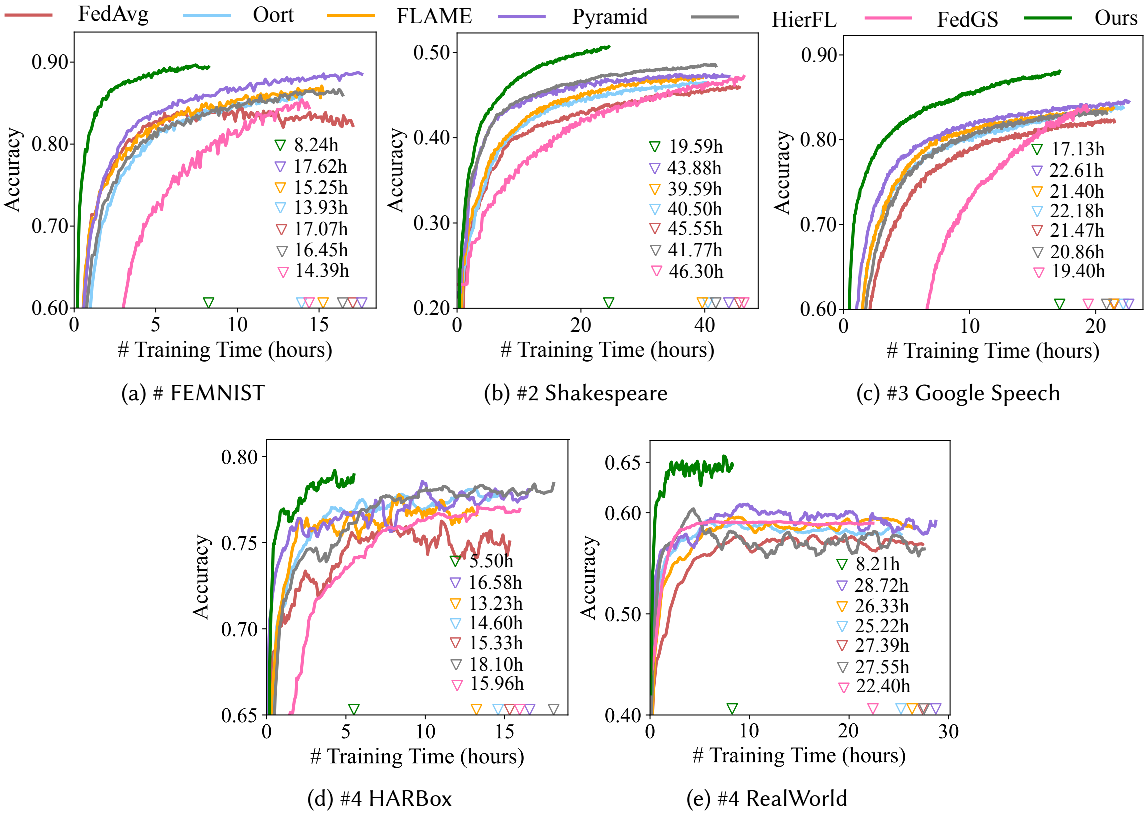
We evaluate the overall performance of ShuffleFL with six baselines on all five datasets.

ShuffleFL improves the time-to-accuracy performance. Figure 7 illustrates the evolving performance of ShuffleFL and the baselines over time, with communication rounds fixed at 150, 300, 500, 200, and 300 for each dataset. We observe that ShuffleFL substantially reduces the total training time consumption in all applications, outperforming six baselines by a large margin. Specifically, ShuffleFL demonstrates a training efficiency improvement of at least 40.8%, 50.5%, 11.7%, 58.4%, and 63.3% respectively (as compared to the best-performing baseline). These results confirm that the data shuffling scheme among the devices of users, as proposed in this paper, effectively reallocates their data, computing, and communication resources, which greatly alleviates

[1] Unless stated otherwise, the accuracy improvements we report are calculated following the pairwise subtraction of global model inference accuracy between shuffleFL and competing methods.

Table 3. Overall performance of ShuffleFL (ours) against baselines across five datasets. We tease apart the overall improvement with data (i.e., the final inference accuracy) and system ones (i.e., wall clock time and communication rounds to reach the target accuracy.) Note that we set the target accuracy to be the highest achievable accuracy by all approaches. The best performance in each dataset is marked in bold.

Dataset	Baselines	Metric		Gains	
		Acc.	Time	Δ *Acc.*	*Speedup*
# 1 FEMNIST	FedAvg	84.38%	10.30h	-	-
	Oort	86.03%	8.40h	1.65%↑	1.23×
	FLAME	87.08%	6.98h	2.70%↑	1.48×
	Pyramid	88.73%	5.75h	4.35%↑	1.79×
	HierFL	87.44%	7.51h	3.06%↑	1.37×
	FedGS	85.43%	12.47h	1.05%↑	0.83×
	Ours	**89.66%**	**1.48h**	**5.28%**↑	**6.96**×
# 2 Shakespeare	FedAvg	46.05%	44.07h	-	-
	Oort	46.56%	33.47h	0.51%↑	1.32×
	FLAME	47.13%	25.03h	1.08%↑	1.76×
	Pyramid	47.46%	17.91h	1.41%↑	2.46×
	HierFL	48.61%	15.95h	2.56%↑	2.76×
	FedGS	47.20%	38.03h	1.15%↑	1.16×
	Ours	**50.68%**	**6.79h**	**4.63%**↑	**6.49**×
# 3 Google Speech	FedAvg	82.35%	20.45h	-	-
	Oort	83.95%	15.62h	1.60%↑	1.31×
	FLAME	83.76%	13.63h	1.41%↑	1.50×
	Pyramid	84.66%	11.15h	2.31%↑	1.83×
	HierFL	83.39%	15.07h	1.04%↑	1.01×
	FedGS	84.05%	17.08h	1.70%↑	1.20×
	Ours	**88.10%**	**4.97h**	**5.75%**↑	**4.11**×
# 4 HARBox	FedAvg	76.40%	7.71h	-	-
	Oort	78.16%	3.97h	1.76%↑	1.94×
	FLAME	77.80%	2.54h	1.40%↑	3.04×
	Pyramid	78.56%	2.01h	2.16%↑	3.84×
	HierFL	78.44%	5.86h	2.04%↑	1.32×
	FedGS	77.09%	10.11h	0.69%↑	0.76×
	Ours	**79.21%**	**0.83h**	**2.81%**↑	**9.29**×
# 4 RealWorld	FedAvg	57.77%	16.45h	-	-
	Oort	59.22%	4.67h	1.45%↑	3.52×
	FLAME	59.58%	5.21h	1.81%↑	3.16×
	Pyramid	60.86%	3.50h	3.09%↑	4.70×
	HierFL	60.39%	2.68h	2.62%↑	6.14×
	FedGS	59.12%	2.71h	1.35%↑	6.07×
	Ours	**65.62%**	**0.45h**	**7.85%**↑	**36.56**×



(a) # FEMNIST

(b) #2 Shakespeare

(c) #3 Google Speech

(d) #4 HARBox

(e) #4 RealWorld

Fig. 7. The time-to-accuracy on different datasets. Triangles of distinct colors represent the total training time for each method during communication rounds of 150, 300, 500, 200, and 300 for the respective datasets.

the straggler issue of FL in the heterogeneous environment. Besides, Table 3 shows the training time consumption needed to converge to the corresponding target accuracy, i.e., the highest achievable accuracy by all baselines, which turns out to be FedAvg accuracy. ShuffleFL reaches the target accuracy 6.96×, 6.49×, 4.11×, 9.29×, 36.56× faster for each dataset. Given that the data distribution of users in the HARBox and RealWorld datasets is low non-IID, indicating a relatively balanced data distribution [8, 29], there is ample room for data shuffling among devices to address data imbalance and expedite model training. As a result, the most significant speedup, reaching 9.29× and 36.56×, is observed on the HARBox and RealWorld datasets. This finding is consistent with the above-mentioned speedup results of the overall training time. Additionally, in the evaluation of time-to-accuracy (i.e., time consumption when reaching target accuracy), comparing ShuffleFL with other baselines reveals that Oort requires additional time ranging from 3.14× to 10.39×, FLAME demands 3.06× to 11.57× more time, Pyramid needs 2.25× to 7.78× more time, HierFL requires 2.35× to 7.04× more time and FedGS needs 3.43× to 12.22× more time. Besides, it can be observed that FedGS tends to demonstrate higher time consumption when compared to other methods. This is attributed to FedGS performing edge aggregation for each mini-batch iteration, leading to a significant training latency due to the constrained volume of training data for each communication round.

ShuffleFL improves the final model accuracy. When fixing the communication rounds, ShuffleFL demonstrates improvements in inference accuracy compared to FedAvg by 5.28%, 4.63%, 5.75%, 2.81%, and 7.85% on

Table 4. ShuffleFL improves the efficacy by two pivotal components: the time balance in device-user and the adaptive scaling guided from the server-user global view, alleviating the device straggler issue and user straggler issue.

Baselines	#1 FEMNIST		#2 Shakespeare		#3 Speech		#4 HARBox		#4 RealWorld	
	Δ *Acc.*	*Speed.*	Δ *Acc.*	*Speed.*	Δ *Acc.*	*Speed.*	Δ *Acc.*	*Speed.*	Δ *Acc.*	*Speed.*
w/o time balance	5.63%↑	3.90×	5.05%↑	4.29×	6.27%↑	2.98×	3.06%↑	5.28×	7.47%↑	24.19×
w/o adaptive scaling	2.77%↑	6.56×	3.75%↑	4.77×	4.47%↑	3.37×	1.89%↑	5.59×	4.90%↑	33.57×

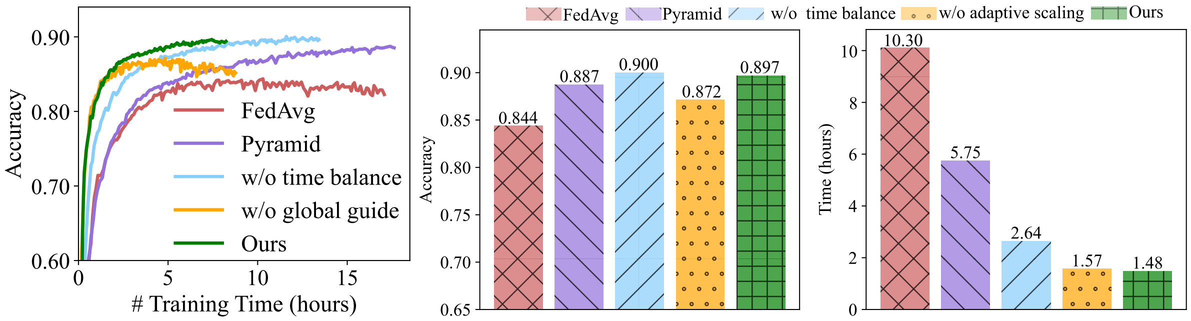
FEMNIST, Shakespeare, Google Speech, HARBox, and RealWorld, respectively. The extent of enhancement in inference accuracy is intricately tied to the prevailing data distribution. By providing the most diverse data distribution (Figure 6b) and the large scale of devices (Table 1), the most substantial improvement in accuracy is observed at 7.85% on the RealWorld dataset. The rationale lies in the great potential of devices with high non-IID to benefit from a relatively balanced data distribution through data shuffling, thereby alleviating the performance degradation induced by data imbalance. In contrast, ShuffleFL achieves marginal accuracy improvements in the HARBox. This phenomenon is likely attributed to the inherent simplicity of the dataset, characterized by a 5-class classification task and the absence of diverse data collected from multiple devices placed in different positions, as observed in the RealWorld dataset. This undoubtedly diminishes the task's complexity, mitigating the negative impact of data heterogeneity on model performance. Consequently, there is limited scope for ShuffleFL to enhance inference accuracy. Compared with other baselines, ShuffleFL increases the inference accuracy by up to 4.23%, 4.12%, 4.71%, 2.12%, and 6.5% across five datasets. We argue that these comparison methods struggle to address the global model bias from non-IID data, while ShuffleFL excels in harmonizing data across devices, achieving a balanced distribution, and significantly enhancing the performance of the global model.

6.2 Ablation Study

Here, we implement two breakdown versions of ShuffleFL to evaluate and understand the effectiveness of each key component incorporated in ShuffleFL. To avoid redundancy, we only report results for two baselines: the classic Federated Learning method, FedAvg, and the top-performing baseline, Pyramid.

- **ShuffleFL w/o time balance.** We remove the time balance term std$\{SL\}$ in the optimization objective (see Equation (7)), which is responsible for balancing the time consumption variance among devices. As such, this adjustment prevents the model from effectively addressing the device straggler issue.
- **ShuffleFL w/o adaptive scaling.** We disable the adaptive scaling that aims to balance the reduction of data imbalance and system latency among devices from a global perspective. To this end, ShuffleFL neglects the dramatically varying time consumed by different users, leading to overfitting in terms of system latency reduction in front-runner users and underfitting for user stragglers.

ShuffleFL greatly relieves the device stragglers issue by incorporating time balance as an optimization term. Table 4 illustrates the accuracy improvement and time consumption reduction achieved by ShuffleFL without adding the time balance constraint for data shuffling among devices, as they reach the corresponding target accuracy. For all four application scenarios, the observed speedup in time is 3.90×, 4.29×, 2.98×, 5.28×, and 24.19× respectively. Checking the wall clock time in Figures 8c, 9c, 10c, 11c, 12c, ShuffleFL without time balance (shown in blue) still outperforms both FedAvg (in red) and Pyramid (in purple) but falls short of the complete ShuffleFL (shown in green). Final accuracy results, as depicted in Figures 8b to 12b, indicate that ShuffleFL without time balance achieves the highest accuracy across five datasets. This outcome is attributed to the relaxation of system latency reduction requirements, allowing for more flexibility in optimizing data imbalance.

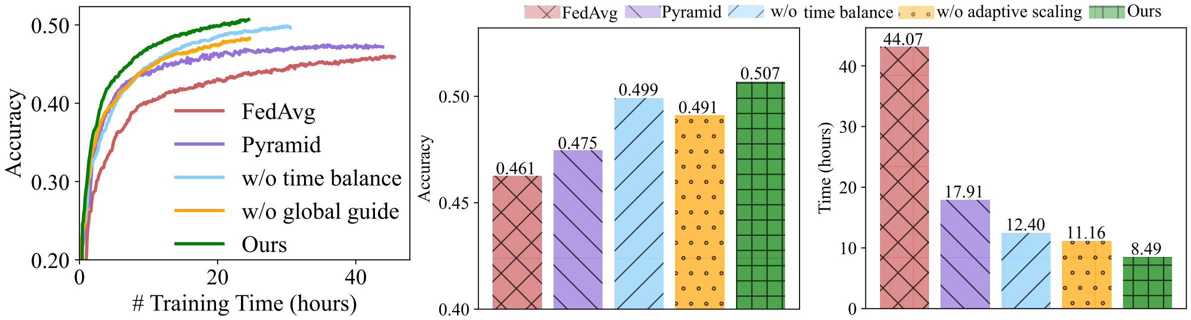


(a) Time-to-Accuracy performance

(b) Inference accuracy

(c) Wall clock time

Fig. 8. The ablation study of ShuffleFL on the FEMNIST dataset. Wall clock time indicates the time consumption when reaching the target accuracy.



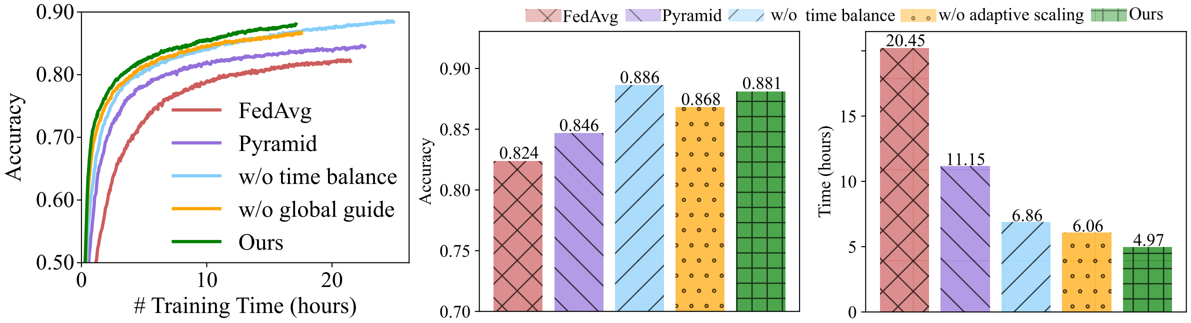
(a) Time-to-Accuracy performance

(b) Inference accuracy

(c) Wall clock time

Fig. 9. The ablation study of ShuffleFL on the Shakespeare dataset. Wall clock time indicates the time consumption when reaching the target accuracy.

ShuffleFL employs an adaptive scaling factor to globally balance data and system heterogeneity, mitigating user straggler issues and enhancing performance. To demonstrate the impact of our adaptive scaling on model performance, Table 4 presents the improvement achieved by ShuffleFL without adaptive scaling compared to FedAvg. The wall clock time notably decreases by up to 6, 56×, 4.77×, 3.37×, 5.59×, and 33.57× across five datasets, which is attributed to the inclusion of the time balance component. However, as depicted in Figures 8b to 12b, there is a severe degradation in inference accuracy. Taking Figure 8b on the FEMNIST dataset as an example, ShuffleFL without adaptive scaling exhibits an improvement in inference accuracy compared to FedAvg by 2.77%. Nevertheless, the inference accuracy is 1.5% lower than that of the best-performing baseline, Pyramid, and significantly trails behind the performance achieved by the complete ShuffleFL (2.5% decrease). While this ShuffleFL without adaptive scaling effectively addresses the device straggler issue, it tends to overly prioritize inter-user optimization, neglecting the time consumption imbalance among users. Therefore, the tendency towards excessive inter-user optimization often leads to an overemphasis on system latency optimization for front-runner users, while insufficient attention is directed towards reducing system latency for user stragglers from a global perspective. Only through the integration of inter-user and intra-user perspectives can improvements be achieved that simultaneously and effectively address both data and system heterogeneity. By providing a global perspective

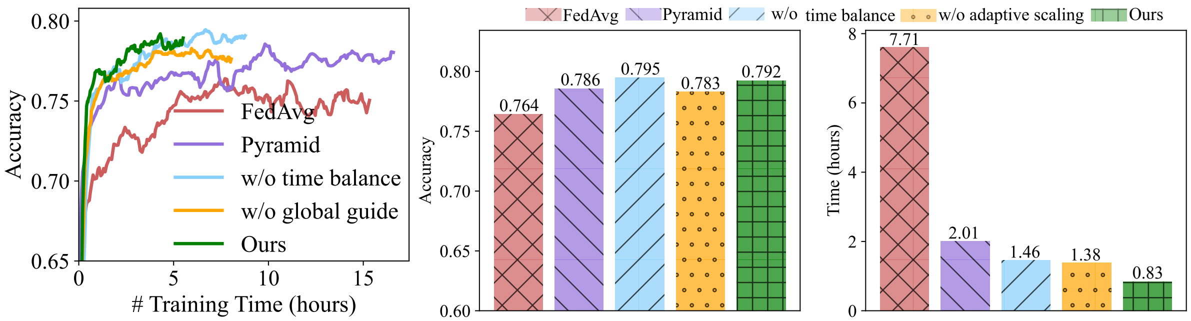


(a) Time-to-Accuracy performance

(b) Inference accuracy

(c) Wall clock time

Fig. 10. The ablation study of ShuffleFL on the Google Speech dataset. Wall clock time indicates the time consumption when reaching the target accuracy.

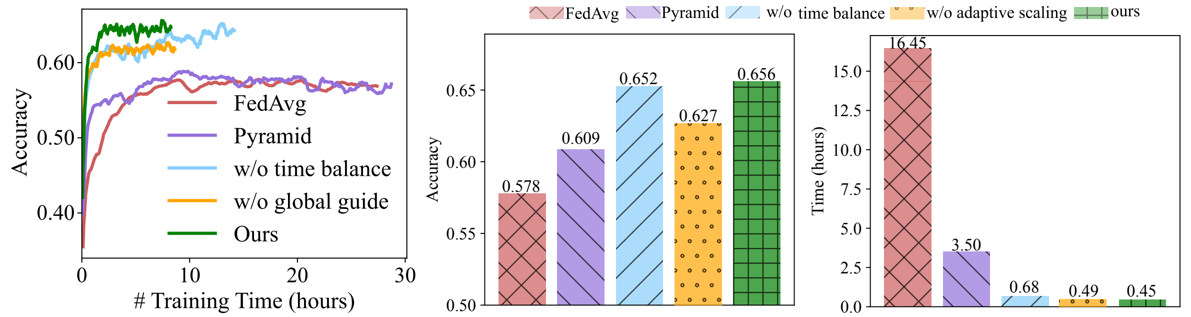


(a) Time-to-Accuracy performance

(b) Inference accuracy

(c) Wall clock time

Fig. 11. The ablation study of ShuffleFL on the HARBox dataset. Wall clock time indicates the time consumption when reaching the target accuracy.

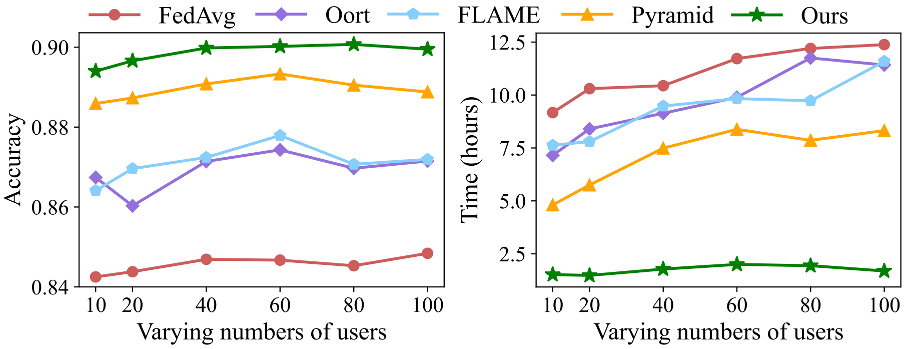


(a) Time-to-Accuracy performance

(b) Inference accuracy

(c) Wall clock time

Fig. 12. The ablation study of ShuffleFL on the RealWorld dataset. Wall clock time indicates the time consumption when reaching the target accuracy.



(a) Inference accuracy

(b) Wall clock time

Fig. 13. The performance of ShuffleFL across varying numbers of users (i.e., $|\mathcal{U}_n|$) on the FEMNIST dataset.

to individual users, front-runners can prioritize enhancing data heterogeneity, while stragglers can focus on optimizing system latency to improve overall performance

6.3 Robustness and Sensitivity Analysis

In this subsection, we analyze the robustness and sensitivity of ShuffleFL under varying active user numbers participating in each round, diverse communication capacities, and different shrinkage ratios.

Varying active user numbers (i.e., $|\mathcal{U}_n|$) in each round. We assess the performance of ShuffleFL on the FEMNIST dataset across varying scales of participants in each round. Specifically, we vary the number of participating users from 10 to 100 to compare the model performance with the baselines. As shown in Figure 13a, the inference accuracy of all methods exhibits an upward trend as the number of participating users increases. ShuffleFL remains robust under different participation ratios and consistently outperforms the other four baselines. Figure 13b showcases the total time cost when the global model reaches the target accuracy with the varying number of participating users. We notice that, for the baselines, the wall clock time significantly increases as more users are involved, which is attributed to the heightened diverse data and system heterogeneity among devices, leading to more severe straggler issues. However, it appears that the training time of the ShuffleFL approach does not increase with the involvement of more users. The reasons for the counterintuitive performance are two-fold: 1) as more users participate in the federated learning task, the volume of training data in a single round increases, which speeds up the convergence of the global model. Consequently, the model requires fewer communication rounds to reach the target accuracy; 2) the objective function with scaling factor α_u in ShuffleFL is designed to balance system latency (SL) and data imbalance (DI). With an increasing number of users, the framework tends to prioritize minimizing system latency. This shift in focus is evidenced by the slight drop in final top-1 accuracy when the number of active users increases from 80 to 100, as depicted in Figure 13a. ShuffleFL consistently achieves a very low time cost, even with a slight increase as the number of participating users grows. The results demonstrate the robustness and superiority of ShuffleFL with varying active user numbers.

Varying communication capacities. Communication time introduced by data shuffling necessitates the investigation on the efficacy of ShuffleFL in scenarios where data shuffling incurs substantial costs, such as transferring high-resolution images. Bringing down the overall communication capabilities of devices imposes an equivalent effect as transferring large-sized raw data, therefore, we systematically vary the base communication bandwidths to approximate such conditions. As shown in Figure 14, we explore three additional scenarios by attenuating the initial base communication capacity (Figure 6d) to 0.1×, 0.01×, and 0.005×. We observe that

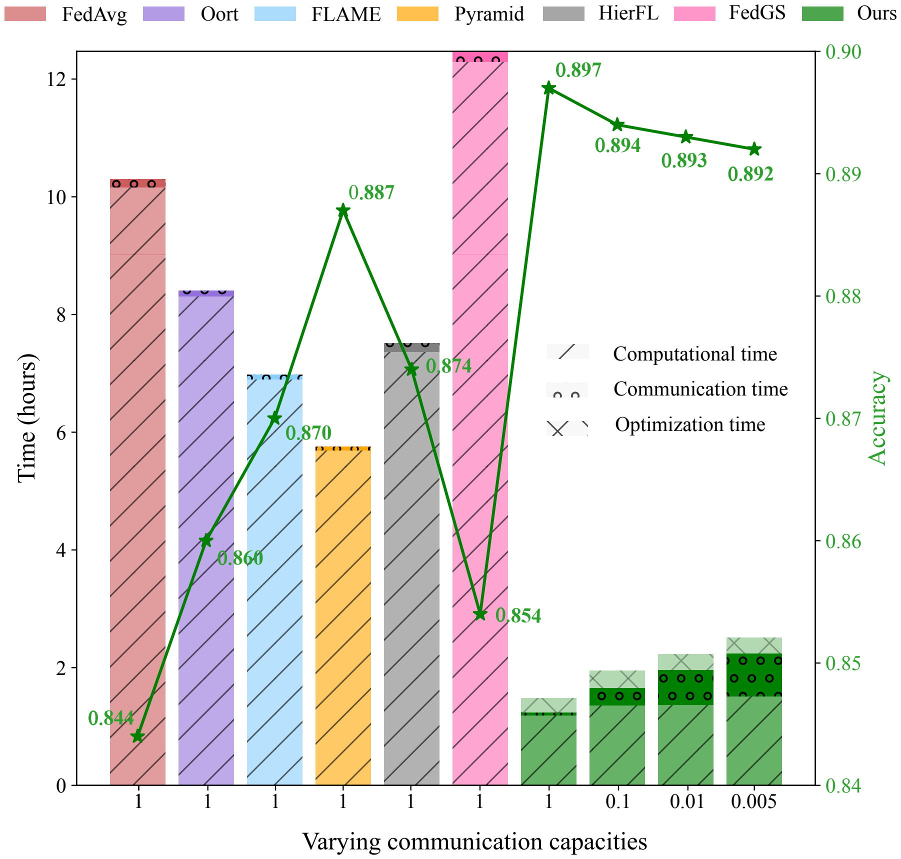
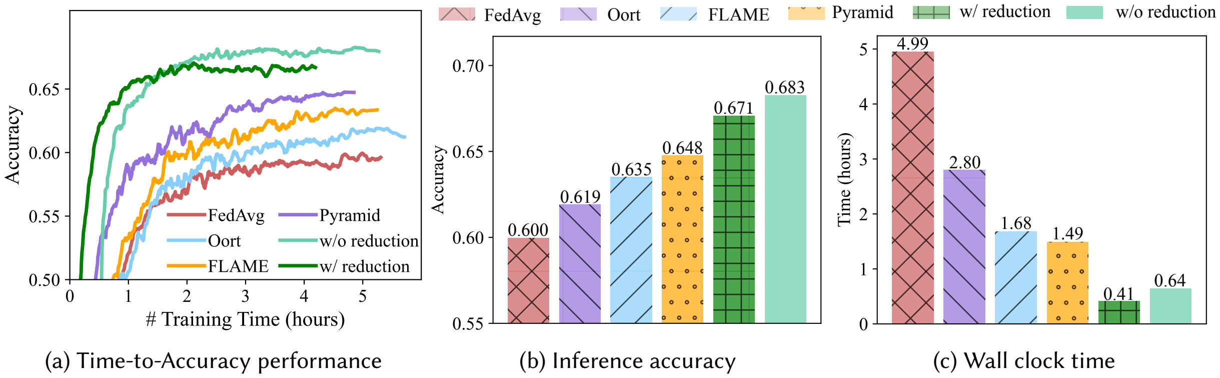


Fig. 14. ShuffleFL shows the superior performance across different communication capacities on the FEMNIST dataset. We additionally consider the other three different levels of communication overheads for our ShuffleFL.

the time latency achieving the target accuracy rises as the reducing communication bandwidths, accompanied by a slight decrease in inference accuracy. However, ShuffleFL still far exceeds the performance of baselines on both metrics. These results reveal that in heterogeneous device-user-server FL applications, despite the additional communication overhead introduced by data exchange, the harmonization of data, communication, and computation resources still provides a clear advantage in promoting FL efficiency.

Additionally, Figure 14 also provides a detailed breakdown of the time consumption required to achieve the target accuracy (defined as the best accuracy obtained by FedAvg) into three components: computing time, communication time, and optimization time. The system communication costs are recorded as follows: 0.14 hours for FedAvg, 0.09 hours for Oort, 0.08 hours for FLAME, 0.06 hours for Pyramid, 0.14 hours for HierFL, 0.18 hours for FedGS, and 0.04 hours for ShuffleFL. Despite the additional communication overhead incurred by data shuffling, communication overhead in ShuffleFL is relatively low since training on the shuffled local data speeds up the global model convergence, as a result, ShuffleFL takes fewer communication rounds to reach the target accuracy. For example, FedAvg needs 91 rounds whereas ShuffleFL achieves the same result in only 29 rounds. Upon reducing the communication capability to 0.1×, 0.01×, and 0.005×, we observe that the communication time of ShuffleFL becomes a more significant component of the total latency. However, ShuffleFL consistently



(a) Time-to-Accuracy performance

(b) Inference accuracy

(c) Wall clock time

Fig. 15. A case study about the performance of ShuffleFL whether using the optimization complexity reduction along the data category on the Google Speech dataset. Regarding ShuffleFL with reduction, we employ four shrinkage ratios — 0.20, 0.25, 0.30, 0.35 — to simulate the diverse ratios adopted by different users with varying computing capacities.

achieves the target accuracy with minimal time latency, even when the overall bandwidth is reduced to as low as 0.005× the reference bandwidth. This is attributed to two factors: 1) the number of rounds for ShuffleFL to the target accuracy shows marginal changes, i.e., 33 rounds, 32 rounds, and 32 rounds for communication capabilities reduced to 0.1×, 0.01×, and 0.005×, respectively; 2) data shuffling also improves the consistency between the computing power and training task of devices, thereby bringing down the overall computational time latency, especially for devices with limited computing budget, often referred to as stragglers.

Varying shrinkage ratios. Applying shrinkage to the data category dimension inevitably introduces bias in the evaluation metric (i.e., JS divergence) for data imbalance. To investigate the trade-off between the optimization time cost and model performance, we conduct a case study on the Google Speech dataset, analyzing the impact of different shrinkage ratios. In this case, data from 20 users is used, involving 3-5 devices per user, totaling 80 devices. This setup is motivated by the fact that the optimization time, without optimization complexity reduction, becomes excessively large, hindering training progress. To this end, we perform a small-scale FL on the Google Speech dataset with a relatively low number of data categories (35 classes). From Figure 15, we can see that ShuffleFL outperforms other baselines on both metrics, regardless of the utilization of reduction. Specifically, ShuffleFL with reduction exhibits a 1.2% decrease in inference accuracy while achieving a total training time speedup of 1.26× (decreasing from 5.27 hours to 4.19 hours) and a time consumption reduction of 1.56× (decreasing from 0.64 hours to 0.41 hours) when reaching the target accuracy, compared to ShuffleFL without reduction. We believe that the advantages of ShuffleFL with reduction in reducing time overhead will become more pronounced when extended to general large-scale scenarios. These results reveal that the optimization complexity reduction strategy proposed in this paper can significantly reduce the time overhead, albeit at a slight cost to accuracy.

To detail the impact of different shrinkage ratios, Figure 16 shows that as the shrinkage ratio increases, both the inference accuracy and the time required to complete a fixed number of rounds for ShuffleFL increase. Specifically, the inference accuracy reaches 65.28%, 65.68%, 66.93%, and 68.31% across shrinkage ratios of 0.20, 0.25, 0.30, 0.35, while total training time (in the light blue bar) is 3.62 hours, 3.96 hours, 4.49 hours, and 4.63 hours, respectively. Time consumption to achieve the target accuracy is determined by changes in both time and accuracy caused by different shrinkage ratios, and there is no clear pattern to illustrate their relationship. For example, with $\gamma = 1$ (indicating no reduction), the time cost is 0.64 hours, slightly higher than $\gamma = 0.35$ with 0.57 hours due to the fewer rounds needed for ShuffleFL without reduction to reach the target accuracy. However, for any shrinkage ratio, the time required is less than that when there is no optimization complexity reduction.

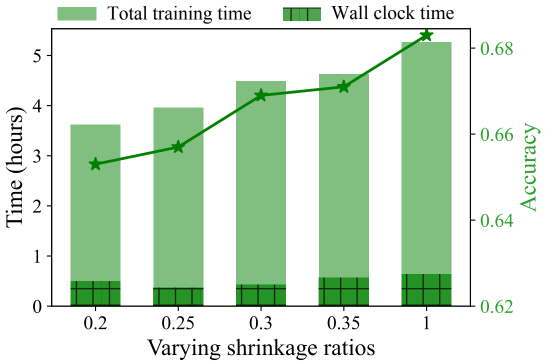


Fig. 16. The performance of ShuffleFL on Google Speech dataset using varying shrinkage ratios (γ). A shrinkage ratio of 1 denotes no reduction applied. The deep blue bars illustrate the wall clock time to achieve the target accuracy, while the light blue bars represent the total time taken to complete a fixed number of communication rounds (i.e., 500 rounds).

7 RELATED WORK

Federated Learning [41] is an emerging privacy-preserving learning paradigm that only requires devices to upload their model updates for collaborative learning, without sharing their local data during the FL training process. The inherent disparity in computing capabilities and data distributions among devices makes FL difficult to obtain a high-quality global model within a limited time, especially with the growing involvement of edge devices. Specifically, intra-device inconsistency between computing resources and sensing capacity (system heterogeneity) inevitably introduces the straggler issue, leading to degradation in FL training efficiency. The inter-device data distribution inconsistency (data heterogeneity) further exacerbates FL, resulting in a significant accuracy decrease.

To abate the impact of data heterogeneity, some methods [2, 12, 22, 30–33, 57] have been proposed to reduce the variance of local updates. For example, FedProx [32] narrows the gap between the global model and local updates by introducing a proximal term into the local loss function. Scaffold [22] corrects client drift with a control gradient variate. Moon [30] proposes to leverage the similarity between model representations to regularize the local training of clients. FedDC [12] dynamically bridges the gap between the local model and the global model with the learned local drift variable. FedDyn [2] introduces the linear and quadratic penalty terms to correct the clients' objectives during local training. Following the idea of reweighing the local updates during the server aggregation, existing work [16, 44, 47, 54, 55] proposed to redesign the global model aggregation to mitigate the objective inconsistency among clients. FedAvgM [16] adds momentum when updating the global model to dampen oscillations caused by the sparse distribution across local data. DynaFed [44] leverages the dynamics of the global model's trajectory to guide global model aggregation. Clustering-based methods [4, 34, 43, 49] group clients according to the similarity of their model parameters. ClusterFL [43] captures the intrinsic clustering patterns among clients by measuring the similarity of client models. [49] proposes a clustered multi-task federated learning on heterogeneous data. In our work, ShuffleFL addresses data heterogeneity by focusing on improving the data distribution among devices, achieved by data shuffling in a device-user-server structured FL.

In addressing system heterogeneity issues, recent research [10, 15, 27, 28] has proposed accommodating local models to varying system capacities, which differs from the conventional FL framework where all clients share the same model architectures. HeteroFL [10] allows heterogeneous clients to select fixed subsets of global parameters with minimal modification to current FL frameworks. FedMask [28] learns a heterogeneous binary mask while freezing the parameters of the local model. FjORD [15] alleviates the problem of client system heterogeneity by tailoring the model width to the client's capabilities. Hwamei [45] proposes a synchronous hierarchical FL scheme, utilizing deep reinforcement learning to allocate aggregation frequencies of cloud and edge serve dynamically. HierFL [39] introduces a client-edge-cloud framework and optimizes client-edge aggregation interval and edge-cloud aggregation interval to achieve high communication efficiency. FedGS [35] takes advantage of naturally clustered factory devices to select a subset of devices within each factory and builds homogeneous nodes to construct an end-edge-cloud hierarchical FL framework. Besides, device selection-based methods [8, 17, 26, 29, 42, 48, 62] have drawn an increasing amount of attention in FL in response to challenges arising from heterogeneous data distribution and systems. Seminal work [26] and [29] proposed to guide the client selection in each round based on the data distribution and computational efficiency of local clients, which yield superior time-to-accuracy performance than random selection. Work can also be found on reducing the negative impact on training efficacy caused by random client selection criteria. [48] adjusts the client training data by selecting local samples with high statistical utility, thereby enabling clients with various computing resources to complete local training before the same deadline. [42] estimate the time consumption by the resource information reported from each client and select as many clients as possible that would complete the current round within a certain deadline. FedMarl [62] builds the decision environment to execute client selection by jointly optimizing model accuracy, processing latency, and communication efficiency based on multi-agent reinforcement learning. FLAME [8] proposes a user-centered federated framework that enables multiple mobile devices to participate in local model training based on a device selection algorithm using energy-efficient consideration. ContextFL [17] selects the participants by combining current and predicted network states to improve the stability and reliability of the federated learning training for edge computing. Our work solves the straggler issue in heterogeneous systems by dynamically optimizing the alignment of local data with the computing capabilities of each device.

8 DISCUSSION

The privacy relaxation in ShuffleFL, allowing data shuffling amongst the devices belonging to the same user or within the same group, enhances the alignment between computational capabilities and training sample volumes on devices, as well as the balance of local data distribution. We have shown through extensive evaluations that ShuffleFL can significantly improve both system efficiency and data efficiency in FL. However, there are some limitations in the current design of ShuffleFL, which could be improved in future works.

Limited number of users. The constraint of having only 100-250 users across different datasets in our experiments is due to the computational complexity and dataset size. The local training runs on the NVIDIA A10 GPU. The computational complexity escalates as the number of users increases, particularly in the context that each user has several devices. Another limiting factor is the dataset size. Each of the datasets is inherently partitioned by the individuals. We assign samples to users by individuals in the dataset and then divide user data into affiliated devices without replacement. For instance, in the HARBox dataset, which comprises data from 121 individuals, we use the data from 100 individuals for training (i.e., data assigned to users) and the remaining 21 for testing (i.e., data residing in the central server for evaluating the global model performance), thereby the number of users is limited to 100 for the HARBox dataset. In the case of the other datasets (FEMNIST, Shakespeare, and Google Speech), despite having a larger number of participants, we have to sample the individuals with sufficient samples as users. This ensures that each device within a user has an adequate subset of local samples, as a limited size of training data could potentially lead to a degradation in global model performance.

Concept of user-layer. The rationale behind the 'user layer' in ShuffleFL lies in providing a trust zone for inter-device data shuffling, thereby relaxing the constraint of taking all devices as data silos. In this way, the definition of users shows flexibility, which can be understood in a broader sense. For example, the 'user' could represent an individual owning multiple devices, or a local network (e.g., wireless sensor networks) where a group of nodes monitors and records the environmental conditions. This broader interpretation of the 'user' significantly extends the applicability of ShuffleFL beyond the conventional understanding of individual users. Adhering to this user concept, it is clear that ShuffleFL imposes the same privacy constraints on the user layer from the perspective of the central server where data exchange among users is not allowed. Therefore, the data distribution of users remains unchanged in the ShuffleFL framework. Note that the objectives of intra-user data shuffling and inter-user optimization are different: the intra-user data shuffling balances the data residing on devices of the same user without breaching privacy; the inter-user optimization, detailed in Section 4.3, focuses on alleviating system heterogeneity across users by trading off the importance of data imbalance and system latency during the intra-user data shuffling.

Practicality of privacy relaxation. Compared to the conventional FL, our ShuffleFL involves the exchange of raw data across devices belonging to the same users, which may pose privacy risks, especially if the communication link is compromised. Our approach introduces the concept of the 'user layer' to define a logical and practicable scope for local data sharing among multi-device affiliations. These devices, such as iPhones and iWatches belonging to the same user, are generally equipped with robust privacy protection mechanisms. This inherent security in device ecosystems provides a layer of safety for data exchange. Furthermore, the privacy guarantees typically employed in conventional FL, such as differential privacy or encryption methods, are also applicable and effective in the ShuffleFL context.

Communication overheads reduction. One notable challenge introduced by the characteristics of ShuffleFL is the unavoidable increase in communication overhead. As we illustrate in Figure 14, the overall time consumption becomes larger with the overall bandwidth decreasing. To cope with this problem, a potential avenue for future research involves incorporating a penalty term before t_{comm} in Equation (4) when calculating the system latency. This modification aims to mitigate the impact of increased communication overhead on the overall system efficiency. An alternative approach to reduce communication overhead is to enhance the efficiency of transferred samples by selectively exchanging data instances with high influences on model training. This strategy may entail leveraging the eXplainable Artificial Intelligence (XAI) methods [23, 37] for importance sampling.

Hybrid hierarchical FL framework. The practicability of ShuffleFL benefits from two key aspects of compatibility with both common two-layer frameworks and other SOTA methods. Firstly, ShuffleFL is designed to accommodate not just scenarios where devices are directly affiliated with a user, but also hybrid situations where some devices may not share a trust zone for data exchange. In these cases, ShuffleFL remains compatible with the conventional device-server framework, allowing for model aggregation from both users and unaffiliated devices. The hybrid scenarios also necessitate the exploration of an asynchronous aggregation scheme [7, 59, 65] where the central server aggregates model updates as long as users/edge servers upload them. This asynchronous approach could provide a solution to challenges posed by individual devices outside the user-device scenario. Secondly, ShuffleFL shows adaptability to existing methods that address heterogeneity issues by enhancing local training, update offloading, and model aggregation.

9 CONCLUSION

The incongruity of computing power and training data residing on the devices, coupled with the non-IID local data amongst devices, pose significant obstacles to achieving high time-to-accuracy performance in the federation training. This paper introduces the ShuffleFL, wherein devices are allowed to exchange local samples within the same user group. This relaxation of privacy constraints results in local data exhibiting significant alignment with device computing power, fostering a more balanced distribution. Consequently, ShuffleFL brings notable improvements in reducing the total wall-clock time required for training and the global model accuracy. To this end, we formulate the shuffling process with the transition matrices as an optimization problem. The optimization objectives are designed to minimize local data imbalance and the training latency of devices concurrently. Going beyond the intra-user optimization, we also strive to balance wall clock training time between users when designing the optimization objective, further reducing overall training latency while improving global model accuracy. Evaluation based on realistic device profiles from the testbed corroborates the efficacy of ShuffleFL. Our investigation of data shuffling in the hierarchical FL framework provides a viable solution to enhance the time-to-accuracy performance of the FL system from the root, offering tangible evidence of the benefits it brings to ubiquitous computing applications.

ACKNOWLEDGEMENTS

This work is partly funded by the EU's Horizon Europe HarmonicAI project under the HORIZON-MSCA-2022-SE-01 scheme with grant agreement number 101131117. We thank the support provided by Samenwerkende Universitaire RekenFaciliteiten (SURF) with their HPC Cloud infrastructure. Many thanks to Dr. Jie Yang for his invaluable insights and constructive comments on this paper.

REFERENCES

[1] 2023. DeepSpeed. *Flops profiler* (2023). https://www.deepspeed.ai/tutorials/flops-profiler/#flops-measurement

[2] Durmus Alp Emre Acar, Yue Zhao, Ramon Matas Navarro, Matthew Mattina, Paul N Whatmough, and Venkatesh Saligrama. 2021. Federated learning based on dynamic regularization. *arXiv preprint arXiv:2111.04263* (2021).

[3] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, Brendan McMahan, et al. 2019. Towards federated learning at scale: System design. In *Proceedings of the machine learning and systems (MLSys).*

[4] Christopher Briggs, Zhong Fan, and Peter Andras. 2020. Federated learning with hierarchical clustering of local updates to improve training on non-IID data. In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN).*

[5] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. 2018. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097* (2018).

[6] Daoyuan Chen, Dawei Gao, Weirui Kuang, Yaliang Li, and Bolin Ding. 2022. pFL-bench: A comprehensive benchmark for personalized federated learning. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS).*

[7] Yujing Chen, Yue Ning, Martin Slawski, and Huzefa Rangwala. 2020. Asynchronous online federated learning for edge devices with non-iid data. In *Proceedings of the IEEE International Conference on Big Data (Big Data).*

[8] Hyunsung Cho, Akhil Mathur, and Fahim Kawsar. 2022. Flame: Federated learning across multi-device environments. In *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies.*

[9] Yae Jee Cho, Jianyu Wang, and Gauri Joshi. 2020. Client selection in federated learning: Convergence analysis and power-of-choice selection strategies. *arXiv preprint arXiv:2010.01243* (2020).

[10] Enmao Diao, Jie Ding, and Vahid Tarokh. 2020. Heterofl: Computation and communication efficient federated learning for heterogeneous clients. *arXiv preprint arXiv:2010.01264* (2020).

[11] Giancarlo Fortino and Paolo Trunfio. 2014. *Internet of things based on smart objects: Technology, middleware and applications.* Springer.

[12] Liang Gao, Huazhu Fu, Li Li, Yingwen Chen, Ming Xu, and Cheng-Zhong Xu. 2022. FedDC: Federated Learning With Non-IID Data via Local Drift Decoupling and Correction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).*

[13] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems* (2013).

[14] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. 2018. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604* (2018).

[15] Samuel Horvath, Stefanos Laskaridis, Mario Almeida, Ilias Leontiadis, Stylianos Venieris, and Nicholas Lane. 2021. Fjord: Fair and accurate federated learning under heterogeneous targets with ordered dropout. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS).*

[16] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. 2019. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335* (2019).

[17] Huawei Huang, Ruixin Li, Jialiang Liu, Sicong Zhou, Kangying Lin, and Zibin Zheng. 2022. Contextfl: Context-aware federated learning by estimating the training and reporting phases of mobile clients. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS).*

[18] Junxian Huang, Cheng Chen, Yutong Pei, Zhaoguang Wang, Zhiyun Qian, Feng Qian, Birjodh Tiwana, Qiang Xu, Z Mao, Ming Zhang, et al. 2011. Mobiperf: Mobile network measurement system. *Technical Report. University of Michigan and Microsoft Research* (2011).

[19] Andrey Ignatov, Radu Timofte, Andrei Kulik, Seungsoo Yang, Ke Wang, Felix Baum, Max Wu, Lirong Xu, and Luc Van Gool. 2019. Ai benchmark: All about deep learning on smartphones in 2019. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*

Workshop (ICCVW).

[20] Younes Jahandideh and A Mirzaei. 2021. Allocating duplicate copies for IoT data in cloud computing based on harmony search algorithm. *IETE Journal of Research* (2021).

[21] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. 2021. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning* (2021).

[22] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. 2020. Scaffold: Stochastic controlled averaging for federated learning. In *Proceedings of the International conference on machine learning (ICML).* PMLR.

[23] Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. In *Proceedings of the International conference on machine learning (ICML).* PMLR.

[24] Dieter Kraft. 1988. A software package for sequential quadratic programming. *Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt* (1988).

[25] Fan Lai, Yinwei Dai, Sanjay Singapuram, Jiachen Liu, Xiangfeng Zhu, Harsha Madhyastha, and Mosharaf Chowdhury. 2022. Fedscale: Benchmarking model and system performance of federated learning at scale. In *Proceedings of the International Conference on Machine Learning (ICML).* PMLR.

[26] Fan Lai, Xiangfeng Zhu, Harsha V Madhyastha, and Mosharaf Chowdhury. 2021. Oort: Efficient federated learning via guided participant selection. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI).*

[27] Ang Li, Jingwei Sun, Binghui Wang, Lin Duan, Sicheng Li, Yiran Chen, and Hai Li. 2020. Lotteryfl: Personalized and communication-efficient federated learning with lottery ticket hypothesis on non-iid datasets. *arXiv preprint arXiv:2008.03371* (2020).

[28] Ang Li, Jingwei Sun, Xiao Zeng, Mi Zhang, Hai Li, and Yiran Chen. 2021. Fedmask: Joint computation and communication-efficient personalized federated learning via heterogeneous masking. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys).*

[29] Chenning Li, Xiao Zeng, Mi Zhang, and Zhichao Cao. 2022. PyramidFL: A fine-grained client selection framework for efficient federated learning. In *Proceedings of the ACM Conference on Mobile Computing And Networking (MobiCom).*

[30] Qinbin Li, Bingsheng He, and Dawn Song. 2021. Model-contrastive federated learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR).*

[31] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine* (2020).

[32] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated optimization in heterogeneous networks. In *Proceedings of the Machine learning and systems (MLSys).*

[33] Xiaoxiao Li, Meirui Jiang, Xiaofei Zhang, Michael Kamp, and Qi Dou. 2021. Fedbn: Federated learning on non-iid features via local batch normalization. *arXiv preprint arXiv:2102.07623* (2021).

[34] Youpeng Li, Xuyu Wang, and Lingling An. 2023. Hierarchical Clustering-based Personalized Federated Learning for Robust and Fair Human Activity Recognition. (2023).

[35] Zonghang Li, Yihong He, Hongfang Yu, Jiawen Kang, Xiaoping Li, Zenglin Xu, and Dusit Niyato. 2022. Data heterogeneity-robust federated learning via group client selection in industrial iot. *IEEE Internet of Things Journal* (2022).

[36] Paul Pu Liang, Terrance Liu, Liu Ziyin, Nicholas B Allen, Randy P Auerbach, David Brent, Ruslan Salakhutdinov, and Louis-Philippe Morency. 2020. Think locally, act globally: Federated learning with local and global representations. *arXiv preprint arXiv:2001.01523* (2020).

[37] Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. 2020. Explainable ai: A review of machine learning interpretability methods. *Entropy* (2020).

[38] Lumin Liu, Jun Zhang, SH Song, and Khaled B Letaief. 2020. Client-edge-cloud hierarchical federated learning. In *Proceedings of the IEEE International Conference on Communications (ICC).*

[39] Lumin Liu, Jun Zhang, Shenghui Song, and Khaled B Letaief. 2022. Hierarchical federated learning with quantization: Convergence analysis and system design. *IEEE Transactions on Wireless Communications* (2022).

[40] Othmane Marfoq, Giovanni Neglia, Aurélien Bellet, Laetitia Kameni, and Richard Vidal. 2021. Federated multi-task learning under a mixture of distributions. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS).*

[41] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics.* PMLR.

[42] Takayuki Nishio and Ryo Yonetani. 2019. Client selection for federated learning with heterogeneous resources in mobile edge. In *Proceedings of the IEEE international conference on communications (ICC).*

[43] Xiaomin Ouyang and et.al. 2021. Clusterfl: a similarity-aware federated learning system for human activity recognition. In *Proceedings of International Conference on Mobile Systems, Applications, and Services (MobiSys).*

[44] Renjie Pi, Weizhong Zhang, Yueqi Xie, Jiahui Gao, Xiaoyu Wang, Sunghun Kim, and Qifeng Chen. 2023. Dynafed: Tackling client data heterogeneity with global dynamics. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).*

[45] Tianyu Qi, Yufeng Zhan, Peng Li, Jingcai Guo, and Yuanqing Xia. 2023. Hwamei: A Learning-Based Synchronization Scheme for Hierarchical Federated Learning. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)*.

[46] Zeyu Qin, Liuyi Yao, Daoyuan Chen, Yaliang Li, Bolin Ding, and Minhao Cheng. 2023. Revisiting Personalized Federated Learning: Robustness Against Backdoor Attacks. *arXiv preprint arXiv:2302.01677* (2023).

[47] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar, and H Brendan McMahan. 2020. Adaptive federated optimization. *arXiv preprint arXiv:2003.00295* (2020).

[48] Jaemin Shin, Yuanchun Li, Yunxin Liu, and Sung-Ju Lee. 2022. FedBalancer: Data and Pace Control for Efficient Federated Learning on Heterogeneous Clients. (2022).

[49] Jiangang Shu, Tingting Yang, Xinying Liao, Farong Chen, Yao Xiao, Kan Yang, and Xiaohua Jia. 2022. Clustered federated multitask learning on non-IID data with enhanced privacy. *IEEE Internet of Things Journal* (2022).

[50] Timo Sztyler and Heiner Stuckenschmidt. 2016. On-body localization of wearable devices: An investigation of position-aware activity recognition. In *IEEE International Conference on Pervasive Computing and Communications (PerCom)*.

[51] Linlin Tu, Xiaomin Ouyang, Jiayu Zhou, Yuze He, and Guoliang Xing. 2021. Feddl: Federated learning via dynamic layer sharing for human activity recognition. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*.

[52] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research* (2008).

[53] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S Rellermeyer. 2020. A survey on distributed machine learning. *Comput. Surveys* (2020).

[54] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. 2020. Federated learning with matched averaging. *arXiv preprint arXiv:2002.06440* (2020).

[55] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H Vincent Poor. 2020. Tackling the objective inconsistency problem in heterogeneous federated optimization. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*.

[56] Jiadai Wang, Lei Zhao, Jiajia Liu, and Nei Kato. 2019. Smart resource allocation for mobile edge computing: A deep reinforcement learning approach. *IEEE Transactions on emerging topics in computing* (2019).

[57] Lixu Wang, Shichao Xu, Xiao Wang, and Qi Zhu. 2021. Addressing class imbalance in federated learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*.

[58] Pete Warden. 2018. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209* (2018).

[59] Chenhao Xu, Youyang Qu, Yong Xiang, and Longxiang Gao. 2023. Asynchronous federated learning on heterogeneous devices: A survey. *Computer Science Review* (2023).

[60] Tianlong Yu, Tian Li, Yuqiong Sun, Susanta Nanda, Virginia Smith, Vyas Sekar, and Srinivasan Seshan. 2020. Learning context-aware policies from multiple smart homes via federated multi-task learning. In *Proceedings of the IEEE/ACM Conference on Internet-of-Things Design and Implementation (IoTDI)*.

[61] Han Zhang, Lavanya Ramapantulu, and Yong Meng Teo. 2019. Harmony: an approach for geo-distributed processing of big-data applications. In *Proceedings of the IEEE international conference on cluster computing (CLUSTER)*.

[62] Sai Qian Zhang, Jieyu Lin, and Qi Zhang. 2022. A Multi-agent Reinforcement Learning Approach for Efficient Client Selection in Federated Learning. *arXiv preprint arXiv:2201.02932* (2022).

[63] Tuo Zhang, Tiantian Feng, Samiul Alam, Sunwoo Lee, Mi Zhang, Shrikanth S Narayanan, and Salman Avestimehr. 2023. Fedaudio: A federated learning benchmark for audio tasks. In *Proceedings of the IEEE Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

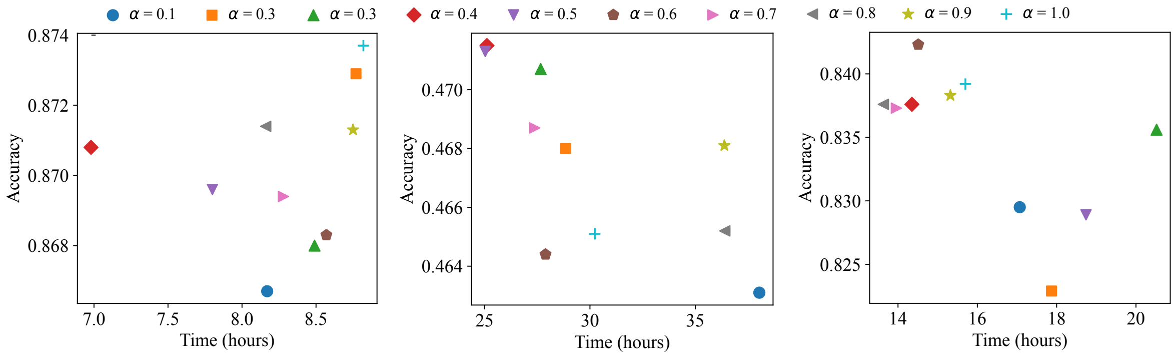
[64] Tuo Zhang, Lei Gao, Sunwoo Lee, Mi Zhang, and Salman Avestimehr. 2023. TimelyFL: Heterogeneity-aware Asynchronous Federated Learning with Adaptive Partial Training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[65] Qihua Zhou, Song Guo, Haodong Lu, Li Li, Minyi Guo, Yanfei Sun, and Kun Wang. 2020. Falcon: Addressing stragglers in heterogeneous parameter server via multiple parallelism. *IEEE Trans. Comput.* (2020).

A APPENDIX

A.1 Hyperparameter Tuning of FLAME

FLAME [8], as one of the baselines selected in this paper, is originally developed for IMU data. To report a more fair comparison, we conduct a grid search for the straggler penalty factor α (adopted in FLAME) to identify the optimal value when extending its application to other domains, including image classification, natural language processing, and speech recognition. Lower α will penalize slow clients more. The rationale behind fine-tuning this factor lies in its crucial role as a hyperparameter that balances data utility and system utility of devices, which directly influences the overall performance of the global model, on both final accuracy and time-to-accuracy. As shown in Figure 17, we report the final accuracy and time consumption when reaching target accuracy within 150 communication rounds, considering variations in the straggler penalty parameter α ranging from 0.1 to 1.0. Taking into account both the accuracy and efficiency of the model, we select α values of 0.4, 0.5, and 0.8 for these three application scenarios, respectively.

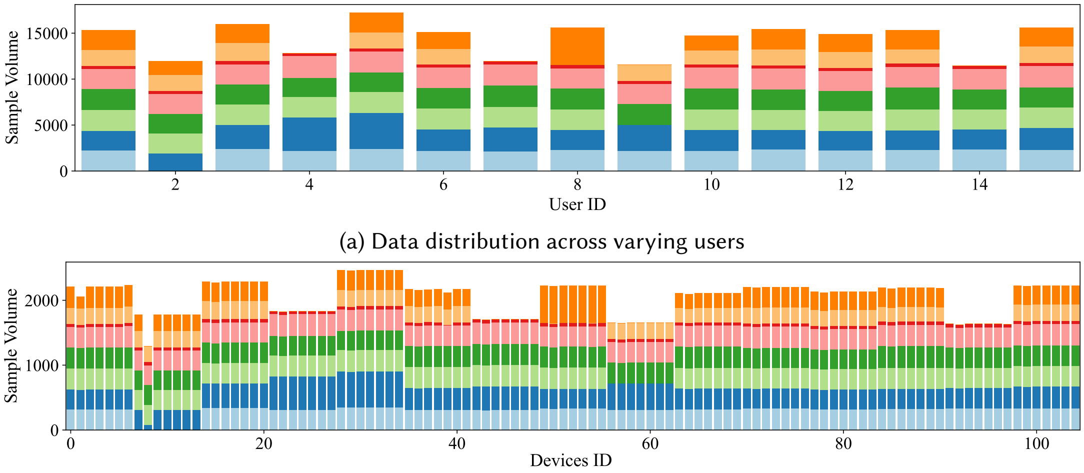


(a) #1 FEMNIST (b) #2 Shakespeare (c) #3 Google Speech

Fig. 17. The performance of FLAME across varying client straggler penalty factors on the FEMNIST, Shakespeare, and Google Speech datasets.

A.2 The Detailed Processing of RealWorld Dataset

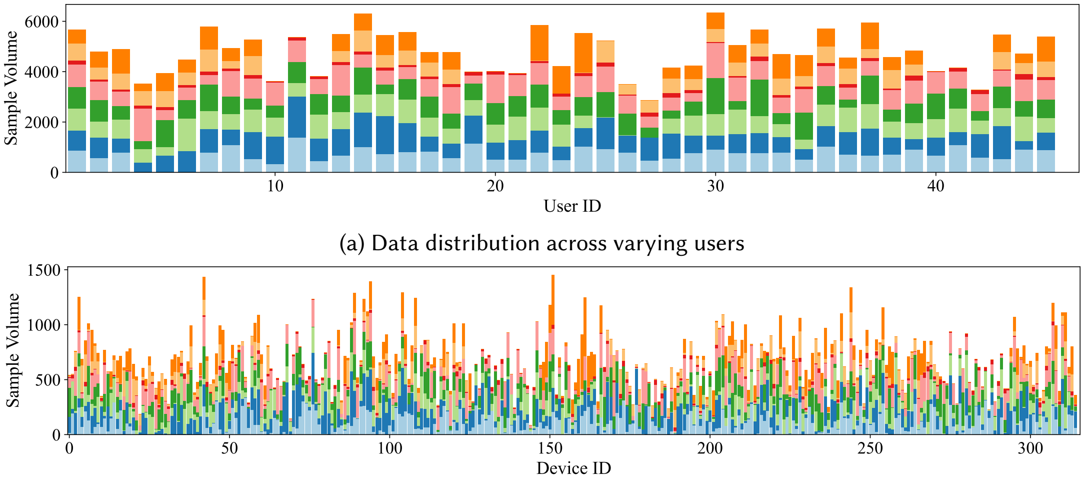
To assess the effectiveness of ShuffleFL in handling data inconsistency, specifically non-IID input spaces introduced by different devices, we employ a real-world multi-device dataset RealWorld [50] for human activity recognition. RealWorld includes motion data from 15 participants engaged in 8 activities: jumping, lying, standing, sitting, running, walking, climbing down, and climbing up. For each activity, it records simultaneously the accelerometer and gyroscope data from 7 body placements, including the chest, forearm, head, shin, thigh, upper arm, and waist, at a sampling rate of 50 Hz. Each participant performed each activity for roughly 10 minutes except for jumping due to the physical exertion (1.7 minutes). A sliding window of 2 seconds with a 2-second step is employed to generate a 600-dimensional feature for all 215,038 samples. Figure 18 illustrates the data distribution of each user (Figure 18a) and their devices (Figure 18b). It can be observed that the inherent experimental setup results in both user and device data distributions being IID, which does not accurately reflect real-world scenarios. To replicate a realistic data distribution, we expand one user into three users using a Dirichlet distribution with the parameter μ=0.5. To be more specific, we partition the data collected from each of the 7 different device placements within one user based on a Dirichlet distribution into 3 sets. These sets from 7 devices constitute 3 users with non-IID data distributions. After that, we can obtain 45 users. Besides, the adoption of a 2-second sliding window with a



(a) Data distribution across varying users

(b) Data distribution across varying devices

Fig. 18. The data (class) distribution of RealWorld dataset across varying users and their devices.



(a) Data distribution across varying users

(b) Data distribution across varying devices

Fig. 19. The data (class) distribution after partitioning the RealWorld dataset across varying users and their devices.

step size of 2 seconds prevents data overlap within the data samples across different users. Figure 19 shows the data distribution after performing such partitioning.

A.3 Optimization Time Consumed by Various Embedded Processors

In this section, we present the exact shrinkage ratios for each dataset used in our experiments (Table 5). Additionally, we offer detailed optimization time measurements obtained from real embedded devices, catering to various user types with varying device volumes (Tables 6–9).

Table 5. The shrinkage ratio γ for each processor profile across three complicated tasks involved in our experiments.

User Type		# 1 FEMNIST	# 2 Shakespeare	# 3 Google Speech
Raspberry Pi 4 (Model B)		0.25	0.20	0.35
Jetson Nano	Mode 1	0.10	0.10	0.20
	Mode 2	0.10	0.10	0.20
Jetson TX2	Mode 1	0.15	0.125	0.25
	Mode 2	0.15	0.125	0.25
Jetson AGX Xavier	Mode 1	0.25	0.20	0.35
	Mode 2	0.20	0.15	0.30

Table 6. Optimization time consumption (in seconds) of different processor profiles for users owning 3 devices across four datasets from different scenarios.

User Type		#1 FEMNIST		#2 Shakespeare		#3 Google Speech		#4 HARBox
		w/ reduction	w/o	w/ reduction	w/o	w/ reduction	w/o	
Raspberry Pi 4 (Model B)		7.56	25.24	4.06	65.68	5.40	20.25	4.35
Jetson Nano	Mode 1	6.05	239.02	4.77	240.41	4.85	69.92	5.72
	Mode 2	9.40	305.87	7.67	829.14	10.16	129.17	9.59
Jetson TX2	Mode 1	4.34	51.64	4.07	135.62	5.11	52.38	3.02
	Mode 2	8.34	245.77	9.08	625.10	7.14	73.32	5.33
Jetson Xavier NX	Mode 1	3.36	22.49	3.84	37.03	4.54	7.80	2.70
	Mode 2	5.06	35.89	4.57	107.66	7.49	29.20	3.50

Table 7. Optimization time consumption (in seconds) of different processor profiles for users owning 4 devices across four datasets from different scenarios.

User Type		#1 FEMNIST		#2 Shakespeare		#3 Google Speech		#4 HARBox
		w/ reduction	w/o	w/ reduction	w/o	w/ reduction	w/o	
Raspberry Pi 4 (Model B)		13.63	1353.06	12.85	500.82	15.10	60.08	8.24
Jetson Nano	Mode 1	18.93	2159.08	13.68	994.04	17.05	260.99	13.24
	Mode 2	29.33	3050.94	26.10	1573.03	24.49	453.31	19.41
Jetson TX2	Mode 1	13.18	1548.95	17.68	799.27	15.24	195.09	10.42
	Mode 2	23.12	2524.30	23.38	1132.73	20.57	486.84	16.05
Jetson Xavier NX	Mode 1	9.07	209.06	12.56	486.08	11.94	56.31	7.17
	Mode 2	10.85	1498.59	21.38	623.84	12.93	167.66	8.87

Table 8. Optimization time consumption (in seconds) of different processor profiles for users owning 5 devices across four datasets from different scenarios.

User Type		#1 FEMNIST		#2 Shakespeare		#3 Google Speech		#4 HARBox
		w/ reduction	w/o	w/ reduction	w/o	w/ reduction	w/o	
Raspberry Pi 4 (Model B)		29.98	3693.89	26.85	2066.64	26.93	246.37	17.09
Jetson Nano	Mode 1	33.79	4785.39	31.95	3068.35	30.18	1220.90	28.04
	Mode 2	56.35	9875.78	66.93	4964.06	44.47	2111.91	44.48
Jetson TX2	Mode 1	30.72	4103.72	30.50	3048.65	19.69	943.78	21.83
	Mode 2	44.83	6460.34	60.54	3522.68	35.95	1506.74	35.94
Jetson Xavier NX	Mode 1	17.86	2583.75	30.13	1612.25	13.28	160.70	15.83
	Mode 2	24.54	3988.52	34.90	2448.82	28.94	449.16	19.23

Table 9. Optimization time consumption (in seconds) of different processor profiles for users owning 6 devices across four datasets from different scenarios.

User Type		#1 FEMNIST		#2 Shakespeare		#3 Google Speech	
		w/ reduction	w/o	w/ reduction	w/o	w/ reduction	w/o
Raspberry Pi 4 (Model B)		63.60	11718.22	56.88	10059.33	48.62	1899.56
Jetson Nano	Mode 1	69.91	19256.10	73.01	13564.07	63.56	3683.57
	Mode 2	96.03	42039.11	95.48	38270.15	78.07	5446.48
Jetson TX2	Mode 1	62.34	18119.44	67.42	12728.67	54.97	3669.90
	Mode 2	99.71	28600.39	85.36	18645.53	75.94	4169.65
Jetson Xavier NX	Mode 1	48.78	11572.61	55.29	8726.23	43.28	759.64
	Mode 2	58.47	13212.79	64.31	10080.24	57.65	2248.08

Table 10. Optimization time consumption (in seconds) without performing complexity reduction for users from the RealWorld dataset. Each user has 7 devices.

User Type	Raspberry Pi 4B	Jetson Nano		Jetson TX2		Jetson Xavier NX	
		Mode 1	Mode 2	Mode 1	Mode 2	Mode 1	Mode 2
# Realworld	25.26	41.51	53.60	30.05	49.47	16.20	35.54