Beyond Fine-Tuning: Efficient and Effective Fed-Tuning for Mobile/Web Users

Bingyan Liu
bingyanliu@bupt.edu.cn
Beijing Key Laboratory of Intelligent Telecommunications Software and Multimedia, School of Computer Science, Beijing University of Posts and Telecommunications
Beijing, China

Yifeng Cai
caiyifeng@pku.edu.cn
MOE Key Lab of HCST, School of Computer Science, Peking University
Beijing, China

Hongzhe Bi
bhz@bupt.edu.cn
School of Computer Science, Beijing University of Posts and Telecommunications
Beijing, China

Ziqi Zhang
ziqi_zhang@pku.edu.cn
MOE Key Lab of HCST, School of Computer Science, Peking University
Beijing, China

Ding Li
ding_li@pku.edu.cn
MOE Key Lab of HCST, School of Computer Science, Peking University
Beijing, China

Yao Guo*
yaoguo@pku.edu.cn
MOE Key Lab of HCST, School of Computer Science, Peking University
Beijing, China

Xiangqun Chen
cherry@pku.edu.cn
MOE Key Lab of HCST, School of Computer Science, Peking University
Beijing, China

ABSTRACT

Fine-tuning is a typical mechanism to achieve model adaptation for mobile/web users, where a model trained by the cloud is further retrained to fit the target user task. While traditional fine-tuning has been proved effective, it only utilizes local data to achieve adaptation, failing to take advantage of the valuable knowledge from other mobile/web users. In this paper, we attempt to extend the *local-user fine-tuning* to *multi-user fed-tuning* with the help of Federated Learning (FL). Following the new paradigm, we propose **EEFT**, a framework aiming to achieve **E**fficient and **E**ffective **F**ed-**T**uning for mobile/web users. The key idea is to introduce lightweight but effective adaptation modules to the pre-trained model, such that we can freeze the pre-trained model and just focus on optimizing the modules to achieve cost reduction and selective task cooperation. Extensive experiments on our constructed benchmark demonstrate the effectiveness and efficiency of the proposed framework.

*Yao Guo is the corresponding author.

CCS CONCEPTS

• **Computer systems organization** → **Client-server architectures**; • **Human-centered computing** → **Ubiquitous and mobile computing**; • **Computing methodologies** → **Neural networks**.

KEYWORDS

Mobile/Web Computing, Model Adaptation, Fine-tuning, Federated Learning

ACM Reference Format:
Bingyan Liu, Yifeng Cai, Hongzhe Bi, Ziqi Zhang, Ding Li, Yao Guo, and Xiangqun Chen. 2023. Beyond Fine-Tuning: Efficient and Effective Fed-Tuning for Mobile/Web Users. In *Proceedings of the ACM Web Conference 2023 (WWW '23), April 30–May 04, 2023, Austin, TX, USA*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3543507.3583212

1 INTRODUCTION

With the popularity of deep learning, more and more researchers are devoted to empowering intelligent mobile and web applications such as sensor-based activity recognition [4, 17] and Web AR [33, 35]. Instead of training a deep learning model from scratch, in mobile and web scenarios, a noteworthy trend is to directly adapt a pre-trained model [42]. Fine-tuning is a typical technique to achieve model adaptation [23, 38] where a cloud server first pre-trains a deep learning model with sufficient data and each mobile/web user then uses their own data to retrain the cloud-trained model. The motivation behind fine-tuning is that the optimized parameters in the pre-trained model include more general knowledge and can adapt well to a related task at a faster speed [10], saving much computational cost for the resource-constrained mobile/web devices.

Despite being proved effective, the conventional fine-tuning technique may not be the optimal solution because it only utilizes the local data to adapt the pre-trained model. Our motivation is that in real-world web scenarios, there are a large number of users who may hold a similar, or even identical, task requirement to others [2]. Therefore, it is possible and meaningful to develop a method to achieve effective user cooperation to improve task performance. Motivated by this, we attempt to extend the *local-user fine-tuning* to *multi-user fed-tuning*, with the help of Federated Learning (FL) [27, 29, 41], which has been widely used to address the problem of data island as well as respecting user privacy.

Unfortunately, conventional FL is not suitable to our adaptation scenario. **First**, traditional FL requires training and exchanging the whole model at each federated round, which is unacceptable for mobile/web devices considering their limited computing power and network connection. Moreover, the pre-trained cloud model is usually much larger than the normal FL model, further exacerbating the training and communication cost. **Second**, different from traditional FL that is focused on a single task, in our scenario, there may include a variety of tasks in terms of user preference, which makes it undesirable to federate all models directly.

To the best of our knowledge, no existing FL technique can simultaneously address the above two problems to fit our scenario. To fill this gap, we propose **EEFT**, a framework aiming to achieve **E**fficient and **E**ffective **F**ed-**T**uning for mobile/web device users. The rationale behind EEFT is that: (1) for training and communication, it is wasteful to fine-tune and exchange the whole pre-trained model since typically the scale of the target task is small and does not need so many parameters to fit; (2) for task diversity, it is necessary to introduce extra information to the pre-trained model because the current pre-trained model does not have the capability to distinguish different task features. Inspired by this, EEFT designs a series of *lightweight adaptation modules* upon the original pre-trained model, such that we can freeze the pre-trained model and just focus on optimizing the modules to achieve cost reduction and selective task cooperation.

Designing lightweight adaptation modules and enabling them to achieve effective fed-tuning requires overcoming two technical challenges. We list them as follows:

- *How to properly select the components of the adaptation modules such that they can be perfectly incorporated to the pre-trained model for joint training?* A pre-trained model has its special architecture. Adding any new layers or connections may destroy the original model features. Therefore, it is important to carefully design and introduce the adaptation modules.
- *How to achieve selective federation for similar or identical user tasks?* Note that in a federated web system, the number of participating clients may be huge. Blindly conducting federation may lead to the cooperation among some irrelevant tasks and thus harm the final performance.

To address the first challenge, instead of inserting modules inside the pre-trained model, EEFT regards the model as several independent modules and augments each one in a skip-connection and parallel manner. As shown in Figure 1, we freeze the pre-trained modules and only fine-tune our new added adaptation modules: *utilization factors* and *task-specific blocks*. The *utilization factors* are responsible for measuring the utility degree of each module (e.g., a block or layer in a ResNet model [11]) in the pre-trained model, such that we can selectively utilize the valuable knowledge embedded in the pre-trained model for better target performance. The *task-specific blocks* aim to learn the unique features for the task of each mobile/web device, which complements the general knowledge of the pre-trained model to enable specific adaptation. In this way, the execution of the pre-trained model will not be interfered. It is worth noting that usually the required adaptation modules are much smaller than the pre-trained model since both the number of pre-trained modules and the task-specific features are limited. Therefore, by introducing these lightweight adaptation modules and only training and exchanging them, the whole process becomes efficient. Here training is optimized by a newly designed loss that penalizes not only the accuracy error but also the *model complexity* that can dynamically control the usage degree between large pre-trained modules and small adaptation modules, in order to favor the learning of the added parts for improved performance.

EEFT addresses the second challenge by precisely picking out the identical or similar tasks to a certain task according to the uploaded *utilization factors* involved in the adaptation modules. The insight is that similar tasks have a great possibility to hold similar utilization patterns to the same pre-trained model. Concretely, EEFT calculates the distance among these utilization factors as the user task similarity and then federates similar ones. Because uploaded modules share a same architecture, we can directly aggregate them with traditional algorithms (e.g., FedAvg [27]).

Considering that there is no available benchmark to conduct fed-tuning, we manually construct a benchmark based on public computer vision(CV) and human activity recognition (HAR) datasets, where we consider the differences in task types, environmental conditions, and user preferences, which are widely seen in the mobile/web scenarios (details in Appendix B.1). We evaluate our framework on the benchmark and compare it to the traditional local-user fine-tuning baseline and several federated learning methods. The results show that our approach significantly outperforms other methods on accuracy performance by up to 6.52% while using only 11.2% training parameters and saving up to 8.9× communication cost. In addition, we conduct a series of in-depth theoretical (details in Appendix A) and empirical analyses to prove the effectiveness of the proposed framework.

This paper makes the following contributions:

- We extend fine-tuning to a new paradigm, fed-tuning, for better model adaptation under the mobile/web scenario. To the best of our knowledge, this is the first attempt in the literature to study and explore multi-user fed-tuning.
- We design and implement *EEFT*, an efficient and effective framework to achieve fed-tuning. By adding, optimizing, and selectively federating two types of adaptation modules, EEFT can achieve higher model accuracy at a small training and communication cost.
- Extensive experiments on our constructed benchmark demonstrate the superiority of EEFT.

Relevance to Web. Recently, "Federated Learning for the Web and Mobile" has attracted more and more attention both in industry and academia. Google is trying to apply FL to its web application such as web tracking [1]. The web conference also considers FL as an important track topic [22, 30, 44]. This paper targets addressing the federated model adaptation issue for mobile/web users, which has high relevance to the submitted conference.

2 RELATED WORK

2.1 Fine-tuning

Fine-tuning [38], which have been widely used in the field of transfer learning [19, 20, 32, 47], can serve as an effective solution to implement model adaptation for mobile/web scenarios. The key idea is to inherit the pre-trained parameters and retrain them with the data in each device. For example, Xu *et al.* [42] proposed to use an on-device input method for next-word prediction, where a model was first trained on the cloud using massive public corpora, and then they incrementally customized the cloud-trained model with data on individual devices. Liu *et al.* [23] focused on conducting model personalization according to the environment of each mobile device. They also utilized the pre-trained model in the cloud as a base. Besides, some algorithms, such as fine-tuning filter [8] and ensemble learning for fine-tuning [49], were proposed to enhance the fine-tuning performance.

Unlike the aforementioned works restricted to local fine-tuning, EEFT conducts the adaptation process in a federated manner, which expands valuable knowledge from other devices and achieves improved performance.

2.2 Federated Learning

Federated learning (FL) has drawn increasing attention and has been applied to many areas due to its superiority in task performance and privacy guarantees [9, 21, 24, 29, 43, 46]. It is originally introduced by Mcmahan *et al.* [27], who aims to collaboratively train a shared global model under the decentralized data settings. Specifically, they developed the FedAvg algorithm to generate the global model by averaging the local parameters uploaded from each client. Recently, a series of research directions have been proposed to advance this promising field [15]. For example, a large number of research papers have been devoted to addressing the statistical heterogeneity (i.e., non-iid) problem that widely exists in FL systems [14, 25, 36]. Various security analyses on FL have been investigated to provide more insights to make FL robust [3, 28]. Besides, several communication-efficient FL techniques have also been applied to improve the overall speed of the FL system [13, 16].

Note that most of the research effort is based on an assumption: there is only *one task* in the FL system, which is inappropriate to our adaptation scenario where multiple tasks might be respectively distributed to different mobile/web devices. One closely related work called *FedMTL* [26] has attempted to apply multi-task learning to FL to cope with the multiple tasks, where a weight factor is applied to different tasks in order to achieve better federation. The proposed EEFT differs in two aspects: (1) we do not need to train and upload the whole model parameters, saving massive computation and communication costs; (2) we directly pick out suitable tasks to federate rather than allocating a weight to each task, which contributes to eliminating the bad influence coming from irrelevant tasks.

3 PROBLEM FORMULATION

3.1 Workflow of FL

Generally, there are two parties involved in the FL process: clients and a central server. Each client trains a shared global model locally with a few epochs and only uploads the parameters/gradients of the model to a central server, where we implement different algorithms to aggregate the uploaded information and then distribute the aggregated model to each client. We may iterate this pipeline many rounds until model convergence. During the process, no raw data are transmitted, thus ensuring user privacy to some extent.

3.2 Fed-tuning

Similar to FL, in fed-tuning, we also need many mobile/web devices as "clients", a shared pre-trained cloud model as the initial model, and a central server to coordinate the uploaded information from each device. The differences lie in that: (1) the tasks among devices may not be identical and sometimes are significantly different; (2) the initial model has been pre-trained in the cloud and has a larger size compared to the model in traditional FL. Therefore, the typical FL pipeline cannot be directly applied to conduct fed-tuning, and we need a new solution to achieve our goal.

Formally, assuming there are N mobile/web devices with tasks $\{T_1, T_2, ..., T_N\}$ and corresponding data $\{D_1, D_2, ..., D_N\}$, we define the goal of fed-tuning as follows.

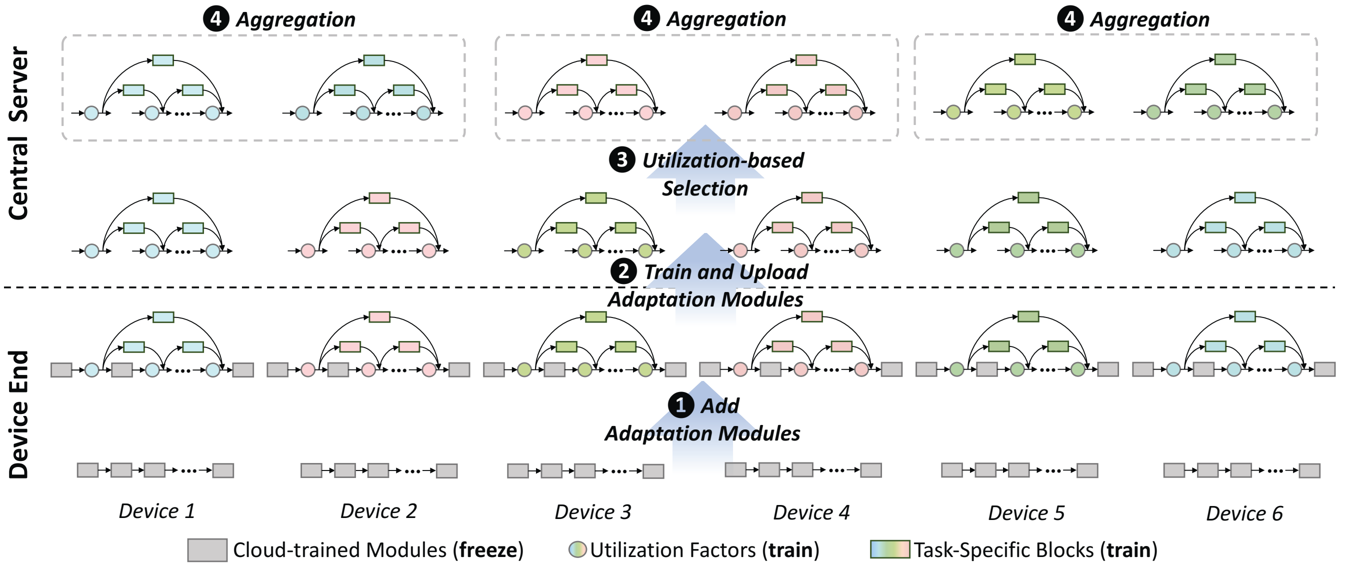
Definition 1. ***(Fed-tuning)****. Suppose a desirable model architecture M_{des} designed for efficient training and communication can be generated based on the shared cloud-trained model M_{global}. The goal of fed-tuning is to: (1) optimize M_{des} with the data $\{D_1, D_2, ..., D_N\}$ locally; (2) upload and selectively federate the local model parameters $\{P_{l_1}, P_{l_2}, ..., P_{l_N}\}$ in order to obtain the final device-specific models $\{M_{T_1}, M_{T_2}, ..., M_{T_N}\}$ for each device after R rounds.*

To accomplish this objective, we first add a series of lightweight adaptation modules to the pre-trained model to obtain the desirable model architecture M_{des}, which is then optimized by a designed loss $\mathcal{L}$ that penalizes both the accuracy error and the model complexity. By comparing the *utilization factors* α included in the added modules, we are able to select n ($n < N$) devices $\{j_1, j_2, ..., j_n\}$ with similar task to a certain device i and aggregate their corresponding uploaded parameters $\{P_{l_i}, P_{l_{j_1}}, ..., P_{l_{j_n}}\}$ (i.e., added adaptation modules) using existing aggregation algorithms, generating the final device-specific models. Note that our framework is based on the following assumption: Randomly given a device in the federated system, there exist some devices with similar/identical tasks, which is commonly seen in the context of large-scale mobile/web federation scenarios.

4 FRAMEWORK DESIGN

4.1 System Overview

We design and develop EEFT, a framework to conduct model adaptation in a federated manner via introducing, training, and federating two types of adaptation modules, such that the data knowledge of a mobile/web device can be effectively expanded at a small cost. Figure 1 depicts the overall pipeline of our framework, which can be briefly summarized as follows.



Central Server
Device End
4 Aggregation
4 Aggregation
4 Aggregation
3 Utilization-based Selection
2 Train and Upload Adaptation Modules
1 Add Adaptation Modules
Device 1
Device 2
Device 3
Device 4
Device 5
Device 6
Cloud-trained Modules (freeze)
Utilization Factors (train)
Task-Specific Blocks (train)

Figure 1: Pipeline of the proposed EEFT framework. It mainly includes four steps: adding adaptation modules to the pre-trained model for each device; training and uploading these adaptation modules with our designed loss while freezing pre-trained modules; measuring the similarity among these uploaded parameters with the utilization factors in order to select suitable federation targets; aggregating the parameters accordingly. ***Here the color represents the task features. Similar tasks are reflected by similar colors.***

(1) *Adding adaptation modules to the pre-trained model,* where the utilization factors are multiplied to each module of the original model and the task-specific blocks are introduced in a form of skip connections as used in DenseNets [12].
(2) *Training and uploading the adaptation modules,* where a new loss $\mathcal{L}$ that encourages the learning of the added modules is applied to optimize the parameters.
(3) *Selecting the federating targets based on a utilization-based comparison,* where the uploaded utilization factors of each device model are used as indicators to compare with each other, generating a task similarity matrix to guide the federating policy.
(4) *Aggregating the selected modules*, where we average the parameters in each location of these selected modules to achieve federation.

The last three steps may be executed multiple times until the training converges. The final aggregated modules are distributed to corresponding device users for deployment.

4.2 Introducing Adaptation Modules

This step adds the adaptation modules to optimize the target task. Specifically, we assume the pre-trained model M_{global} has L blocks $\{B_1, B_2, ..., B_L\}$ and the inference function can be denoted as

$$f(x) = B_L \odot B_{L-1} \odot \cdots \odot B_1 \odot (x) \tag{1}$$

where x is the data sample and $\odot$ represents the operations of the blocks, such as convolutions, poolings, etc. As shown in Figure 2, for each block B_z, we introduce a utilization factor α_z and several task-specific skip blocks $\{S_z^i\}_{i=z-k}^{z-1}$ that come from previous k blocks.

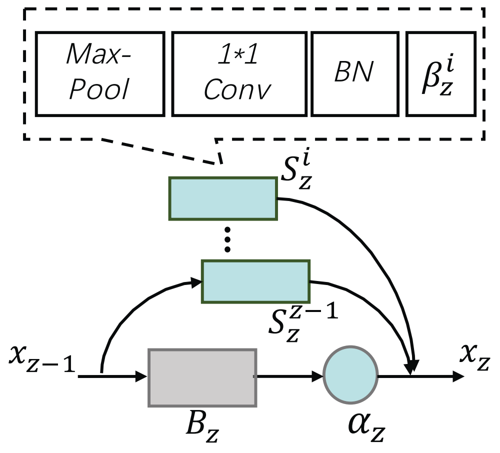


Figure 2: Illustration of the added adaptation modules. For each pre-trained block B, we attach a utilization factor α and several task-specific skip blocks S. Each skip block is composed of a max-pooling layer, a convolutional layer, a Batch Normalization (BN) layer, and a soft-attention parameter that is used to measure the importance of the block.

After this augmentation, the final output of x_z is the combination of the B_z output, α_z output, and $\{S_z^i\}_{i=z-k}^{z-1}$ output, which can be calculated by

$$x_z = \alpha_z * B_z \odot (x_{z-1}) + \sum_{i=z-k}^{z-1} S_z^i \odot (x_i) \tag{2}$$

where $*$ is the multiplication operation. α_z is a scalar that satisfies $\alpha_z \in [0, 1]$. Each skip block S_z^i contains the following parts: (1) a spatial max-pooling layer that converts activations from the spatial resolution of x_i to x_z; (2) a 1×1 convolutional layer (attached by a Batch Normalization (BN) layer) that projects the input feature map x_i to the desired number of channels for x_z; (3) a soft-attention parameter β_z^i to represent the importance of this block, which will be normalized by computing the *softmax* across all skip blocks at the z_{th} block.

In this way, we are able to control the importance of different blocks in the pre-trained model by α and complement the extra task features by S, providing a feasible and lightweight architecture for later training and uploading. We will analyze the concrete tuning and communication cost in Section 5.3.

4.3 Training and Uploading

Given an expanded architecture generated by the last step, a key question is how to effectively train it. A natural idea is to optimize the added parameters directly by minimizing accuracy error loss as traditional fine-tuning does. However, this simple optimization is not optimal since the adaptation modules may not match the pre-trained ones. On the one hand, the weights of the adaptation modules are randomly initialized while the weights of the original model are pre-trained on a large-scale dataset. On the other hand, the overall number of the added parameters is extremely small compared to the pre-trained model. Thus, it is difficult to focus on training the adaptation modules to fit the new task.

In our work, we design a new loss, where not only the accuracy error but also the *model complexity* are penalized for encouraging and contributing to the learning of the added modules. Here the *model complexity* can be defined as

$$\begin{aligned} model\_complexity = \sum_{i=1}^{L} \alpha_i * params(B_i) - \\ \sum_{i=1}^{L}\sum_{j=1}^{i} \beta_i^j * params(S_i^j) \end{aligned} \tag{3}$$

where $params()$ represents the number of parameters of the input modules. From the equation, we can observe that: (1) *model complexity* describes the usage degree of pre-trained modules and added modules by α and β; (2) minimizing the *model complexity* can weaken the pre-trained modules while strengthening the added modules (i.e., decrease α and increase β), which contributes to the learning of the added modules. Therefore, we design the new loss as follows:

$$\mathcal{L} = \mathcal{L}_{ce} + \gamma * model\_complexity, \tag{4}$$

where $\mathcal{L}_{ce}$ is the cross-entropy loss used to penalize accuracy error, and γ is a hyper-parameter that controls the proportion of the *model complexity.* We use this loss to optimize the adaptation modules in each device and upload them to a central server.

4.4 Federating Strategy and Aggregation

In conventional federated learning, all of the uploaded parameters come from the same task and thus can be aggregated directly without any additional processing. However, it is infeasible for our scenario where multiple tasks might exist in the federated system. Confronted with diverse tasks, we design two key steps to achieve federation: (1) comparing the similarity among these device tasks based on the corresponding uploaded adaptation modules; (2) leveraging the similarity to conduct a selective aggregation.

We first use the *utilization factors* in adaptation modules to compare the task similarity. Intuitively, the *utilization factors* can characterize the task property because they are based on an identical and freezing pre-trained model. In other words, if all of the pre-trained blocks have similar utilization values for two tasks, we believe the two tasks share similar feature space and should be aggregated. Specifically, we use Euclidean Distance [5] as the metric to compare the similarity between these *utilization factors*

$$sim(\alpha_i, \alpha_j) = \sqrt{\sum_{c=1}^{L} (\alpha_i^c - \alpha_j^c)^2} \tag{5}$$

where α_i^c and α_j^c denote the c_{th} element of the factor α_i and α_j and in total, we have L elements (same as the number of pre-trained blocks). In terms of this equation, we are able to generate a task similarity matrix, which determines which tasks should be selected and federated by setting a threshold (the threshold is apparent as shown in Figure 4 (a)). According to the similarity matrix, we then aggregate the adaptation modules of similar tasks by averaging them and distribute them to each device, finishing a single round of fed-tuning. In this way, we can finally obtain the device-specific models $\{M_{T_1}, M_{T_2}, ..., M_{T_N}\}$ for each device after R rounds.

5 EVALUATION

5.1 Experimental Setup

Benchmark. Since there is no available benchmark to evaluate the performance of our framework, we manually construct one with three typical mobile/web scenarios: *task-specific* scenario, *environment-specific* scenario and *user-specific* scenario. The detailed statistics are illustrated in Table 1 and the concrete description of these scenarios can be found in Appendix B.1.

Baselines. EEFT is compared to the fine-tuning baseline and several federated learning methods, which are briefly summarized as follows:

(1) *Fine-tuning [38]:* This baseline directly retrains the pre-trained model with data of each user.
(2) *FedAvg [27]:* This method averages the local parameters uploaded from each user to generate a global model. Towards the scenario where user tasks own a different number of categories (e.g., CUB-200 has 200 classes while Caltech-256 has 256 classes), we only average the shared pre-trained layers and leave the last classification layer to be optimized in each user device.
(3) *Selective Masking (SM) [13]:* A communication-efficient FL techniques achieved by selectively uploading partial weights or gradients.

Table 1: Statistics of our simulated benchmarks. Here $Envir-1$ represents the simulation of Office-Home and $Envir-2$ represents the simulation of HAR-Depth.

Scenario		Device index	#training sample	#testing sample
Task	*CUB-200*	1,2,...,5	5990	5790
	MIT-67	6,7,...,10	5355	1339
	Caltech-256	11,12,...,15	5120	5120
Envir-1	*Art*	1,2,...,5	1940	485
	Clipart	6,7,...,10	3490	873
	Product	11,12,...,15	3550	887
Envir-2	*Outdoor*	1,2,3	1339	899
	Indoor_nor	4,5,6	1293	895
	Indoor_dark	7,8,9	2023	902
User	*Type1*	1,2,...,20	2000	2000
	Type2	21,22,...,40	2000	2000
	Type3	41,42,...,60	2000	2000
	Type4	61,62,...,80	2000	2000
	Type5	81,82,...,100	2000	2000

(4) *FedMTL [26]:* This method borrows the idea of multi-task learning to facilitate task performance under the federated settings, where we apply a weight factor to each user task in the system for better assisting the target task.

Implementation details All our experiments are simulated and conducted in a server that has 4 GeForce GTX 2080Ti GPUs, 48 Intel Xeon CPUs, and 128GB memory. We implement EEFT in Python with PyTorch, and all the experiments are conducted on the ResNet-18 architecture [11], which is pre-trained with the ImageNet dataset [6].

We now describe the standard implementation of EEFT, which is used throughout our experiments unless otherwise specified. The concrete parameter settings are as follows: For adaptation modules, we augment three skip connections (i.e., $k = 3$) to each block of the pre-trained model. The initial values of the utilization factor α and soft-attention parameters β are set to -3 and 2 for better training. The hyper-parameter γ is set to 0.3 to provide a good trade-off between accuracy and model complexity. We use SGD as the optimizer for training, and the learning rate is set to 0.01 with a momentum of 0.5. All of the experiments are conducted for 50 federating rounds to guarantee convergence. Finally, we run each experiment 3 times and average them as the reported results. We will release our code at: *https://github.com/lebyni/fed-tuning*.

5.2 Performance Comparison

Here we use the pre-trained ResNet-18 as the backbone and replace the last classification layer (i.e., fully connected layer) according to the concrete task. Next, we illustrate the detailed results of each benchmark.

Accuracy comparison on the simulated benchmarks. From Table 2-5, we can clearly see that the proposed framework outperforms other methods with an average of 1.81%-2.77% accuracy

Table 2: Results on the task-specific scenario, where we test the accuracy performance (%) with different methods. Here devices with the same task are considered together and we average their accuracy.

Method	Task-specific			
	CUB-200	*MIT-67*	*Caltech-256*	*Average*
Fine-tune	44.63	57.34	58.07	53.35
SM	44.74	55.01	57.55	52.43
FedAvg	45.67	56.07	58.52	53.42
FedMTL	47.69	**59.35**	54.36	53.80
Ours	**48.99**	59.09	**58.75**	**55.61**

Table 3: Results on the environment-specific scenario. Here is the accuracy performance (%) on CV applications (Office-Home).

Method	Environment-specific(Office)			
	Artistic	*Clipart*	*Product*	*Average*
Fine-tune	48.70	53.06	78.17	59.98
SM	67.22	71.48	86.70	75.13
FedAvg	67.42	71.94	86.13	75.16
FedMTL	67.00	71.21	86.44	74.88
Ours	**69.48**	**73.31**	**89.85**	**77.55**

Table 4: Results on the environment-specific scenario.Here is the accuracy performance (%) on HAR applications (HAR-Depth).

Method	Environment-specific(HAR)			
	Outdoor	*Indoor_normal*	*Indoor_dark*	*Avg*
Fine-tune	81.35	90.13	89.17	86.88
SM	91.69	99.11	99.26	96.68
FedAvg	98.22	100.00	87.22	98.48
FedMTL	99.18	98.80	98.76	98.69
Ours	**99.67**	**100.00**	**100.00**	**99.89**

Table 5: Accuracy performance (%) on the user-specific scenario.

Method	User-specific					
	Type1	*Type2*	*Type3*	*Type4*	*Type5*	*Avg*
Fine-tune	91.58	77.18	84.60	92.64	89.42	87.08
SM	44.15	41.55	35.55	48.80	42.67	42.54
FedAvg	48.50	32.20	40.55	56.05	45.33	44.53
FedMTL	41.99	20.81	32.23	55.47	44.72	39.04
Ours	**93.35**	**83.70**	**87.75**	**94.55**	**89.90**	**89.85**

Table 6: Effect of the proposed components on our task-specific scenario.

Type	Dataset	#added modules in each block				loss effect		federating effect	
		1-skip	*3-skip*	*5-skip*	*7-skip*	$\mathcal{L}_{ce}$	$\mathcal{L}$	*w/o aggregation*	*aggregation*
Task	*CUB-200*	47.55	**48.99**	48.85	48.70	48.04	**48.99**	33.65	**48.99**
	MIT Indoor-67	57.91	59.09	**60.01**	59.95	58.25	**59.09**	51.86	**59.09**
	Caltech-256	57.89	58.75	**59.34**	59.17	58.45	**58.75**	54.77	**58.75**

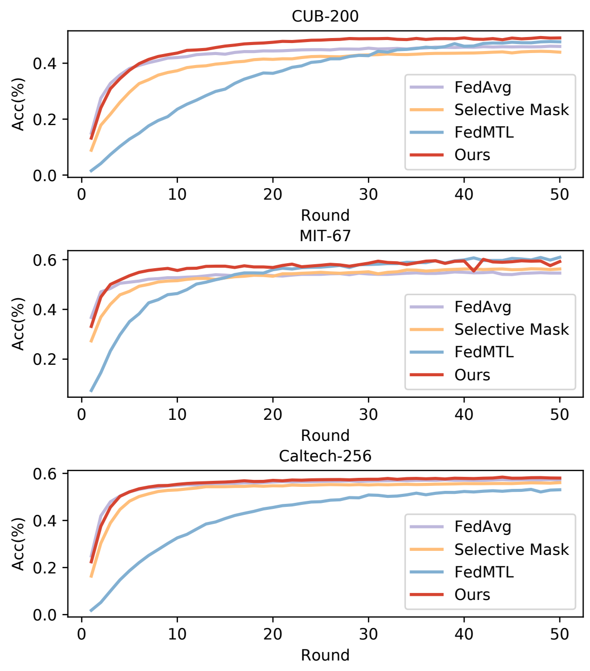


Figure 3: Convergence performance on the task-specific scenario. For each task type, we select a user as an example and plot the corresponding convergence line.

Table 7: The number of parameters needed for training and communication on different methods.

Method	#Train Params (M)	#Communication (M)
Fine-tune	11.18	0
SM	1.34	2*50*1.34
FedAvg	11.18	2*50*11.18
FedMTL	11.18	2*50*11.18
Ours	1.25	2*50*1.25

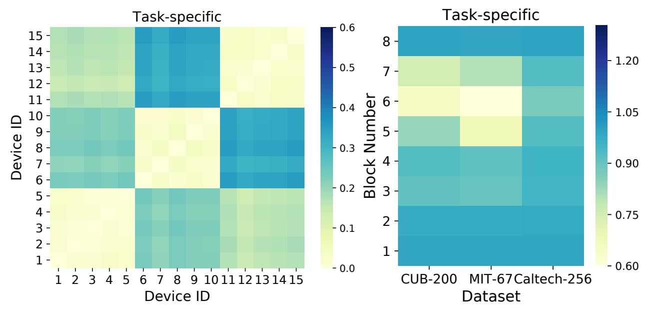
improvement. This demonstrates that by adding and selectively federating the adaptation modules, we can greatly boost the accuracy performance with less training or communication cost (details in Section 5.3). Besides, it is worth noting that: for the task-specific and environment-specific scenarios, a higher accuracy can be achieved using the conventional FL methods compared to the local fine-tuning. In other words, we can directly benefit from federated learning when the device difference only lies in task types or environments. However, most conventional FL methods cannot obtain a good performance under the user-specific condition, even largely worse than the local fine-tuning. This demonstrates that simply aggregating all of the uploading parameters is inappropriate when the heterogeneity of data distribution is more apparent. Unlike these FL methods, our framework selectively conducts the aggregation process and achieves better performance no matter what scenario we are in.

Convergence comparison on the simulated benchmarks. We record the test accuracy of each round in FL and plot the convergence lines of different methods. Here we use the *task-specific* scenario as an example, and for each task type, we select a user to observe the convergence performance. Note that fine-tuning does not need the federation process and hence is not included in the comparison. As shown in Figure 3, our EEFT converges faster than other methods and can reach the highest final test accuracy on average. Another interesting finding is that although *FedMTL* performs badly at the beginning, the final test accuracy is comparable to ours, especially for the first two task types. This indicates that *FedMTL* is able to gradually learn the correct relation among these tasks as the round increases.

5.3 Efficiency of EEFT

Unlike traditional FL, in our settings, the target model is a pre-trained model, resulting in massive training and communication cost if we directly follow the existing FL pipeline. In this subsection, we study the efficiency of EEFT, where we record the training parameters and communication cost of the task-specific scenario.

Table 7 demonstrates the results. Note that the communication cost is calculated by accumulating the number of uploading and offloading parameters. Obviously, we need fewer training parameters than other methods that tune the whole pre-trained model. Specifically, we can only train 11.2% parameters of the pre-trained ResNet-18 while obtaining improved accuracy, which significantly relieves the pressure of training on the resource-constrained mobile/web devices. We assume that the FL process will run for 50 rounds towards the communication comparison, which means the communication cost is $2 \times 50 \times uploaded\_parameters$. Note that we do not need to upload any parameters in fine-tuning; thus, the cost is zero. For other traditional FL methods, as shown in the table, our approach can save up to 8.9× communication cost, which confirms the communication efficiency of EEFT. Although the *SM* method can achieve comparable training and communication cost to ours, its efficiency is at the expense of model accuracy, which will significantly affect user experience.



(a) Device similarity (b) Utilization degree

Figure 4: Visualization of the user similarity matrix and the learned utilization factor value at each pre-trained block.

5.4 Effect of the Introduced Components

Our framework mainly introduces three key components: two types of adaptation modules, a new loss, and a federating strategy. Here we conduct ablation studies to explore whether each of them is beneficial to the final performance. Specifically, for adaptation modules, we investigate how different numbers (1,3,5,7) of added skip connections in each block affect the results. For the loss, we respectively employ our designed loss and the traditional cross-entropy loss to conduct our pipeline. Finally, we compare the achieved accuracy using the conventional aggregation and our federating strategy to test the usefulness of the proposed strategy.

Table 6 summarizes the results, where a pre-trained ResNet-18 is used as the backbone, and we average the user results of the same task type. For each study, we fix the other two and focus on the specific component. From the figure, we can observe that our designed loss and aggregation strategy perform better than the corresponding baselines across all of the task types, which validates their effectiveness. Towards the added modules, we find that different task types have different best choices. For example, adding *5-skip* is desirable for the MIT Indoor-67 while in CUB-200, adding *3-skip* achieves the highest accuracy. Besides, an interesting observation is that when we increase the number of skip connections to 7 for each block, the performance is surprisingly degraded. We believe this is because excessively introduced modules increase the training parameters, which will cause the overfitting problem and thus harm the accuracy. In conclusion, *3-skip* or *5-skip* is enough to obtain good results.

5.5 Implication of the Learned utilization Factors

As aforementioned, the goal of introducing utilization factors lies in two aspects: (1) using them as an indicator to compare the similarity between two tasks for selective aggregation; (2) measuring the importance of different modules of the pre-trained model in order to utilize them better. Here we extract the learned utilization factors from each device model of the *task-specific* scenario to explore how they achieve the two goals.

Specifically, we first extract the utilization factors from the learned models and use them to calculate the task similarity according to (5). Figure 4(a) demonstrates the similarity matrix. We can observe that user devices coming from the same task type can be easily picked out by setting an apparent threshold, proving the effectiveness of the utilization factors in selecting suitable device tasks. Besides, we further visualize their averaged value of the utilization factors of each task type. As shown in Figure 4(b), each pre-trained block has its unique utilization to different task types, which confirms that the introduced factors can learn the different utilization patterns in terms of different inputs, providing guidance to the better usage of the pre-trained model. In addition, we find that Caltech-256 has a higher overall utilization degree compared to others. This is reasonable because this dataset is more similar to the pre-trained ImageNet dataset, which indicates that we can directly transfer the pre-trained blocks without the need to tune them much. In summary, these figures imply the necessity and usefulness of introducing the utilization factors.

6 DISCUSSION

This section discusses the limitations of the paper. First, EEFT requires adding specific modules to the pre-trained model, which will increase the model size and then introduces extra inference time. However, considering that the added parameters are small compared to the large pre-trained model, we believe this consumption is acceptable. Another limitation is that our implementation is based on CNN models. In the future, we will further explore how to apply our framework to other models such as RNNs or GNNs. Finally, one possible threat to validity is that whether the studied benchmark can represent the mobile/web scenarios. We try our best to minimize this threat by covering different possible data settings for simulating real users. We will continue to explore other settings in the future.

7 CONCLUSION

In this paper, we propose *EEFT*, where diverse users cooperate with each other to learn their specific tasks. We design and implement EEFT by adding and federating light-weight adaptation modules. As these modules are much smaller than the pretrained model, they can be used to efficiently and effectively expand the valuable data knowledge and benefit each user. Experiments on our simulated benchmark demonstrate the effectiveness of the proposed framework, which outperforms other state-of-the-art methods while saving much training or communication cost. In the future, we will further explore fed-tuning from other perspectives, such as privacy and robustness.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable feedback. This work was partly supported by the Director Fund of Beijing Key Laboratory and the National Natural Science Foundation of China (62141208).

REFERENCES

[1] 2021. Federated Learning of Cohorts. https://en.wikipedia.org/wiki/Federated_Learning_of_Cohorts.
[2] 2022. Web of Things (WoT): Use Cases and Requirements. https://www.w3.org/TR/wot-usecases/.
[3] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. 2019. Analyzing federated learning through an adversarial lens. In *International Conference on Machine Learning*. PMLR, 634–643.

[4] Ling Chen, Yi Zhang, and Liangying Peng. 2020. METIER: a deep multi-task learning based activity and user recognition model using wearable sensors. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 1 (2020), 1–18.

[5] Per-Erik Danielsson. 1980. Euclidean distance mapping. *Computer Graphics and image processing* 14, 3 (1980), 227–248.

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[7] Gregory Griffin, Alex Holub, and Pietro Perona. 2007. Caltech-256 object category dataset. (2007).

[8] Yunhui Guo, Yandong Li, Liqiang Wang, and Tajana Rosing. 2020. AdaFilter: Adaptive Filter Fine-tuning for Deep Transfer Learning. *Thirty-Fourth AAAI Conference on Artificial Intelligence* (2020).

[9] 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).

[10] Kaiming He, Ross Girshick, and Piotr Dollár. 2019. Rethinking imagenet pre-training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 4918–4927.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[12] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4700–4708.

[13] Shaoxiong Ji, Wenqi Jiang, Anwar Walid, and Xue Li. 2020. Dynamic sampling and selective masking for communication-efficient federated learning. *arXiv preprint arXiv:2003.09603* (2020).

[14] Yihan Jiang, Jakub Konečnỳ, Keith Rush, and Sreeram Kannan. 2019. Improving federated learning personalization via model agnostic meta learning. *arXiv preprint arXiv:1909.12488* (2019).

[15] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. 2019. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977* (2019).

[16] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. *NIPS workshop* (2016).

[17] Hyeokhyen Kwon, Catherine Tong, Harish Haresamudram, Yan Gao, Gregory D Abowd, Nicholas D Lane, and Thomas Ploetz. 2020. IMUTube: Automatic extraction of virtual on-body accelerometry from video for human activity recognition. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 3 (2020), 1–29.

[18] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. 2019. On the convergence of fedavg on non-iid data. *arXiv preprint arXiv:1907.02189* (2019).

[19] Yuanchun Li, Ziqi Zhang, Bingyan Liu, Ziyue Yang, and Yunxin Liu. 2021. ModelDiff: testing-based DNN similarity comparison for model reuse detection. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 139–151.

[20] Bingyan Liu, Yifeng Cai, Yao Guo, and Xiangqun Chen. 2021. TransTailor: Pruning the pre-trained model for improved transfer learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 8627–8634.

[21] Bingyan Liu, Yifeng Cai, Ziqi Zhang, Yuanchun Li, Leye Wang, Ding Li, Yao Guo, and Xiangqun Chen. 2021. DistFL: Distribution-aware Federated Learning for Mobile Scenarios. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 5, 4 (2021), 1–26.

[22] Bingyan Liu, Yao Guo, and Xiangqun Chen. 2021. PFA: Privacy-preserving Federated Adaptation for Effective Model Personalization. In *Proceedings of the Web Conference 2021*. 923–934.

[23] Bingyan Liu, Yuanchun Li, Yunxin Liu, Yao Guo, and Xiangqun Chen. 2020. Pmc: A privacy-preserving deep learning model customization framework for edge computing. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 4 (2020), 1–25.

[24] Bingyan Liu, Nuoyan Lv, Yuanchun Guo, and Yawen Li. 2023. Recent Advances on Federated Learning: A Systematic Survey. *arXiv preprint arXiv:2301.01299* (2023).

[25] Yishay Mansour, Mehryar Mohri, Jae Ro, and Ananda Theertha Suresh. 2020. Three approaches for personalization with applications to federated learning. *arXiv preprint arXiv:2002.10619* (2020).

[26] Othmane Marfoq, Giovanni Neglia, Aurélien Bellet, Laetitia Kameni, and Richard Vidal. 2021. Federated multi-task learning under a mixture of distributions. *Advances in Neural Information Processing Systems* 34 (2021), 15434–15447.

[27] 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, 1273–1282.

[28] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. 2019. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 691–706.

[29] Chaoyue Niu, Fan Wu, Shaojie Tang, Lifeng Hua, Rongfei Jia, Chengfei Lv, Zhihua Wu, and Guihai Chen. 2020. Billion-scale federated learning on mobile clients: a submodel design with tunable privacy. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. 1–14.

[30] Seungeun Oh, Jihong Park, Praneeth Vepakomma, Sihun Baek, Ramesh Raskar, Mehdi Bennis, and Seong-Lyun Kim. 2022. LocFedMix-SL: Localize, Federate, and Mix for Improved Scalability, Convergence, and Latency in Split Learning. In *Proceedings of the ACM Web Conference 2022*. 3347–3357.

[31] Xiaomin Ouyang, Zhiyuan Xie, Jiayu Zhou, Jianwei Huang, and Guoliang Xing. 2021. ClusterFL: a similarity-aware federated learning system for human activity recognition. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*. 54–66.

[32] Sinno Jialin Pan and Qiang Yang. 2009. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22, 10 (2009), 1345–1359.

[33] Xiuquan Qiao, Pei Ren, Schahram Dustdar, and Junliang Chen. 2018. A new era for web AR with mobile edge computing. *IEEE Internet Computing* 22, 4 (2018), 46–55.

[34] Ariadna Quattoni and Antonio Torralba. 2009. Recognizing indoor scenes. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 413–420.

[35] Pei Ren, Xiuquan Qiao, Yakun Huang, Ling Liu, Calton Pu, and Schahram Dustdar. 2021. Fine-grained elastic partitioning for distributed dnn towards mobile web ar services in the 5g era. *IEEE Transactions on Services Computing* (2021).

[36] Khe Chai Sim, Petr Zadrazil, and Françoise Beaufays. 2019. An investigation into on-device personalization of end-to-end automatic speech recognition models. *arXiv preprint arXiv:1909.06678* (2019).

[37] Sebastian U Stich. 2018. Local SGD converges fast and communicates little. *arXiv preprint arXiv:1805.09767* (2018).

[38] Nima Tajbakhsh, Jae Y Shin, Suryakanth R Gurudu, R Todd Hurst, Christopher B Kendall, Michael B Gotway, and Jianming Liang. 2016. Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE transactions on medical imaging* 35, 5 (2016), 1299–1312.

[39] Hemanth Venkateswara, Jose Eusebio, Shayok Chakraborty, and Sethuraman Panchanathan. 2017. Deep hashing network for unsupervised domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5018–5027.

[40] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. 2011. The caltech-ucsd birds-200-2011 dataset. (2011).

[41] Hao Wang, Zakhary Kaplan, Di Niu, and Baochun Li. 2020. Optimizing federated learning on non-iid data with reinforcement learning. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 1698–1707.

[42] Mengwei Xu, Feng Qian, Qiaozhu Mei, Kang Huang, and Xuanzhe Liu. 2018. Deeptype: On-device deep learning for input personalization service with minimal privacy concern. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2, 4 (2018), 1–26.

[43] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. 2018. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903* (2018).

[44] Xue Yang, Yan Feng, Weijun Fang, Jun Shao, Xiaohu Tang, Shu-Tao Xia, and Rongxing Lu. 2022. An Accuracy-Lossless Perturbation Method for Defending Privacy Attacks in Federated Learning. In *Proceedings of the ACM Web Conference 2022*. 732–742.

[45] Hao Yu, Sen Yang, and Shenghuo Zhu. 2019. Parallel restarted sgd with faster convergence and less communication: Demystifying why model averaging works for deep learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 5693–5700.

[46] Ziqi Zhang, Yuanchun Li, Bingyan Liu, Yifeng Cai, Ding Li, Yao Guo, and Xiangqun Chen. 2023. Protecting Federated Learning Models from Malicious Participants with Model Slicing. In *Proceedings of the 45th International Conference on Software Engineering*.

[47] Ziqi Zhang, Yuanchun Li, Jindong Wang, Bingyan Liu, Ding Li, Yao Guo, Xiangqun Chen, and Yunxin Liu. 2022. ReMoS: reducing defect inheritance in transfer learning via relevant model slicing. In *Proceedings of the 44th International Conference on Software Engineering*. 1856–1868.

[48] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. 2018. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582* (2018).

[49] Kuo Zhong, Ying Wei, Chun Yuan, Haoli Bai, and Junzhou Huang. 2020. TranSlider: Transfer Ensemble Learning from Exploitation to Exploration. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 368–378.

A THEORETICAL ANALYSIS

In this part, we provide theoretical analysis to support our framework. Specifically, we respectively analyze the effectiveness of the utilization-based comparison and the convergence property of the final federating strategy. To begin with, we make the following commonly used assumptions.

Assumption 1. *The objective functions $F_1, F_2, ..., F_N$ in each device are all L-smooth: for all* $\mathbf{v}$ *and* $\mathbf{w}$, $F_k(\mathbf{v}) \leq F_k(\mathbf{w}) + (\mathbf{v}-\mathbf{w})^T \nabla F_k(\mathbf{w}) + \frac{L}{2}\|\mathbf{v}-\mathbf{w}\|_2^2$.

Assumption 2. *The objective functions $F_1, F_2, ..., F_N$ in each device are all μ-strongly convex: for all* $\mathbf{v}$ *and* $\mathbf{w}$, $F_k(\mathbf{v}) \geq F_k(\mathbf{w}) + (\mathbf{v}-\mathbf{w})^T \nabla F_k(\mathbf{w}) + \frac{\mu}{2}\|\mathbf{v}-\mathbf{w}\|_2^2$.

Besides, we borrow the assumptions made in related works [37, 45] as follows.

Assumption 3. *Let ξ_t^k be sampled from the k-th device's local data uniformly at random. The variance of stochastic gradients in each device is bounded:* $\mathbb{E}\left\|\nabla F_k\left(\mathbf{w}_t^k, \xi_t^k\right) - \nabla F_k\left(\mathbf{w}_t^k\right)\right\|^2 \leq \sigma_k^2$, *for k=1,...,N.*

Assumption 4. *The expected squared norm of stochastic gradients is uniformly bounded, i.e.,* $\mathbb{E}\left\|\nabla F_k\left(\mathbf{w}_t^k, \xi_t^k\right)\right\|^2 \leq G^2$, *for all k=1,...,N and t=1,...,$T_{total}-1$. Here T_{total} represents the total number of every device's SGDs.*

According to a recent work [48], the *weight divergence* among the uploaded models will harm the aggregation performance, which can be formally defined as the following Theorem.

Theorem 1. *Suppose Assumption 1 to 2 hold and the synchronization is conducted every H steps. We have the following inequality for the weight divergence after the m-th synchronization,*

$$\begin{aligned}&\|\Delta \boldsymbol{w_m}\| \leq \\ &\sum_{k=1}^{N} \frac{sample^{(k)}}{\sum_{k=1}^{N} sample^{(k)}} (z^k)^T \|\Delta \boldsymbol{w_{m-1}}\| \\ &+ \eta \sum_{k=1}^{N} \frac{sample^{(k)}}{\sum_{k=1}^{N} sample^{(k)}} \left\|p^k - p^{global}\right\| \sum_{j=1}^{H-1} \\ &\left(z^k\right)^j \max_{i=1} \left\|\nabla_{\boldsymbol{w}} \mathbb{E}_{\boldsymbol{x}|y=i} \log f_i(\boldsymbol{x}, \boldsymbol{w})\right\|\end{aligned} \tag{6}$$

where $z^k = 1 + \eta \sum_{i=1}^{C} p^{(k)}(y=i)\lambda_{\boldsymbol{x}|y=i}$ and C is the number of category. p represents the data distribution. Here we omit the proof due to space limitations.

Based on the inequality, we can observe that the distribution heterogeneity among devices has a large impact on the final divergence degree. In our scenario, the data distribution among different user tasks might be extremely different, which can be regarded as a type of strong heterogeneity situation. Therefore, the *weight divergence* issue will inevitably exist in our fed-tuning process. We present a utilization-based comparison method to pick out parameters from similar tasks to conduct aggregation, which can significantly mitigate the distribution heterogeneity and decrease the weight divergence degree.

To further illustrate the convergence property of our federating strategy, we analyze the number of communications (i.e., federation rounds) in terms of the following Theorem proposed by other researchers [18].

Theorem 2. *Let Assumption 1 to 4 hold and the complexity of the communications can be formulated as,*

$$O\left[\frac{1}{\epsilon}\left(\left(1+\frac{1}{K}\right)EG^2 + \frac{\sum_{k=1}^{N} p_k^2 \sigma_k^2 + \Gamma + G^2}{E} + G^2\right)\right] \tag{7}$$

where Γ represents the heterogeneity degree of data distributions. Other constant definitions can be founded in the related paper [18], and we do not need them to carry on our analysis. The proof is omitted due to space limitations. Note that we only aggregate the parameters of similar tasks, which means that the value of Γ is extremely small compared to the traditional all-aggregation scheme. Therefore, the communication times can be theoretically reduced, and we are able to reach convergence at a faster speed.

B DETAILED EXPERIMENT SETTINGS

B.1 Constructed benchmarks.

We manually construct a benchmark with public computer vision (CV) and human activity recognition (HAR) datasets to simulate the adaptation for mobile/web scenarios. The detailed statistics are illustrated in Table 1 and we briefly introduce these scenarios as follows.

- *Task-specific* scenario, where different mobile/web users may have different task types. We employ three public datasets: CUB-200 [40], MIT Indoor-67 [34] and Caltech-256 [7], to simulate this scenario. Specifically, CUB-200 represents a fine-grained object recognition task with 200 species of birds. MIT Indoor-67 represents a scene recognition task with 67 indoor scene categories. Caltech-256 represents a general object recognition task with 256 object categories. We partition the training samples in each task into five parts, and each device is allocated to one part to denote its target task. Thus, this benchmark has 15 users with three types of tasks. Considering that Caltech-256 has no official train/test settings, we randomly pick out 20 disjoint samples of each category as the training set and testing set. The final performance is evaluated on the testing set of each task.
- *Environment-specific* scenario, where the task is identical while the environment background of each user may be significantly different. Here we employ two types of applications to simulate it. For CV applications, we pick out three types of background images: *artistic images*, *clipart images* and *product images* from the Office-Home dataset [39], which owns 65 identical object categories but different background distributions. Specifically, each image type is divided into a training set (80%) and a testing set (20%). Then each training set is further partitioned and allocated as the settings in the *task-specific* scenario. For HAR applications, we use the latest HAR-Depth dataset [31], where five types of gestures (good/ok/win/stop/fist) are recorded in this dataset by two subjects using a depth camera in three environments (i.e., outdoor, dark, and indoor). Specifically, we follow the official

partition settings, which have 3 scenes and simulate 9 nodes for FL.

- *User-specific* scenario, where different users may have different preferences (e.g., one may like collecting dog images while others may cat images). Here CIFAR-10 is used to simulate 100 clients, each of which belongs to one *type* that has two disjoint categories of data. Thus in total, we have 5 *types*. Note that the heterogeneity of data distribution is more apparent under this scenario since user preference is more diverse.