Feature-based Federated Transfer Learning: Communication Efficiency, Robustness and Privacy

Feng Wang, M. Cenk Gursoy and Senem Velipasalar

***Abstract*—In this paper, we propose feature-based federated transfer learning as a novel approach to improve communication efficiency by reducing the uplink payload by multiple orders of magnitude compared to that of existing approaches in federated learning and federated transfer learning. Specifically, in the proposed feature-based federated learning, we design the extracted features and outputs to be uploaded instead of parameter updates. For this distributed learning model, we determine the required payload and provide comparisons with the existing schemes. Subsequently, we analyze the robustness of feature-based federated transfer learning against packet loss, data insufficiency, and quantization. Finally, we address privacy considerations by defining and analyzing label privacy leakage and feature privacy leakage, and investigating mitigating approaches. For all aforementioned analyses, we evaluate the performance of the proposed learning scheme via experiments on an image classification task and a natural language processing task to demonstrate its effectiveness.** [1]

***Index Terms*—Federated learning, transfer learning, communication efficiency, robustness, privacy**

I. Introduction

Federated learning (FL) is a form of distributed learning in which only model parameters are exchanged while datasets are kept local with the goal to maintain data privacy [1]. Typically, in FL, a central server, known as the parameter server (PS), coordinates the collaborative training of a deep neural network (DNN) model [2] by aggregating updates on the weights and biases from multiple participating devices/clients. The widespread use of mobile phones and tablets with sufficient computational power and wireless communication capability enables FL in a wide range of applications such as speech recognition and image classification. Internet of Things (IoT) with large number of devices/sensors may generate even larger amount of data while casting further constraints on the computational power and transmission power [3]. With these, FL has gained widespread attention from both academic and industrial communities, resulting in a rapid increase in research on FL techniques, such as federated averaging (FedAvg) [4], federated transfer learning (FTL) [5], [6], and FL with differential privacy (DP) [7]. Among them, model-based FTL stands out as a particularly efficient scheme, because it transfers a well-trained source model into the target task of interest where samples may have different input and output spaces, and thus FTL requires fewer training samples and shortens the training process [8]–[11]. Especially, with the availability of open-source big data repositories and deep learning models [12], [13], transfer learning (TL) has become an attractive solution for various applications, such as text sentiment classification [14]–[16], image classification [17]–[22], human activity classification [23]–[25], software defect classification [26], [27], and multi-language text classification [28]–[30].

However, when the clients in FL are mobile devices and the training process is performed over a wireless network such as a wireless local area network (WLAN) or cellular system, it is crucial to minimize the data sent during updates of the model parameters. This is due to the limited availability of radio spectrum, and the uncertain nature of the wireless environment caused by factors such as channel fading and interference [31]. In the context of FL applications on IoT devices, the added constraints on computation and power consumption for uploading data make it even more challenging to perform complex tasks that require deep neural networks (DNNs) with many layers and a large number of trainable parameters. While many existing FL studies have focused on shallow DNNs with a few layers, state-of-the-art DNN models used in various applications often have dozens of layers and millions or billions of trainable parameters in order to achieve the highest accuracy in e.g., image segmentation [32], [33], image classification [34], [35], object detection [36], [37], question answering [38], medical image segmentation [39], and speech recognition [40].

A. Contributions

In response to the above-mentioned constraints on the uplink payload budget and limitations on the local computational power, we develop a novel scheme to perform model-based transfer FL and refer to this scheme as feature-based federated transfer learning (FbFTL). In FbFTL, rather than uploading the gradients, the input and output of the subset of DNN to be trained are uploaded with the goal to reduce the uplink payload. This is one of the key differences from prior FL and FTL strategies. In this paper, after describing the proposed FbFTL scheme, we analyze its overall uplink payload and provide comparisons with prior schemes to quantify the gains. Specifically, we test this approach by transferring the VGG-16 convolutional neural network (CNN) model [41] trained with ImageNet dataset [42] to CIFAR-10 dataset [43], and show that FL, two types of FTL, and FbFTL require uploading 3216 Tb, 949.5 Tb, 599 Tb, and 6.6 Gb of data, respectively, until performance convergence. With this, we demonstrate that the proposed FbFTL outperforms FL and FTL substantially by reducing the upload requirements by at least 5 orders of magnitude. We note that the ITU standard of 5G uplink

The authors are with the Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY, 13244. E-mail: fwang26@syr.edu, mcgursoy@syr.edu, svelipas@syr.edu.

The material in this paper has been presented in part at the IEEE Global Communications Conference (Globecom), Dec. 2022.

[1]Code implementation available at: https://github.com/wfwf10/Feature-based-Federated-Transfer-Learning.

user experienced data rate is only 50 Mb/s [44], and even the 6G uplink user experienced data rate at Gbit/s level may not sufficiently support such huge uploading requirements of regular FL. Additionally, to our best knowledge, existing works on improving FL efficiency (such as FedAvg [4], sparsification [45], [46] and quantization [47], [48] with or without error feedback [49], [50], federated distillation [51]–[53], pruning [54], [55] or partially trainable network [56], [57], and over-the-air computation [58]–[60]) still consider the transmission of gradient updates and can achieve a relatively limited reduction in payload and experience degradation in performance. For instance, the payload reduction is of only two orders of magnitude of the original payload on the same CIFAR-10 dataset [61], [62]. Therefore, FbFTL is still a more effective approach in reducing the uplink payload even after the efficiency of FL has been improved via the aforementioned methods. We further show that FbFTL has substantially lower downlink payload, and requires significantly less computational power from the edge devices, and therefore it is much more friendly in terms of facilitating the training tasks on clients with limited power budget.

To validate our approach, we further analyze the robustness of FbFTL against FL and FTL. We show significant reduction on packet loss rate (PLR) with FbFTL with the same block loss rate (BLR), and demonstrate the robustness of FbFTL with insufficient training data. While there have been numerous studies on gradient quantization and sparsification to reduce the uplink payload [63], [64], we illustrate that FbFTL also achieves significant compression rate by quantization while the validation accuracy is barely affected. Furthermore, we analyze the privacy leakage to a potential adversary. We define the label privacy leakage and feature privacy leakage for both feature-based and gradient-based frameworks. To our best knowledge, this is the first work that divides the privacy into different categories, defines each type, and proposes mitigation approaches. Via experimental results, we show that FbFTL has better performance (e.g., classification accuracy) with the same level of privacy protection when each client obtains a small set of samples.

In summary, our main contribution can be listed as follows:

- We propose the FbFTL scheme that uploads extracted features instead of gradients to reduce uplink payload by five orders of magnitude.
- We analyze the robustness of FbFTL, and demonstrate that it maintains the payload advantage when strategies such as sparsification, quantization and error feedback are deployed.
- We study the privacy guarantees in terms of entropy based formulations that quantify the uncertainty and also by utilizing differential privacy mechanisms.
- We show that a small batch size is preferred to protect label privacy of shuffled batches, and FbFTL has better performance (such as in terms of accuracy) given the differential privacy constraint for input privacy in a small batch.

B. Organization and Notations

The remainder of the paper is organized as follows. In Section II, we discuss the preliminary concepts regarding

Table I: Notations

FL Parameters:	Section II-A
$\mathcal{U}$, U	Set of clients, Number of clients
$\mathcal{K}_u$	Set of local training samples at client u
K_u	Number of local training samples at client u
$\mathbf{s}_{u,k}$	The kth sample at client u
$\mathbf{x}_{u,k}$	Input vector of $\mathbf{s}_{u,k}$ with N_0 attributes
$\mathbf{y}_{u,k}$	Output vector of $\mathbf{s}_{u,k}$ with N attributes (or classes)
$f_{\boldsymbol{\theta}}(\mathbf{x})$	DNN that maps the input $\mathbf{x}$ to estimated output $\hat{\mathbf{y}}$
$\boldsymbol{\theta}$	Trainable parameter vector of DNN
$L(f_{\boldsymbol{\theta}}\|\mathbf{s}_{u,k})$	Loss function
$\mathbf{g}_u$	Sum of updates (or gradients) at client u
α	Learning rate
I	Number of communication iterations during training process (superscript indicates methods)
C	Fraction of clients selected in each iteration
FTL Parameters:	Section II-B, III-A, III-B
M'	Number of layers without trainable parameters
M	Number of layers with trainable parameters
m_c	Index of the layer partitioning DNN into feature extraction part and task-specific part
$f^1_{\boldsymbol{\theta}^1}$	Feature extraction sub-model with trainable parameters $\boldsymbol{\theta}^1$
$f^2_{\boldsymbol{\theta}^2}$	Task-specific sub-model with trainable parameters $\boldsymbol{\theta}^2$
$\mathbf{z}_{u,k}$	Extracted features $f^1_{\boldsymbol{\theta}^1}(\mathbf{x}_{u,k})$
T_m	Number of trainable parameters in the mth trainable layer
N_m^-	Number of input nodes at the mth trainable layer
N_m^+	Number of output nodes at the mth trainable layer
Performance Measurements:	Section II-A, III-B, ,III-C
d	Payload of each float number in bits
P	Uplink payload (superscript indicates methods)
D	Downlink payload (superscript indicates methods)
$O(X)$	Computation time complexity for training
Robustness Measurements:	Section IV
PLR	Packet loss rate
BLR	Block loss rate
n_b	Number of blocks in each packet
n_r	Maximum number of packet retransmission allowed
$\mathcal{S}_r(\cdot)$	Sparsification function keeping a ratio r of the input elements
$\mathcal{Q}_q(\cdot)$	Quantization function keeping q bits of the input elements
$\mathbf{m}_{u,t}$	Local memory vector for error feedback
Privacy Measurements:	Section V
$H(\cdot)$	Entropy to quantify the uncertainty
N	Output dimension (or number of labels)
$\mathcal{B}$	Set of possible batches
$\|\mathcal{B}\|$	Number of possible batches
B	Unknown batch type of client u as a random variable
P_y	Distribution over labels
$P_{y,\text{uni}}$	Uniform label distribution
$n^y_{a,b}$	Count of output label type a in a batch of type b
$\mathbf{P}_{B\|\text{uni}}$	Adversary's presumed probability distribution of batches without prior knowledge
$H(B \mid P_{y,\text{uni}})$	Adversary's uncertainty of a batch without prior knowledge of the content
$P_{y,\text{true}}$	True label distribution
$\mathbf{P}_{B\|\text{true}}$	Adversary's presumed probability distribution of batches given the true sample distribution
$H\{B \mid P_{y,\text{true}}\}$	Adversary's uncertainty of a batch given the true sample distribution $P_{y,\text{true}}$
L^l_s	Label Privacy Leakage with Statistical Information
$c(b)$	Count for batches of different types among shuffled batches
$\mathcal{C}_{\mathcal{U}}$	Number of each batch type b in shuffled batches as a set of random variables
$\mathbf{P}_{B,\text{emp}}$	Adversary's presumed empirical distribution of batches given shuffled batches $\mathcal{C}_{\mathcal{U}} = c_U = [c(1), c(2), \ldots, c(\|\mathcal{B}\|)]$
$H(B\|\mathcal{C}_{\mathcal{U}} = c_U)$	Adversary's uncertainty of a batch given shuffled batches $\mathcal{C}_{\mathcal{U}}$
L^l_q	Label Privacy Leakage with Query
L^l_t	Total Label Privacy Leakage
$\mathbf{n}$	Additive independent Gaussian noise to protect local instance privacy, where $\mathbf{n} \sim \mathcal{N}(0,\, \sigma^2 R^2 \text{std}^2(g(\mathbf{d}_1))\mathbf{I})$
$\mathbf{d}$	Training samples for one batch
$g(\mathbf{d})$	Uploading vector in one batch
(ϵ, δ)	Differential privacy level
R	Maximum relative distance
Additionally:	f in subscript ($\star_f$) denotes transfer learning updating full model, and c in subscript ($\star_c$) denotes transfer learning updating task-specific sub-model

FL and FTL, describe the system design via FedAvg, and demonstrate their uplink payload. In Section III, we introduce the proposed FbFTL algorithm, introduce the learning structure and system design, and compare its payload with FL and FTL. In this section, we also evaluate the performance of FbFTL via simulations, and provide comparisons with FL and FTL. In Section IV, we illustrate the robustness of FbFTL in the presence of packet loss, data insufficiency, and quantization. In Section V, we define the label privacy leakage and the feature privacy leakage. For label privacy, we discuss the leakage with statistical information and the leakage with query, and investigate mitigation approaches to avoid these privacy leakages. For feature privacy leakage, we illustrate the mitigation approach via differential privacy. Finally, we conclude the paper in Section VI. The list of all notations in the paper is provided in Table I as a quick reference.

II. Preliminaries

In this section, we introduce preliminary concepts related to FL and FTL, and analyze the requirements on their successful uplink payload.

A. Federated Learning

We address a common FL task in which a PS coordinates a set $\mathcal{U}$ of U clients to cooperatively train a DNN model. Each client u possesses a local dataset $\mathcal{K}_u$ with K_u samples for training. In this dataset, the kth sample $\mathbf{s}_{u,k} \in \mathcal{K}_u$ is comprised of an input vector $\mathbf{x}_{u,k} \in \mathbb{R}^{N_0}$ and an output vector $\mathbf{y}_{u,k} \in \mathbb{R}^{N}$. The DNN, represented by the function $f_{\boldsymbol{\theta}}(\mathbf{x})$, maps the input $\mathbf{x}$ to an output $\hat{\mathbf{y}}$ (as an estimate of $\mathbf{y}$) with trainable parameter vector $\boldsymbol{\theta}$. The goal of the FL training process is to update the parameter vector $\boldsymbol{\theta}$ using the training samples from each client in order to minimize the expected loss $\mathbb{E}(L(f_{\boldsymbol{\theta}}|\mathbf{s}_{test}))$ on the unseen sample $\mathbf{s}_{test}$ with the same distribution as the training samples. For most DNNs with classification tasks, the output label $\mathbf{y}$ is an axis-aligned unit vector with one element equal to 1 indicating the class of this sample, and all others equal to 0. In this case, the loss function is typically the categorical cross-entropy:

$$L(f_{\boldsymbol{\theta}}|\mathbf{s}_{u,k}) = -\sum_{n=1}^{N} \boldsymbol{y}_{u,k}[n] \log f_{\boldsymbol{\theta}}(\boldsymbol{x}_{u,k})[n]. \tag{1}$$

In order to minimize the loss and keep the training samples local, the authors in [4] proposed an iterative distributed optimization scheme called FedAvg with the following steps:

1) The PS initiates trainable parameters $\boldsymbol{\theta}$, and broadcasts the model structure f with non-trainable parameters to each client.
2) The PS chooses a subset of UC clients (with C denoting the fraction of clients in this iteration) and broadcasts the trainable parameters $\boldsymbol{\theta}$.
3) Each client performs stochastic gradient descent (SGD) to obtain the parameter update corresponding to each training sample: $\nabla_{\boldsymbol{\theta}} L(f_{\boldsymbol{\theta}}|\mathbf{s}_{u,k})$.
4) Each client sends the sum of the updates over all local samples $\mathbf{g}_u = \sum_{k=1}^{K_u} \nabla_{\boldsymbol{\theta}} L(f_{\boldsymbol{\theta}}|\mathbf{s}_{u,k})$ and K_u to the PS.
5) The PS updates the parameter $\boldsymbol{\theta}$ with $\boldsymbol{\theta} - \sum_{u=1}^{U} \frac{\alpha}{K_u} \mathbf{g}_u$ where α is the learning rate that controls the training speed.
6) Return to step 2) until convergence.

Let us assume that the number of iterations in FL with FedAvg is I^{FL} (in the absence of any processing and transmission failures). This implies that the total number of times the clients upload $\mathbf{g}_u$ is $I^{FL}UC$ within the training process. Assume further that the DNN includes M layers with trainable parameters (such as convolutional layer and fully connected layer, i.e., dense layer) and M' layers without trainable parameters (such as pooling layer and residue connection). The mth trainable layer has T_m trainable parameters. Thus, for each trainable parameter, the client uploads one float number of d bits for the corresponding element in the update $\mathbf{g}_u$ during each iteration. Therefore, the overall uplink payload to train a DNN via FL with FedAvg is

$$P^{FL} = dI^{FL}UC \sum_{m=1}^{M} T_m \text{ bits}, \tag{2}$$

and we will set this as a benchmark to compare with three other training methods in the remainder of this paper.

B. Federated Transfer Learning

TL is an effective learning technique that utilizes knowledge from a different domain to enhance the performance in the target domain. FTL incorporates the privacy-preserving distributed learning paradigm into conventional TL in order to address the challenges of spectrum limitations in wireless applications and training data scarcity. In this paper, we consider FTL to be performed in the same fashion as in the previous subsection, where one PS orchestrates U clients. Based on the difference between the domains, FTL can be divided into three different categories: instance-based FTL, feature-based FTL, and model-based FTL [6], [65]. The former two types of FTL assume similarity in the distribution of the input and output, and hence we in this paper focus on the model-based FTL which only assumes similarity in the functionality to extract a high-dimensional description from the input data.

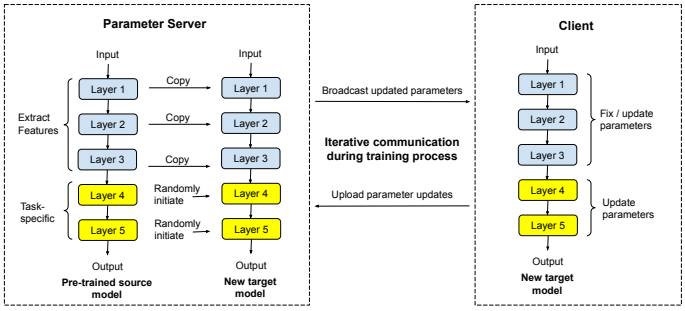


Figure 1: Diagram of the iterative training process of model-based federated transfer learning.

As shown in Fig. 1, we consider FTL with a DNN model pre-trained with an open-source dataset that corresponds to a similar but not the same task as the source model. The goal here is to reduce the number of iterations during training. Many DNNs can be partitioned into two sections. The first section generates the high-dimensional features from the sample input data with general information. The second section carries

out operations for a particular task, such as classification, and is less likely to be transferable to a new task. To move the knowledge of the source model into a new task before training, the feature extraction section of the source model is directly transferred to the new target model, while the task-specific part is randomly initiated. FTL can be performed by retraining all the parameters with new data. However, there typically exist a large number of parameters to train, and in order to reduce the training time and data requirements, a common approach is to perform FTL by fixing the feature extraction part and simply updating the task-specific part. In particular, we select one fully connected layer m_c (which is Layer 4 in Fig. 1), close to the output without a paralleling path (such as residue connection), and randomly initiate all trainable parameters of layers $m_c, m_c+1, \ldots, M$ (which are Layer 4 and Layer 5 in Fig. 1). During the distributed training process, all copied parameters are fixed, and we only update the parameters of layers $m_c, \ldots, M$.

Similar to the previous sub-section on FL, we determine the least requirements on successful uplink payload for FTL until convergence. For FTL that updates all parameters, assuming that FTL with FedAvg is iterated I_f^{FTL} times, the overall uplink payload required for training is

$$P_f^{FTL} = dI_f^{FTL}UC \sum_{m=1}^{M} T_m \text{ bits}. \tag{3}$$

On the other hand, assuming FTL with FedAvg that only updates the task-specific part with I_c^{FTL} iterations and assuming that each sum update $\mathbf{g}_u$ consists of $\sum_{m=m_c}^{M} T_m$ trainable parameters, the overall uplink payload for training is

$$P_c^{FTL} = dI_c^{FTL}UC \sum_{m=m_c}^{M} T_m \text{ bits}. \tag{4}$$

Typically, training from scratch requires more training samples, and thus we have $I_c^{FTL}UC \approx I_f^{FTL}UC < I^{FL}UC$. For most of the models, the majority of the DNN structure is dedicated to the feature extraction, and obviously $\sum_{m=m_c}^{M} T_m < \sum_{m=1}^{M} T_m$. Therefore,

$$P_c^{FTL} < P_f^{FTL} < P^{FL}. \tag{5}$$

C. Other Recent Transfer and Federated Learning Schemes

While we in this paper focus on FL schemes that generalize to the majority of use cases, there are also recent works that improve the performance under certain assumptions and can be used in FL. We list these well-known methods, along with FL, FTL, and the proposed FbFTL, in Table II and provide a comparison. Below, we briefly describe the key assumptions of these methods, and highlight the differences of the proposed FbFTL scheme.

Specifically, elastic weight consolidation (EWC) [66] focuses on a large number of different tasks simultaneously and the transfer between them by remembering the importance of each weight. The performance is typically measured on all experienced tasks. EWC is inspired by neuroscience and Bayesian inference, and it aims to quantify the importance of each weight to the tasks the model has previously learned. The fundamental idea is that the weights critical to prior tasks should be altered less when new data is encountered. In this paper, we consider a different problem, where we transfer to a single target task from a single source model on another task whose performance is no longer considered. In this problem setting, there is no clear difference between EWC and plain SGD in [66] in terms of both training performance or training time, but EWC does introduce extra communication payload and computation cost in FL. On the contrary, FbFTL achieves five orders of magnitude payload reduction.

Split federated learning (SFL) [67] partitions the neural network into two segments: the initial part near the input undergoes training in a federated learning manner using a "Fed Server", and the subsequent part near the output is trained using split learning with a "Main Server," which receives the smashed data. Compared to FbFTL, the federated learning segment utilizing the Fed Server operates in the same way as FL and requires significant uplink and downlink payload. Hence, this part alone is typically much more communication-expensive compared to FbFTL. The advantage of SFL lies in its similar performance to FL and not needing a pre-trained model as it updates all parameters. We will present the experimental performance comparison in Table III in Section III-D.

Personalized federated learning (PFL) [68] considers clients with large number of heterogeneous local samples, enabling generalization to the heterogeneous local distribution. However, we in this paper consider a large number of clients with a small set of local samples from each. As explained in Section V-A and Section V-C, such a setting better protects label privacy. Therefore, personalized federated learning focuses on a different problem from ours. We cannot apply this approach to FbFTL because personalized FbFTL violates the shuffle-batch assumption, unless we train the global model by FbFTL and then retrain locally. Although PFL has better local performance than FL with large heterogeneous batches, it requires several independent local batches and significantly more local computation. Furthermore, it does not allow quantization and sparsification since the weights instead of gradients are uploaded. Therefore, it is comparatively even less communication-efficient when quantization and sparsification are deployed in other methods, as shown in Section IV-C and Table V.

III. Feature-based Federated Transfer Learning

In this section, we describe the proposed FbFTL framework and demonstrate that it requires a substantially smaller uplink payload compared to FL and FTL. This efficiency arises from the fact that the extracted features and outputs are uploaded rather than the parameter updates.

A. Learning Structure

As depicted in Fig. 2, in FbFTL we consider the model-based TL on a source DNN model f' pre-trained on a different task. We choose one fully connected layer m_c without paralleling path as the cut layer, and divide the new target model $f \leftarrow f'$ into two parts. Those layers before layer m_c (which is layer 4 in Fig. 2, i.e., $m_c = 4$) are regarded as the feature extraction sub-model $f^1_{\boldsymbol{\theta}^1}$, and the parameters $\boldsymbol{\theta}^1$ for the new target model f are copied from the pre-trained

Method	Payload	Computation	Key Assumption
FL	iterative gradient-level	medium, distributed	start with random initialization
FTL	iterative gradient-level	low, distributed	start with source model
FbFTL (ours)	**one-time feature-level**	low, mostly centralized	start with source model
EWC [66]	iterative gradient-level	high, distributed	evaluating many different tasks
SFL [67]	iterative gradient-level	medium, hybrid of distributed and centralized	two servers needed
PFL [68]	iterative gradient-level	very high, distributed	large and heterogeneous local batch

Table II: Comparison between different federated learning methods

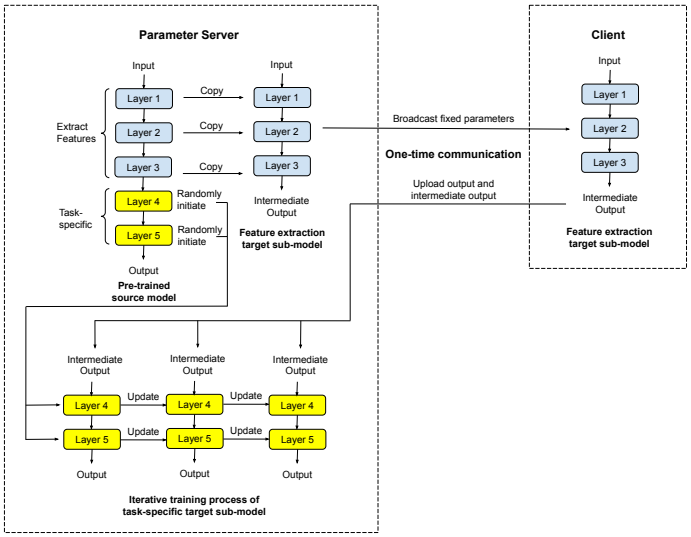


Figure 2: Diagram of the training process of feature-based federated transfer learning, which reduces to one-time communication of the output and the intermediate output (i.e., extracted features).

source model f' and fixed without any further update. The other layers are regarded as the task-specific sub-model $f^2_{\boldsymbol{\theta}^2}$, and all trainable parameters $\boldsymbol{\theta}^2$ in the new target model f are randomly initiated for training with the uth client's kth training sample $\mathbf{s}_{u,k} = \{\mathbf{x}_{u,k}, \mathbf{y}_{u,k}\}$ for all u and k.

The forward pass of the full model $f_{\boldsymbol{\theta}}(\mathbf{x}_{u,k}) = f^2_{\boldsymbol{\theta}^2}(f^1_{\boldsymbol{\theta}^1}(\mathbf{x}_{u,k}))$ maps the input $\mathbf{x}_{u,k}$ to the estimated output $\hat{\mathbf{y}}_{u,k}$, and we note that the output of the feature extraction sub-model $\mathbf{z}_{u,k} = f^1_{\boldsymbol{\theta}^1}(\mathbf{x}_{u,k})$ is also the input of the task-specific sub-model $f^2_{\boldsymbol{\theta}^2}$. Note that we require only the task-specific sub-model to be trained. In this setting, each client generates the features $\mathbf{z}_{u,k}$ from input $\mathbf{x}_{u,k}$ and sends these features to the PS only once. Subsequently, in FbFTL, the PS performs the gradient back-propagation iteratively without sending any feedback to the clients. Contrary to this, in FL and FTL, the parameter update, which is based on the gradients, highly relies on the parameters in the current training iteration, and the same data sample may generate different parameter updates in different iterations within the training process. Therefore, in FL and FTL, we either require many more training samples, or need to upload updates multiple times for the same sample. However in FbFTL, each client only needs to upload the intermediate output $\mathbf{z}_{u,k}$ and output $\mathbf{y}_{u,k}$ once, instead of iteratively uploading the gradients. The PS may deem these as the input and the output of the training sample for the task-specific sub-model $f^2_{\boldsymbol{\theta}^2}$. Such pairs of samples are not correlated with the model parameters $\boldsymbol{\theta}^2$, and therefore they can be used in different training iterations without downloading or uploading anything again. We provide the steps of the FbFTL algorithm in Algorithm 1 below.

Algorithm 1 FbFTL

PS copies fixed sub-model $f^1_{\boldsymbol{\theta}^1}$ from pre-trained source model and randomly initiates trainable sub-model $f^2_{\boldsymbol{\theta}^2}$.
PS broadcasts $f^1_{\boldsymbol{\theta}^1}$ to each client $u \in \mathcal{U}$
Clients execute:
for client $u \in \mathcal{U}$ **do**
  for sample $\mathbf{s}_{u,k} \in \mathcal{K}_u$ **do**
    Upload $\mathbf{z}_{u,k} = f^1_{\boldsymbol{\theta}^1}(\mathbf{x}_{u,k})$ and $\mathbf{y}_{u,k}$ to PS
  end for
end for
PS executes:
while *not converge* **do**
  for mini-batch b of pairs $\{u,k\} \in \mathcal{U} \times \mathcal{K}_u$ **do**
    $\boldsymbol{\theta}^2 \leftarrow \boldsymbol{\theta}^2 - \alpha \sum_{\{u,k\}\in b} \nabla_{\boldsymbol{\theta}^2} L(f_{\boldsymbol{\theta}^2}|\mathbf{z}_{u,k}, \mathbf{y}_{u,k})$
  end for
end while

As emphasized above, a key distinction of FbFTL is that the training samples of task-specific sub-model are uploaded rather than the direct gradient updates. With this, FbFTL provides additional important benefits that can lead to further improvements in the training performance and efficient management in practice. One of the most important benefits is the significant reduction of packet loss rate given the same batch loss rate since FbFTL has much less data in each batch, as we will illustrate in detail in Section IV-A. One other benefit is the waiving of the requirement of synchronization between clients, since there is only one iteration in FbFTL. The other benefits pertain to hyper-parameter fine-tuning, dataset balancing, and enabling flexible training batch size selection, as detailed below.

Specifically, one of the most important additional benefit is that FbFTL enables iterative fine-tuning of the SGD optimizer hyper-parameters such as the learning rate. To obtain the optimized DNN, one needs to find the optimized hyper-parameters that provide the best convergence performance. In FbFTL, the model can be trained from scratch several times at the PS to identify the best hyper-parameter setting without additional communication with the clients. On the other hand, in FL and FTL, the entire online training process has to be run several times, resulting in a much higher cost compared to the ideal uplink payload P.

Another benefit of FbFTL is the dataset balancing. If the overall dataset is imbalanced and hence the samples with

certain types of output appears much more frequently than those of other outputs, it is hard for FL and FTL to distinguish this via gradient updates, and such imbalanced data distribution could significantly degrade FL performance [69], [70]. However, FbFTL with direct output information enables techniques, such as re-sampling specific classes or merging near-identical classes, to improve dataset imbalance.

One more benefit of FbFTL is to lift the constraint of UC and K_u on the training batch size. To avoid over-fitting to the training dataset, one needs to validate the performance on a separate validation set of data to identify the optimal number of training iterations to stop training and conclude the final model. It is straightforward for FbFTL to divide the obtained dataset into training and validation subsets. However, the gradient-based FL frameworks call for extra effort for the communication system to meticulously perform the training process and validation process with the desired order and number of samples. Additionally for the training process, due to the broadcast nature of the downlink in wireless FL and FTL, all selected clients in the same communication iteration receive the same parameters. Hence, each SGD mini-batch has a larger size than $\sum_{u=1}^{UC} K_u$, and an overwhelmingly large SGD mini-batch size may delay the training process and require more training iterations. The mini-batches in these gradient-based FL frameworks are also on the order of given clients' samples. Therefore, when the clients' sample distribution is biased, we cannot shuffle the samples between iterations and have to accept the loss in the final performance. However, FbFTL may choose any rational size of SGD mini-batch without the constraints of the communication system, and can reshuffle the data in each training iteration.

B. Payload Analysis

Note that FbFTL has the major benefit of requiring one-time communication between the clients and the PS. In addition to this, FbFTL has much less data in a single upload batch compared to that of each sample in the upload batch of FL and FTL. Let us assume that in the fully connected layer m with bias, the number of input nodes and the number of output nodes are denoted by N_m^- and N_m^+, respectively Then, the number of trainable parameters in this layer is given by $T_m = N_m^+(N_m^- + 1)$. Compared to the dimension of $N_{m_c}^-$, the amount of information that we need to represent $\mathbf{y}_{u,k} \in \{1, \ldots, N\}$, i.e., $\log_2 N$ bits, is negligible. Note that in FbFTL, gradients are computed at the PS. Therefore, FedAvg cannot be applied and each sample is required to be uploaded separately. Therefore for FbFTL, the uplink payload for each sample is $dN_{m_c}^-$, and the overall successful uplink payload required for training is

$$P^{FbFTL} = d \sum_{u=1}^{U} K_u N_{m_c}^- \text{ bits}. \tag{6}$$

Compared to the uplink payload in (4) of FTL with FedAvg (in which only the task-specific sub-model is updated), the ratio of each upload for single sample between FTL and FbFTL is

$$\begin{aligned} \frac{d\sum_{m=m_c}^{M} T_m}{dN_{m_c}^-} &= \frac{T_{m_c} + \sum_{m=m_c+1}^{M} T_m}{N_{m_c}^-} \\ &= \frac{N_{m_c}^+(N_{m_c}^- + 1) + \sum_{m=m_c+1}^{M} T_m}{N_{m_c}^-} \\ &> N_{m_c}^+. \end{aligned} \tag{7}$$

Therefore,

$$\frac{P_c^{FTL}}{P^{FbFTL}} = \frac{dI_c^{FTL} UC \sum_{m=m_c}^{M} T_m}{d\sum_{u=1}^{U} K_u N_{m_c}^-} > \frac{I_c^{FTL} UC}{\sum_{u=1}^{U} K_u} N_{m_c}^+. \tag{8}$$

The number of extracted features for many state-of-the-art models is larger than 10^3, and TL usually requires a relatively deeper task-specific sub-model, and therefore $N_{m_c}^+$ can be large. For the cross-device federated learning, the clients do not obtain huge local datasets, $\sum_{u=1}^{U} K_u < I_c^{FTL} UC$. Therefore, we have $P^{FbFTL} \ll P_c^{FTL}$. We note that FbFTL and FTL that updates the task-specific sub-model have the same performance at every iteration because the only difference between the two methods is the communication model but not the numerical process to generate the gradient updates. Combining this observation with the conclusion in (5), we have

$$P^{FbFTL} \ll P_c^{FTL} < P_f^{FTL} < P^{FL}, \tag{9}$$

and therefore we expect extremely smaller uplink payload in FbFTL compared to FTL and FL.

We also note that FbFTL has the smallest downlink broadcast payload and the least local computation compared to FL and FTL. For FL, FTL that updates all parameters, FTL that updates the task-specific sub-model, and FbFTL, the overall downlink broadcast payloads, respectively, are

$$D^{FL} = dI^{FL} \sum_{m=1}^{M} T_m \text{ bits}, \tag{10}$$

$$D_f^{FTL} = dI_f^{FTL} \sum_{m=1}^{M} T_m \text{ bits}, \tag{11}$$

$$D_c^{FTL} = dI_c^{FTL} \sum_{m=1}^{M} T_m \text{ bits}, \tag{12}$$

$$D^{FbFTL} = d \sum_{m=1}^{m_c-1} T_m \text{ bits}. \tag{13}$$

C. Time Complexity Analysis

We note that FbFTL consumes less computation time and power for training in total, and also transfers a proportion of computation from the client devices to the PS. In FL and FTL, each client must complete one full forward pass and one back-propagation of the parameters to be trained for each training sample at each iteration. However, in FbFTL, each sample is only processed once by the client, and each client only needs to complete the forward pass of feature extraction sub-model, while all other computations are completed at the PS. Such shift of computational load to the PS is particularly beneficial if the end users and devices are severely limited in their

computational capabilities and power resources (for instance, in IoT networks) and the PS is equipped with advanced processors and has access to more resources.

In this subsection, we use time complexity to estimate the computation time and energy consumption. For each fully connected layer m_1 with $N^-_{m_1}$ input nodes and $N^+_{m_1}$ output nodes, there are $T_{m_1} = N^+_{m_1}(N^-_{m_1}+1)$ trainable parameters, and each sample requires $O(N^+_{m_1}(N^-_{m_1}+1)) = O(T_{m_1})$ time complexity for matrix multiplication during forward pass, and the same time complexity for matrix multiplication during back-propagation. Each 2-dimensional convolutional layer m_2 with stride 1 in each direction and same padding has the input shape $\{a^-, b^-, c^-\}$ where $N^-_{m_2} = a^- b^- c^-$, kernel shape $\{c^+, a', b', c^-\}$ where $T_{m_2} = c^+(a'b'c^- + 1)$, and output shape $\{a^-, b^-, c^+\}$ where $N^+_{m_2} = a^- b^- c^+$. Thus, each sample requires $O(c^+(a'b'c^-+1)a^-b^-) = O(a^-b^-T_{m_2})$ time complexity for matrix multiplication during forward pass, and the same during back-propagation. Comparatively, the time complexity of activation functions, max-pooling layers and dropout layers is negligible. Therefore, we denote the time complexity of each trainable layer m for a single training sample as $O(X_m)$, and the overall time complexity required for all clients as $O(X)$. Specifically, the overall time complexities at clients for the methods of FL, FTL that updates all parameters, FTL that updates the task-specific sub-model, and FbFTL, respectively, are

$$O(X^{FL}) \propto O\left(2I^{FL}UC\sum_{m=1}^{M} X_m\right), \tag{14}$$

$$O(X_f^{FTL}) \propto O\left(2I_f^{FTL}UC\sum_{m=1}^{M} X_m\right), \tag{15}$$

$$O(X_c^{FTL}) = O\left(U\sum_{m=1}^{m_c-1} X_m + 2I_c^{FTL}UC\sum_{m=m_c}^{M} X_m\right), \tag{16}$$

$$O(X^{FbFTL}) \propto O\left(U\sum_{m=1}^{m_c-1} X_m\right), \tag{17}$$

where $\propto$ stands for "proportional to". Typically, we have $X^{FbFTL} < X_c^{FTL} \ll X_f^{FTL} < X^{FL}$.

D. Experimental Results

In this section, we consider the application of FL, FTL and FbFTL to the VGG-16 CNN model [41] and transfer the knowledge learned from ImageNet dataset [42] to CIFAR-10 dataset [43].

ImageNet is a vast online database containing over 14 million images, each with hand-annotated labels describing the classification types or intended outputs for training. The pre-trained source model we use for TL was created on the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC2012, [71]), and the images come from 1000 different categories. As an illustration, we provide 10 samples with their labels in Fig. 3.

CIFAR-10 is a database of 60000 images, each with one of $N = 10$ distinct labels. Out of these images, 50000 images are used for training and 10000 images for testing. Fig. 4 showcases the first ten samples with their labels.

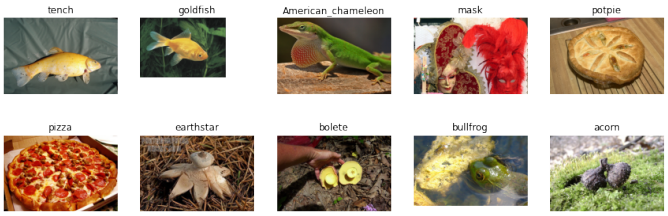


Figure 3: ImageNet samples with labels.

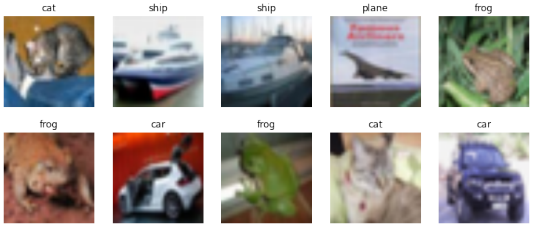


Figure 4: CIFAR-10 samples with labels.

Note that these samples have simple labels and appear more blurred compared to those in ImageNet due to their lower resolution/dimension.

In Fig. 5, we depict the structure of VGG-16 used for training on CIFAR-10. For TL, we consider the first half of the layers marked blue as the feature extraction sub-model $f^1_{\boldsymbol{\theta}^1}$ and directly transfer this sub-model from that trained on ImageNet. We deem the latter part marked yellow as the task-specific sub-model $f^2_{\boldsymbol{\theta}^2}$ and randomly initiate this part. For FbFTL, the dimension of the intermediate output is $N^-_{m_c} = 4096$.

To train the model on CIFAR-10, we utilize Nvidia GeForce GPU with CUDA to run the algorithms with PyTorch [72]. We assume that there are $U = 6250$ clients in total, each iteration takes a fraction $C = 1.28 \times 10^{-3}$ of all clients, and each

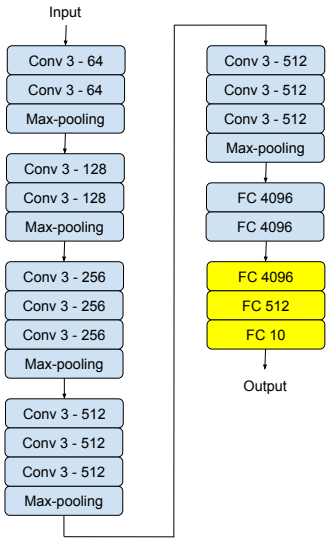


Figure 5: Diagram of the VGG-16 model for training on CIFAR-10 dataset. "Conv {receptive field size} - {number of output channels}" depicts the convolutional layers. "FC {output size}" depicts the fully connected layers.

batch contains $K_u = 8$ samples. In the two most commonly used deep learning tools TensorFlow (including Keras) [73] and PyTorch, the default data type of each number has 32 bits and therefore $d = 32$ bits. The learning rate is 10^{-2}, the momentum of the optimizer is 0.9, and the L2 penalty is 5×10^{-4}.

In Table III, we compare the performances of FL, SFL, FTL updating the full model (FTL$_f$), FTL updating the task-specific sub-model (FTL$_c$), and FbFTL. For increased fairness in the comparison of payloads, we further demonstrate the performances FL^{low} and FTL$_f^{low}$, which are the FL and FTL$_f$ algorithms that terminate training process when the validation accuracy reaches those of FTL$_c$ and FbFTL (i.e., $\approx 86\%$). As we have analyzed, the other algorithms require IUC successfully uploaded batches, while FbFTL only requires $\sum_{u=1}^{U} K_u$ batches. Also, FL, SFL, FTL$_f$ and FTL$_c$ require uploading $d\sum_{m=1}^{M} T_m$ bits, $d(K_u N_{m_c}^- + \sum_{m=m_c}^{M} T_m)$ bits, $d\sum_{m=1}^{M} T_m$ bits, and $d\sum_{m=m_c}^{M} T_m$ bits, respectively, for each batch, while FbFTL only requires uploading $dN_{m_c}^-$ bits for each batch. In the third row of the table, we observe that the FbFTL algorithm significantly reduces the uplink payload per batch by four orders of magnitude (i.e. a factor of 10^{-4}) compared to the other algorithms for each client. Additionally, in the fourth row, FbFTL leads to a reduction of five orders of magnitude (i.e. a factor of 10^{-5}) in the total uplink payload during training when compared to the other algorithms. These results demonstrate that FbFTL is apparently the most efficient scheme. FbFTL also results in a substantial decrease in the overall downlink payload. If larger models are trained for more complex tasks or if the size of the training dataset is more limited, the difference in payload could be even greater. In (14) through (17), we have described the overall computation time complexity required for training at clients as $O(X)$. In the second-to-last row of the table, we quantify and provide this time complexity for each method by counting the number of multiplications among floating numbers. We readily observe that FbFTL requires the least computation time and power consumption at the clients, and has two orders of magnitude reduction compared to FL, SFL and FTL$_f$, since it only runs the forward pass over feature extraction sub-model for each sample one time. We further note that for FbFTL, the computation complexity at the PS is also low with $X = 3.00 \times 10^{14}$. Furthermore, we note that different number of clients CU in each iteration does not affect the performance of FbFTL, but a large number of clients CU reduces the performance of FL and FTL, since it defines the minimum "training batch size" CUK_u. The performance of FL and FTL will decrease with an overwhelmingly large training batch size, which is another benefit of FbFTL. We in the experiment pick a small $CU = 8$ so that the benchmark schemes (i.e., FL, FTL$_f$, and FTL$_c$) have the best performance (at which point FTL$_c$ and FbFTL have the same performance). Moreover, we also observe that FTL$_c$ and FbFTL only update a small portion of the parameters and exhibit a slight decrease in validation accuracy, which is the trade-off for the reduced payload. However, we will show in Section IV and Section V that under communication efficiency or privacy constraints, FbFTL may also prevail in terms of validation accuracy in certain situations.

Furthermore, we note that in transfer learning, there is an additional trade-off between privacy protection, performance and payload. When partitioning a model into feature extraction sub-model and task-specific sub-model, choosing the cut layer closer to the output better preserves data privacy, while picking a cut layer closer to the input improves the training performance. In order to demonstrate such a trade-off, we next consider a language model as our application scenario. In FL, FTL, and FbFTL on natural language processing tasks, we consider tasks where the model and the intermediate features are not proprietary or private, and thus we assume that the clients are willing to share them while keeping the local data private As an example, we show the results on a conversation summary task SAMSum [74] with 32128 distinct token types out of 14732 training dialogues and 819 testing dialogues. For instance, dialogue ID 13728867 in SAMSum is "Olivia: Who are you voting for in this election? Oliver: Liberals as always. Olivia: Me too!! Oliver: Great", and ground truth summary is "Olivia and Olivier are voting for liberals in this election." For this task, we deploy a language model FLAN-T5-small [75], which is a transformer with 110 million parameters, including 8 encoders and 8 decoders. This model is pre-trained and the dataset is relatively small, so we do not have FL in this case. We assume that there are $U = 7366$ clients, each has $K_u = 2$ dialogues, and each iteration takes a fraction $C = 5.43 \times 10^{-4}$ of all clients. Our experiment is based on HuggingFace [76] with learning rate 2×10^{-4}.

In Table IV, we compare the performances of FTL and FbFTL in terms of ROUGE-1 score, which measures the match between the generated text and reference text. A larger ROUGE-1 indicates better performance. We note that FTL$_f$ trains all components including encoders, decoders, embedding and the final linear layer. On the other hand, FTL$_c$ and FbFTL freeze embedding, and may or may not freeze several encoders close to the input prompts. The number of trained encoders is given in the second row of Table IV. For instance, the performance results in the third and fourth columns are for FTL$_c$ and FbFTL that have trained all encoders (and hence have not frozen any of them), while the other columns provide the performances with 4 or 2 encoders trained (indicating that 4 or 6 encoders close to the input are frozen). We do not need to freeze decoders since label privacy leakage converges to zero with shuffled batches, as will be shown in Section V. In Table IV, we observe that FbFTL reduces the uplink and downlink payload by similar orders of magnitude as in the VGG-16 experiment. Furthermore, we notice that training less layers or encoders leads to a slight reduction in ROUGE-1 score but it does not guarantee lower payload. While FTL may require more iterations to arrive convergence, FbFTL may need to broadcast a larger feature extraction sub-model.

IV. Robustness Analysis

In this section, we compare the performance of FbFTL with that of FL and FTL under the same packet loss rate (PLR) for each batch being uploaded, and illustrate the robustness of FbFTL against packet losses, data insufficiency, and compression, including quantization, sparsification and error feedback.

	FL	SFL	FL^{low}	FTL$_f$	FTL$_f^{low}$	FTL$_c$	FbFTL
number of upload batches	656250	656250	68750	193750	25000	525000	50000
upload parameters per batch	153144650	117483328	153144650	153144650	153144650	35665418	4096
uplink payload per batch	**4.9 Gb**	**3.8 Gb**	**4.9 Gb**	**4.9 Gb**	**4.9 Gb**	**1.1 Gb**	**131 Kb**
total uplink payload P	**3216 Tb**	**2467 Tb**	**337 Tb**	**949 Tb**	**123 Tb**	**599 Tb**	**6.6 Gb**
total downlink payload D	402 Tb	308 Tb	42 Tb	253 Tb	15 Tb	322 Tb	3.8 Gb
computation time complexity X	1.63×10^{17}	1.63×10^{17}	1.7×10^{16}	4.80×10^{16}	6.19×10^{15}	1.07×10^{15}	7.74×10^{14}
validation accuracy	89.42%	89.42%	86.64%	93.75%	86.45%	86.51%	86.51%

Table III: Performance comparison on VGG-16 between FL, SFL, FTL updating full model (FTL$_f$), FTL updating task-specific sub-model (FTL$_c$) and FbFTL. Additionally, we compare with cases in which FL and FTL$_f$ achieve the same accuracy as FTL$_c$ and FbFTL ($\approx 86\%$) by terminating training slightly earlier. Algorithms in these cases are referred to as FL^{low} and FTL$_f^{low}$.

	FTL$_f$	FTL$_c$	FbFTL	FTL$_c$	FbFTL	FTL$_c$	FbFTL
number of trained encoders	8	8	8	4	4	2	2
number of upload batches	132588	36830	7366	88392	7366	103124	7366
upload parameters per batch	109860224	60511616	1024	51070144	1024	46349504	1024
uplink payload per batch	**3.5 Gb**	**1.9 Gb**	**32.7 Kb**	**1.6 Gb**	**32.7 Kb**	**1.5 Gb**	**32.7 Kb**
total uplink payload P	**466.1 Tb**	**71.3 Tb**	**241.4 Mb**	**144.5 Tb**	**241.4 Mb**	**152.9 Tb**	**241.4 Mb**
total downlink payload D	116.0 Tb	32.2 Tb	1.58 Gb	77.3 Tb	1.88 Gb	90.2 Tb	2.03 Gb
validation ROUGE-1	45.9249	45.4680	45.4680	45.2827	45.2827	44.9862	44.9862

Table IV: Performance comparison on FLAN-T5-small between FTL$_f$ updating full model, FTL$_c$ updating task-specific sub-model and FbFTL, with different number of encoders trained.

A. Packet Loss

As we have demonstrated in the previous section, gradient-based FedAvg FL frameworks including FL and FTL upload the gradient update $\mathbf{g}_u = \sum_{k=1}^{K_u} \nabla_{\boldsymbol{\theta}} L(f_{\boldsymbol{\theta}}|\mathbf{s}_{u,k})$ iteratively, where each batch contains the gradient summation from all samples of the client. In contrast, our proposed FbFTL uploads the data for each sample only once, and each batch contains extracted features $\mathbf{z}_{u,k}$ and output $\mathbf{y}_{u,k}$ from one sample.

In (7), we have shown that each batch for gradient-based FL is much larger than each batch for FbFTL (by about 10^4 larger in our experiments), and we assume that the packets to transmit both types of batches consist of multiple transmission blocks of the same size. For both types of packets, we consider measuring the robustness of all frameworks against packet loss caused by block losses due to network congestion or link outage (e.g., as a result of deep fading in wireless networks). For each learning framework whose packet consists of n_b transmission blocks, we consider the same block loss rate (BLR). The PLR without retransmission is

$$\text{PLR} = 1 - (1 - \text{BLR})^{n_b}. \tag{18}$$

Obviously, PLR highly depends on the value of n_b. FbFTL has significantly lower packet size and consequently we expect much lower PLR for the given same BLR. We show the significant performance difference among different learning frameworks in our experiments at the end of this section.

If, on the other hand, we allow at most n_r retransmissions for all packets and assume the channel state of each transmission to be independent and identically distributed (i.i.d.), we can lower the PLR and achieve similar packet-level reliability. However, this is realized at the cost of higher total uplink payload. Specifically for each learning framework requiring uplink payload P, the expected uplink payload with retransmissions is

$$P(\text{BLR}) = \sum_{n'=1}^{n_b} \frac{P}{n_b} \sum_{n=0}^{n_r} (\text{BLR})^n = P \sum_{n=0}^{n_r} (\text{BLR})^n. \tag{19}$$

We note that $\sum_{n=1}^{n_r}(\text{BLR})^n$ is the expected number of retransmissions, which is the same for all different learning frameworks. If the transmission has a fixed bandwidth, $\sum_{n=1}^{n_r}(\text{BLR})^n$ also represents the delay factor of the total transmission. We further note that if BLR $= 0$, then $P(\text{BLR}) = P$. Otherwise, $P(\text{BLR}) = P + P\sum_{n=1}^{n_r}(\text{BLR})^n > P$. Notice that the increase in the payload $P\sum_{n=1}^{n_r}(\text{BLR})^n$ grows with P and n_r. Hence, learning frameworks with higher payload experience a higher increase in payload when retransmissions are introduced.

B. Data Insufficiency

At the end of Section III-A, we have discussed the benefit that FbFTL does not require additional online uploads from the clients to test different sets of hyper-parameters, while gradient-based frameworks need to run the entire uploading process multiple times. In practice, in addition to hyper-parameters, another intangible aspect prior to training is whether there exists a sufficient number of participating clients and training samples. If the planned set of samples is insufficient to train the neural network, gradient-based frameworks require rerunning the overall process with more clients, which leads to high consumption and potential waste of both computational and transmission resources. However, FbFTL only requires the new clients to compute and upload their batches, and hence there is no waste of transmission resources.

C. Quantization, Sparsification and Error Feedback

To further reduce the uplink payload of gradient-based frameworks, there have been extensive works on gradient compression, including gradient sparsification [77], [78] and gradient quantization [63], [64], especially signSGD, the extreme case in which each element is reduced to be binary valued without scaling [79]. Several recent studies utilize error feedback (or quantization and sparsification with memory) that reduces the error in compression at client's local device to improve the updates in future iterations [49], [50].

For gradient-based frameworks utilizing gradient update $\mathbf{g}_u$ with X_g elements, we denote the sparsification function as $\mathcal{S}_r(\cdot) : \mathbb{R}^{X_g} \to \mathbb{R}^{X_g}$, where $r \in (0, 1]$. This function keeps the value of rX_g elements in the vector with the highest absolute values, and set the value of all other elements to 0. We denote the quantization function as $\mathcal{Q}_q(\cdot) : \mathbb{R}^{X_g} \to \mathbb{R}^{X_g}$ where $q \in \mathbb{Z}^+$, and this function quantizes each element in the vector to q bits within the max/min range. Furthermore, we denote the error memory vector for client u at iteration t as $\mathbf{m}_{u,t} \sim \mathbb{R}^{X_g}$. Therefore, the process of quantization and sparsification with error feedback at iteration t is described as follows:

$$\mathbf{g}'_u = \mathcal{Q}_q(\mathcal{S}_r(\mathbf{g}_u + \mathbf{m}_{u,t})), \tag{20}$$

$$\mathbf{m}_{u,t+1} = \mathbf{g}_u + \mathbf{m}_{u,t} - \mathbf{g}'_u, \tag{21}$$

where $\mathbf{g}'_u$ is uploaded to the PS, and the local error memory vector is updated to $\mathbf{m}_{u,t+1}$. For initialization, $\mathbf{m}_{u,1} = \mathbf{0}$.

Similarly, for FbFTL utilizing extracted feature $\mathbf{z}_{u,k}$ with X_z elements, we may also apply sparsification and quantization. However, error feedback is not applicable, because there is only one upload iteration in FbFTL, and the extracted feature is not additive unlike the gradient. Therefore, the process of quantization and sparsification for FbFTL is described as follows:

$$\mathbf{z}'_{u,k} = \mathcal{Q}_q(\mathcal{S}_r(\mathbf{z}_{u,k})), \tag{22}$$

where $\mathbf{z}'_{u,k}$ is uploaded to the PS.

In both cases, the compression rate is close to $2^{d/q}/r$ where d is the number of bits for the representation of the original data type. We note that there is also a potential drawback in the practical deployment of sparsification and error feedback. On the one hand, while all other operations including training, communication, inference, quantization and error feedback have time complexity no more than $O(X')$ where X' is the total number of parameters in the neural network, we notice that depending on the sparsification ratio r, sparsification requires up to $O(X_g \log X_g)$ time complexity for gradient-based frameworks and $O(X_z \log X_z)$ time complexity for FbFTL. Typically, we have $O(X_z) \ll O(X_g) = O(X')$, and FbFTL can achieve more significant time complexity reduction in local sparsification computation compared to gradient-based frameworks. On the other hand, by compensating for the compression error in future iterations, the error feedback significantly improves the training performance when the compression rate is high, i.e., r and q are low. However, the error feedback requires $O(X^g)$ additional memory throughout the entire training process and not just during the local training. Also, we demonstrate below in the experimental results that the gradient-based frameworks do not prevail even with error feedback.

D. Experimental Results

In this section, we numerically demonstrate the aforementioned robustness metrics in our experiments with the VGG-16 CNN model on transfer learning from ImageNet dataset to CIFAR-10 dataset.

In Fig. 6, we show the PLR (without retransmissions) of different learning frameworks when the block size equals the batch size of FbFTL (i.e., the BLR equals the PLR of FbFTL). As we have shown in the row of *uplink payload per batch* of Table III, the batch sizes of other learning frameworks are about 10^4 times larger than that of FbFTL. Due to the huge difference in the batch sizes, the PLR curves of the other learning frameworks almost reach 1 when the PLR of FbFTL is less than 0.001 (or equivalently when BLR< 0.001). The difference is still substantially high if a few rounds of retransmissions are allowed in gradient-based frameworks. Fig. 7 provides the magnified plot of Fig. 6 for BLR< 0.001, and we note that the curve of FL and the curve of FTL that updates full model overlap since they have the same number of parameters to update and therefore have the same batch size. In this figure, we again observe that the PLR curves of learning mechanisms other than FbFTL quickly approach 1 within the considered range of BLR < 0.001, while the PLR of FbFTL (being equal to BLR) stays below 0.001.

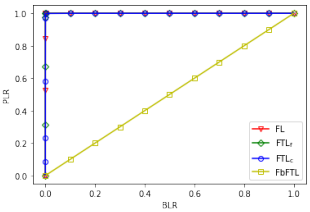


Figure 6: Comparison of PLR (without retransmission) among FL, FTL updating full model, FTL updating task-specific sub-model, and FbFTL.

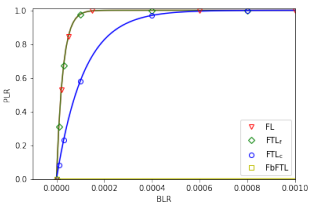


Figure 7: Magnification of Fig. 6 for BLR$<0.1\%$.

Since FbFTL has significantly lower PLR, we subsequently analyze its performance with different amount of data. In Fig. 8, we see that FbFTL has relatively high validation accuracy even with only 10% of the samples. Indeed, when we have only 0.1% of the samples (i.e., the case with 50 samples), the accuracy is 82.5% even with PLR$= 0.5$. Furthermore, we observe that as PLR increases, the accuracy curves remain relatively flat, and experience sharp drops only when PLR approaches 1. Therefore, FbFTL is considerably robust against data insufficiency and PLR. We note that such robustness requires significantly more training iterations I' such that IU and $I'U'$ have the same order of magnitude, where I is the original number of training iterations with all U participating clients, and I' is the number of training iterations with limited data from U' clients. In the case of smaller number of participating clients $U' \ll U$ and PLR $= 0$, FbFTL requires the same level of computational power consumption and more total downlink payload compared to the original case with U clients. However, there is a huge reduction in total uplink payload and total local computational power

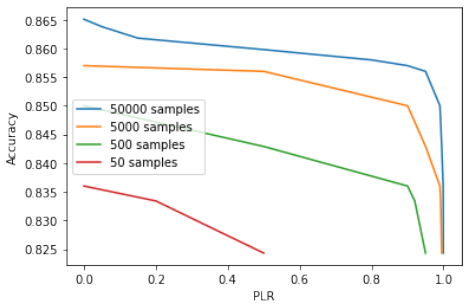


Figure 8: Validation accuracy of FbFTL with different number of participating samples and PLRs.

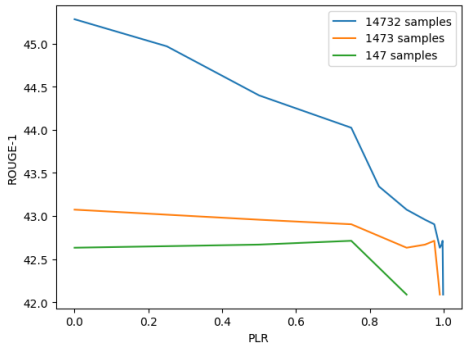


Figure 9: ROUGE-1 score of FbFTL with different number of participating samples and PLRs on FLAN-T5-small with SAMSum.

consumption, because FbFTL only requires uploading once for each participating client. In comparison, gradient-based frameworks require uploading in every training iteration, and therefore there is no such benefit in terms of reduced uplink payload and local computations, while they also suffer from higher downlink payload. Similarly, we have the performance curves for FLAN-T5-small in Fig. 9.

In Table V, we show the validation accuracy of each framework with data from all U clients and PLR $= 0$, but for different values of quantization size q bits and sparsification ratio r. In the experiment, we pick the same set of values for q and r as in [77], [78]. Note that the highest reduction in uplink payload is achieved when we set $r = 0.001$ and $q = 2$. However, even in this case, the uplink payload of gradient-based frameworks (i.e., FTL, FTL$_f$, FTL$_c$) is still greater than that of FbFTL with $r = 1$ and $q = 32$. Moreover, with such drastic reduction, gradient-based schemes achieve lower accuracies compared to that of FbFTL with $r = 1$ and $q = 32$. Therefore, even without sparsification and more restrictive quantization (e.g., $q = 8$ or $q = 2$), FbFTL outperforms gradient-based frameworks in terms of both accuracy and payload reduction. We can further reduce the uplink payload of FbFTL by choosing $r < 1$ and $q < 32$. We note that FbFTL in the extreme case of $r = 0.001$ only keeps 4 elements in the extracted features to distinguish among 10 classes. In this setting, the information is severely limited and is insufficient to make an accurate prediction, resulting in accuracy levels of 65% for FbFTL. If, on the other hand, we pick a certain threshold on the validation accuracy, for

FL FTL$_f$ FTL$_c$ FbFTL	$r = 1$	$r = 0.1$	$r = 0.01$	$r = 0.001$
$q = 32$	89.42% 93.75% 86.51% 86.51%	83.96% 93.46% 85.45% 86.47%	59.97% 90.98% 84.49% 82.10%	42.26% 81.98% 82.78% 65.71%
$q = 8$	88.46% 93.36% 86.16% 86.41%	82.34% 93.34% 85.44% 86.39%	57.04% 90.83% 84.41% 81.85%	39.34% 81.91% 82.53% 65.55%
$q = 2$	45.96% 88.20% 81.44% 85.50%	44.67% 88.16% 81.29% 85.48%	33.63% 86.19% 80.91% 80.99%	31.91% 80.15% 78.61% 65.18%

Table V: Validation accuracy comparison between FL, FTL updating full model, FTL updating task-specific sub-model and FbFTL with quantization to q bits, sparsification ratio r of elements, and error feedback (except for FbFTL).

instance 82%, FbFTL requires the sparsification ratio to be no lower than $r = 0.01$ (which is 10 times more than that of the best gradient-based framework), but still maintains a reduction of more than four orders of magnitude (i.e., reduction of 10^{-4}) in total uplink payload compared to the gradient-based algorithms while achieving the same validation accuracy. Such good performance of FbFTL without error feedback is due to the robustness of extracted features against noise. If we consider the error from compression as random noise, the extracted features from a well-trained source model is typically robust to noise while gradient update does not necessarily have the same level of robustness.

V. Privacy Analysis

In previous sections, we have described the FbFTL framework where each client u with K_u samples uploads extracted features $\mathbf{z}_{u,k}$ and output $\mathbf{y}_{u,k}$ for $k = 1, \ldots, K_u$ instead of the gradient update $\mathbf{g}_u = \sum_{k=1}^{K_u} \nabla_{\boldsymbol{\theta}} L(f_{\boldsymbol{\theta}}|\mathbf{s}_{u,k})$ as done in FL and FTL with FedAvg. In this section, we conduct a privacy analysis by studying privacy leakage to a potential adversary, and propose protection strategies. In particular, we consider leakage due to the unveiling of information regarding the outputs/labels $\{\mathbf{y}_{u,k}\}$ (henceforth referred to as label privacy leakage) and the unveiling of intermediate features $\{\mathbf{z}_{u,k}\}$ (henceforth referred to as feature privacy leakage). For example, during the training process of a classifier to distinguish the character in a photo to be a dog or a cat, the identity of the character being labeled as a dog is considered as label privacy, while the feature privacy includes additional information of the certain photo besides its label privacy such as the color of the fur and the furniture in the background.

The label privacy leakage quantifies how much information about the batch of a targeted client is revealed to an adversary through the information on the outputs $\{\mathbf{y}_{u,k}\}$ (or type of label in classification problems). The information on the outputs/labels can be of statistical nature or can be obtained

via the unveiling of the outputs to the adversary. The former considers the information leakage via the knowledge of the general sample output distribution, while the latter specifies the leakage when the adversary has access to the output/label values. We analyze the label privacy leakage via the entropy and mutual information from the adversary's perspective and propose an uploading design that randomly shuffles all batches to conceal the dependency between the client address and the output data.

Feature privacy leakage describes the amount of information that possibly leaks when the adversary obtains uploaded contents from clients. We will analyze the feature privacy leakage via the differential privacy (DP) framework [7] and provide comparisons between FbFTL and the gradient-based frameworks (FL and FTL, in which cases feature privacy is leaked when the adversary obtains the gradient updates) through experiments and numerical results.

A. Label Privacy Leakage

First, we analyze the label privacy leakage. While FbFTL directly uploads the output in each batch for each sample, we note that FL and FTL with FedAvg that update the final layer also leak the output, and hence label privacy leakage also occurs in these cases. According to [80], the count of each output type in a batch can be numerically solved given the average gradient. Adversaries with certain prior knowledge on the training data are able to gain further knowledge and reconstruct the input from the average gradient, such as deep leakage [81] or gradient inversion [82], [83]. Although the following label privacy analysis applies to both feature-based FbFTL and gradient-based FedAvg frameworks, we in label privacy analysis use the word "batch" to indicate all uploaded information from one client with multiple samples. Compared to the high-dimensional gradient $\mathbf{g}_u$ or feature $\mathbf{z}_{u,k}$, the output $\mathbf{y}$ also potentially reveals clients' private information but up to a certain degree. In this setting, we analyze the conditions under which the label privacy leakage (with statistical information) vanishes. We also address the role of shuffling the output information from all clients as a way of hiding the client's address from uploaded content when adversary has access to outputs.

To validate these approaches, we analyze the private output information leakage of a specific batch to a potential adversary without client addresses. According to [84], the privacy loss of a query can be evaluated as the difference in the privacy before and after the query. In our setting, we determine the amount of label privacy via the uncertainty from the adversary's perspective, and quantify it by utilizing the entropy formulation.

As shown in Fig. 10, we consider a common learning task in which the output $\mathbf{y} \in [0,1]^N$ is an axis-aligned unit vector with one element having a value of 1 indicating the label associated with the given input, and all others equal to 0. We assume that each sample $\{\mathbf{x}_{u,k}, \mathbf{z}_{u,k}, \mathbf{y}_{u,k}\}$ is independent and identically distributed (i.i.d.). Each client $u \in \mathcal{U}$ transmits one batch with $K_u = K$ samples, including the outputs/labels $\{\mathbf{y}_{u,k}\}_{k=1}^K$. Therefore for this client, there are N^K different possible ordered batches of outputs in total (where N is the number of possible different labels that can be associated with the input).

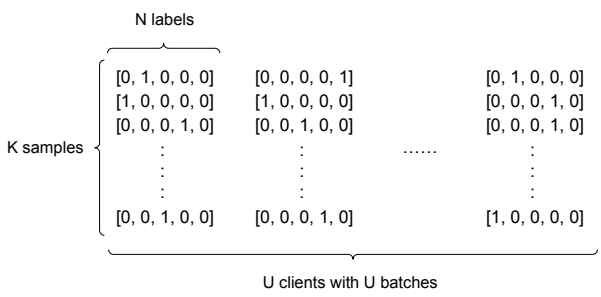


Figure 10: Diagram of every sample label $\mathbf{y}_{u,k} \in [0,1]^N$ in batches.

Assuming that the order in the batch does not contain information, we use $\mathcal{B}$ to denote the set of possible batches without order from client u, and the privacy information of the real batch b from client u is the sum vector $\sum_{k=1}^K \mathbf{y}_{u,k} = [n_{1,b}^y, n_{2,b}^y, \ldots, n_{N,b}^y]$, where $\sum_{a=1}^N n_{a,b}^y = K$. Subsequently, the number of possible batches without order is the combination with the replacement of N items taken K times and is given by

$$|\mathcal{B}| = \binom{N+K-1}{K}. \tag{23}$$

We denote the index of client u's batch as a random variable B, and the index of the uploaded batch as $B = b \in \mathcal{B}$. We denote the type of the label $\mathbf{y}_{u,k}$'s distribution as P_y. Typically to achieve better performance, in most of the machine learning applications we desire a uniform distribution over different labels, and we denote the uniform distribution over N labels as $P_{y,\text{uni}} = [\frac{1}{N}, \frac{1}{N}, \ldots, \frac{1}{N}]$.

To evaluate the adversary's knowledge on one given batch $B = b$ of a target victim client, we consider the presumed distribution $\mathbf{P}_B$ of the batch B from the adversary's perspective to quantify the adversary's uncertainty through the entropy $H(B)$ of the given batch.

1) Label Privacy Leakage with Statistical Information: A weak adversary without any prior knowledge of the sample distribution may presume the sample distribution to be uniform, i.e., $P_{y,\text{uni}}$. In this case, the distribution of ordered batches from the adversary's perspective is also uniform with probability $1/N^K$. Each batch index $b \in \mathcal{B}$ without order corresponds to $K!/\prod_{a=1}^N n_{a,b}^y!$ different batch indices with order, and hence the adversary's presumed probability distribution of batches without order (under the assumption of uniform sample distribution) is $\mathbf{P}_{B|\text{uni}} = [p_{0,1}, p_{0,2}, \ldots, p_{0,|\mathcal{B}|}]$, where

$$p_{0,b} = \frac{K!}{N^K \prod_{a=1}^N n_{a,b}^y!}. \tag{24}$$

Therefore from the weak adversary's perspective, the uncertainty of the batch from client u with known format and size but without prior knowledge about the content can be quantified by the entropy

$$H(B \mid P_{y,\text{uni}}) = -\sum_{b=1}^{|\mathcal{B}|} p_{0,b} \log_2 p_{0,b} \tag{25}$$

where the condition in the entropy signifies that this is the entropy under the assumption of uniformly distributed samples.

On the contrary, a stronger adversary with the ability to steal and decode a large amount of uploaded data may learn the structure of the DNN and is able to decode the labels of the K samples in a batch, and hence such an adversary is likely to have the prior knowledge of the general distribution of batches from a large number of clients. In the case that the true sample distribution $P_{y,\text{true}}$ is known, we denote the strong adversary's presumed distribution of the batch from client u as

$$\mathbf{P}_{B|\text{true}} = [p_1, p_2, \ldots, p_{|\mathcal{B}|}], \tag{26}$$

and the corresponding uncertainty at the adversary in the original setting before query is

$$H(B \mid P_{y,\text{true}}) = -\sum_{b=1}^{|\mathcal{B}|} p_b \log_2 p_b. \tag{27}$$

By comparing the uncertainty between the weak adversary and the strong adversary, we give the definition of label privacy leakage with statistical information.

Definition 1. *Label Privacy Leakage with Statistical Information:* For an adversary without prior knowledge of the sample distribution (and hence that initially assumes uniform sample distribution), the label privacy leakage is the amount of information regarding the target batch B that leaks to the adversary when it obtains the true distribution $P_{y,\text{true}}$. This privacy leakage can be formulated as follows:

$$L_s^l = H(B \mid P_{y,\text{uni}}) - H(B \mid P_{y,\text{true}}). \tag{28}$$

Note that $H(B \mid P_{y,\text{uni}})$ is the uncertainty in B under the assumption that the labels are uniformly distributed. $H(B \mid P_{y,\text{true}})$ is the remaining uncertainty in B when the true sample distribution is learned by the adversary. Hence, the difference is the information gained by or equivalently leaked to the adversary.

Machine learning tasks typically require dataset balancing. If the data collection process is sufficiently well designed so that each output type is almost equally likely and we have $\mathbf{P}_{B|\text{true}} \to \mathbf{P}_{B|\text{uni}}$, then the label privacy leakage $L_s^l \to 0$. As we have illustrated in section III-A, achieving this goal is more viable in FbFTL.

2) Label Privacy Leakage with Access via Query: Next, we analyze the label privacy leakage when the adversary has access to the shuffled outputs (e.g., via a query). In the worst case, the strongest query is the process that the adversary acquires all batches without clients' addresses. Specifically, we assume that the adversary has access to randomly shuffled U batches. Recall that there are $|\mathcal{B}|$ different types of batches. We use $c(b)$ to denote the count for type b batches among given U batches. With this definition, we have $\sum_{b=1}^{|\mathcal{B}|} c(b) = U$. For different set of U batches, the counts will be different. We define $\mathcal{C}_U$ as a random vector of counts of different types of batches among a total of U batches. Hence, for given U batches, the realization of this vector is $\mathcal{C}_\mathcal{U} = c_U = [c(1), c(2), \ldots, c(|\mathcal{B}|)]$. In the absence of any other statistical information, the adversary can utilize the following empirical distribution of B based on the frequency of each batch type among all U batches:

$$\mathbf{P}_{B,\text{emp}} \triangleq \mathbf{P}\{B \mid \mathcal{C}_\mathcal{U} = c_U\} = \left[\frac{c(1)}{U}, \frac{c(2)}{U}, \ldots, \frac{c(|\mathcal{B}|)}{U}\right]. \tag{29}$$

With this empirical distribution based on the shuffled U batches, the adversary's uncertainty is given by the entropy

$$H(B \mid \mathcal{C}_\mathcal{U} = c_U) = -\sum_{b=1}^{|\mathcal{B}|} \frac{c(b)}{U} \log_2 \frac{c(b)}{U}. \tag{30}$$

In the case in which the adversary has acquired U batches via the query and knows the distribution $P_{y,\text{true}}$ of the samples, we denote the adversary's presumed distribution as $\mathbf{P}(B \mid \mathcal{C}_\mathcal{U} = c_U, P_{y,\text{true}})$, and its uncertainty on client u's batch as $H(B \mid \mathcal{C}_\mathcal{U} = c_U, P_{y,\text{true}})$.

First, we establish the following result.

Lemma 1. Assume that the true sample distribution is known. Once U batches are revealed to the adversary, the distribution of the batches (from the adversary's perspective) depends only on the frequency/count of each batch type in the unveiled U batches and not on the sample distribution, i.e.,

$$P(B = b \mid \mathcal{C}_\mathcal{U} = c_U, P_{y,\text{true}}) = P\{B = b \,|\, \mathcal{C}_\mathcal{U} = c_U\}. \tag{31}$$

Proof. We prove this lemma by utilizing the Bayes' rule and determining the ratio of two conditional probabilities as follows:

$$\begin{aligned} P(B = b \mid \mathcal{C}_\mathcal{U} = c_U, P_{y,\text{true}}) =& \frac{P(\mathcal{C}_\mathcal{U} = c_U, B = b | P_{y,\text{true}})}{P(\mathcal{C}_\mathcal{U} = c_U | P_{y,\text{true}})} \\ =& \frac{\frac{(U-1)!}{(c(b)-1)! \prod_{b' \neq b} c(b')!}}{\frac{U!}{c(b)! \prod_{b' \neq b} c(b')!}} \\ =& \frac{\frac{(U-1)!}{(c(b)-1)!}}{\frac{U!}{c(b)!}} \\ =& \frac{c(b)}{U} \\ =& P\{B = b \,|\, \mathcal{C}_\mathcal{U} = c_U\}. \end{aligned} \tag{32}$$

□

This result indicates that the batch distribution is equal to the empirical distribution in (29). Consequently, we also have the following characterization for the entropies:

$$H(B \mid \mathcal{C}_\mathcal{U} = c_U, P_{y,\text{true}}) = H(B \mid \mathcal{C}_\mathcal{U} = c_U) \tag{33}$$

Next, we give the definition of the label privacy leakage with query, and identify a condition under which the privacy leakage vanishes.

Definition 2. *Label Privacy Leakage with Query:* For an adversary with prior knowledge of the true sample distribution $P_{y,\text{true}}$, the label privacy leakage with query is the amount of information regarding the batch of a target client that leaks to the adversary when it obtains the randomly shuffled set of uploaded U batches including the target's batch. This privacy

leakage can be formulated as follows:

$$\begin{aligned} L_q^l &= H(B \mid P_{y,\text{true}}) - H(B \mid P_{y,\text{true}}, \mathcal{C}_\mathcal{U} = c_U) \\ &= H(B \mid P_{y,\text{true}}) - H(B \mid \mathcal{C}_\mathcal{U} = c_U). \end{aligned} \tag{34}$$

Lemma 2. As the number of shuffled batches goes to infinity, the label privacy leakage with query converges to 0, i.e.,

$$\lim_{U\to\infty} L_q^l = 0. \tag{35}$$

Proof. As U grows without bound, we have $\lim_{U\to\infty} \mathbf{P}_{B,\text{emp}} = \mathbf{P}_{B|\text{true}}$ by the law of large numbers, and as a result, we have the characterization that $\lim_{U\to\infty} H(B \mid \mathcal{C}_\mathcal{U} = c_U) = H(B \mid P_{y,\text{true}})$. Hence, the label privacy leakage L_l^q with query converges to 0. □

Note that if the adversary does not initially know even the distribution of the samples and assume a uniformly distributed samples, then total label privacy leakage after the unveiling of the U batches to the adversary can be defined as

$$L_t^l = L_s^l + L_q^l = H(B \mid P_{y,\text{uni}}) - H(B \mid \mathcal{C}_\mathcal{U} = c_U). \tag{36}$$

If the shuffled dataset is perfectly balanced and we have $\mathbf{P}_{B,\text{emp}} = \mathbf{P}_{B|\text{uni}}$, the total label privacy leakage is zero, i.e., $L_t^l = L_s^l + L_q^l = 0$.

B. Feature Privacy Leakage

In this section, we analyze the privacy leakage on the input $\mathbf{x}$ via uploaded content. Such privacy leakage includes the information that is not necessarily needed to determine the output $\mathbf{y}$, and we will define it as feature privacy. Different from label privacy where the adversary knows the implication of each type of output (or the meaning of each label), it is hard to define the adversary's prior knowledge on the gradients and extracted features, and it is unlikely to cancel feature privacy leakage via shuffling. Furthermore, it is hard to quantify and analyze the entropy and mutual information of gradients and extracted features because of the complexity of neural networks. Instead, we will analyze the feature privacy leakage via differential privacy.

For FbFTL, the intermediate output $\mathbf{z}$ is also referred to as the smashed data in split learning [85], [86], and cannot be directly transformed back to the input $\mathbf{x}$ due to the nonlinearity of the activation functions in each layer. However, it is possible that FbFTL leaks privacy to some extent and partially reveals the input. Since the feature before a fully connected layer from a sample can be analytically solved from its gradient [87], it is possible that gradient-based frameworks also leak the input. The strategy to extract the input from FedAvg gradients includes deep leakage [81], [87], [88] and gradient inversion [82], [83], and is extensively studied for image recognition. However, deep leakage highly depends on the dataset and DNN structure. Therefore, an analysis of the privacy leakage beyond label privacy is needed to identify what may be partially revealed regarding the input $\mathbf{x}$ from the uploaded features.

Differential privacy (DP) provides an upper bound on the privacy leakage using a different measuring approach, and we compare the feature privacy preserving performances of FbFTL and other schemes via DP. In the DP analysis of feature privacy, we denote the training samples for each batch as $\mathbf{d}$ and the function that generates the upload vector as g. More specifically, $\mathbf{d} = \mathbf{s}_{u,k}$ and $g(\mathbf{d}) = \mathbf{z}_{u,k} = f^1_{\boldsymbol{\theta}^1}(\mathbf{x}_{u,k})$ for FbFTL, while $\mathbf{d} = \bigcup_{k=1}^K \mathbf{s}_{u,k}$ and $g(\mathbf{d}) = \mathbf{g}_u = \sum_{k=1}^K \nabla_{\boldsymbol{\theta}} L(f_{\boldsymbol{\theta}}|\mathbf{s}_{u,k})$ for gradient-based FedAvg frameworks. Similarly as in [89]–[91], we define the feature privacy through the following condition: g satisfies (ϵ, δ)-DP if for any subset of possible outputs G it holds that

$$\Pr\{g(\mathbf{d}) \in G\} \le e^\epsilon \Pr\{g(\mathbf{d}') \in G\} + \delta, \tag{37}$$

where $\mathbf{d}$ and $\mathbf{d}'$ differ in a single sample with the true distribution $P_{y,\text{true}}$.

In [92], it has been shown via moments accountant approach that adding Gaussian noise $\mathbf{n}$ to $g(\mathbf{d})$ prior to transmission maintains an overall privacy loss of (ϵ, δ). Typically, the variance of the noise depends on the maximum distance $\max \|g(\mathbf{d}) - g(\mathbf{d}')\|$ for any two adjacent inputs $\mathbf{d}$ and $\mathbf{d}'$. To provide a fair comparison between different learning frameworks, we consider the maximum relative distance $R = \max_{d,d'} \frac{\max|g(\mathbf{d})-g(\mathbf{d}')|}{K\text{std}(g(\mathbf{d}))}$ within training set for each framework, where $|\cdot|$ denotes absolute value and std$(\cdot)$ denotes the standard deviation. Thus, the additive Gaussian noise to each output $g(\mathbf{d}_1)$ should be $\mathbf{n} \sim \mathcal{N}(0, \sigma^2 R^2 \text{std}^2(g(\mathbf{d}_1))\mathbf{I})$ to mitigate the feature privacy leakage, and the lower bound of the σ value can be determined via the moments accountant approach.

As a result, according to [92, Theorem 1], there exist constants c_1 and c_2 such that for any $\epsilon < c_1 C^2 I$, the training process with $g(\mathbf{d}) + \mathbf{n}$ is (ϵ, δ)-differentially private for any $\delta > 0$ if we choose $\sigma \ge \frac{c_2 C}{\epsilon}\sqrt{-I \log \delta}$, where I is the number of iterations ($I = 1$ for FbFTL), and C is the fraction of clients selected in each iteration. When the value of δ and noise factor σR are fixed, ϵ for FedAvg DP decreases as the batch size K grows, but increases as training iterations I (and potentially, the number of retraining for hyper-parameter tuning) increase. However, ϵ for FbFTL DP does not depend on these factors. Furthermore, the final performance also depends on the robustness of each framework against noise. Therefore, we demonstrate the comparisons via experiments in the following subsection.

C. Experimental Results

In our experiments, we utilize the dataset CIFAR-10 which has $N = 10$ types of labels, and is well-balanced (i.e., $P_{y,\text{true}} = P_{y,\text{uni}}$). We generate shuffled batches according to the uniform sample distribution $P_{y,\text{uni}}$ and evaluate the total label privacy leakage in different scenarios in Fig. 11. The blue curve plots the initial uncertainty $H(B \mid P_{y,\text{uni}})$ for different values of K (where K is the number of samples from each client). Each of the other curves shows the uncertainty given shuffled data $H(B \mid \mathcal{C}_U = c_U)$ with fixed total amount of samples UK from all clients, and the number of clients U is determined by each value of K accordingly. By Definition 1 and Definition 2, the total label privacy leakage given shuffled data is $L_t^l = L_s^l + L_q^l$ and is equal to the difference $H(B \mid P_{y,\text{uni}}) - H(B \mid \mathcal{C}_U = c_U)$. We see that $H(B \mid P_{y,\text{uni}}) \approx H(B \mid \mathcal{C}_U = c_U)$ when K is small, and thus the label privacy is well preserved. However, for given value of the product UK, $H(B \mid \mathcal{C}_U = c_U)$ diminishes fast once a

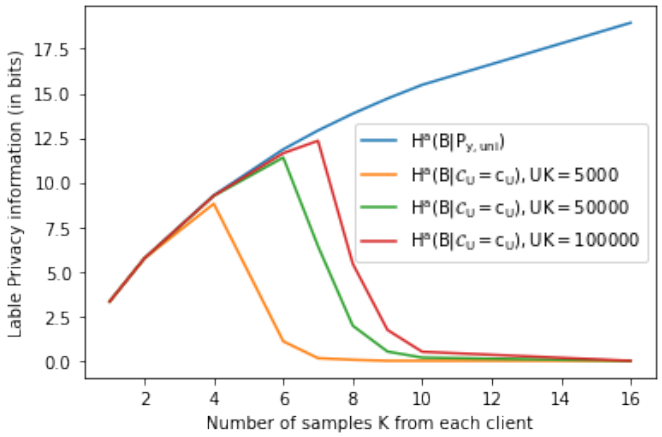
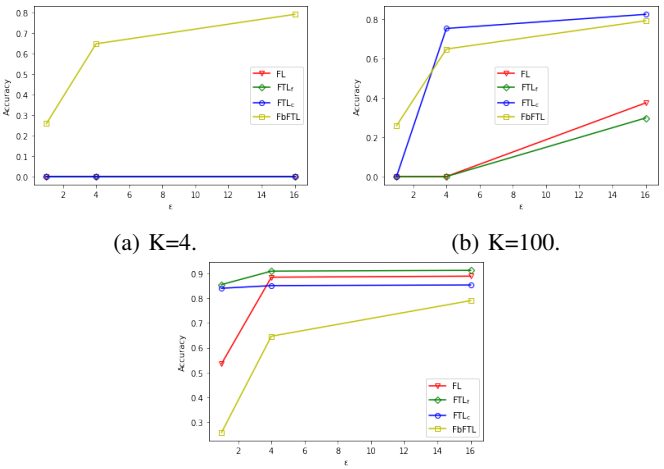


Figure 11: Label privacy of each client in bits against the number of samples K from each client.

certain threshold of K is exceeded. Therefore, we note that the label privacy leakage becomes high when K is large. As addressed before, the label privacy applies in the same way to both FbFTL and gradient-based FedAvg frameworks, and hence a smaller K might be preferred in order to preserve the label privacy. However, current FL privacy analyses typically focus on DP that provides a relatively loose upper bound on the total local privacy. Within the DP setting, studies typically consider a small number of clients and assume that each client obtains hundreds of samples, if not thousands of, to achieve improved DP performance. However, as we have observed above, a setting with higher values of K can lead to higher label privacy loss (in addition to requiring more training epochs and achieving less validation accuracy). This leads to the important conclusion that the balance between different types of privacy leakage needs to be considered carefully.

To compare the DP performances of feature privacy leakage, we apply different noise levels σ on upload content $g(\mathbf{d})$ of different frameworks to achieve the same level (ϵ, δ) of privacy, and compare the validation accuracy at convergence with different K in Fig. 12. In our experiments, we implement the moments accountant via Rényi Divergence-based DP accountant to get a tighter bound. With fixed $\delta = 10^{-6}$, each plot shows the validation accuracy at convergence as a function of ϵ for given K. We note that higher ϵ indicates weaker DP protection, lower noise level σ, and thus higher accuracy (closer to the original performance without noise). As shown in Fig. 11, label privacy is better preserved at $K = 4$, and we show the DP performance within the same setting in Fig. 12a. While gradient-based FedAvg frameworks totally fail, FbFTL has good performance because it has much smaller data size $|g(\mathbf{d})|$ and is more robust against noise, as it receives constant data instead of varying perturbation in each training iteration and converges to a sub-optimal model. Such advantage of FbFTL remains until K increases to 100 as shown in Fig. 12b, where FTL that updates the task-specific sub-model outperforms FbFTL at $\epsilon \geq 4$ as they have the same data size $|g(\mathbf{d})|$ while the noise level of FedAvg decreases as K increases. However, the label privacy completely leaks for $K \geq 16$ as we have seen in Fig. 11. FedAvg for $K \geq 16$ only prevails in protecting the subset of feature privacy that is not mitigated via shuffling. In Fig. 12c, we show the comparison for even larger $K = 600$, where all types of FedAvg perform better than FbFTL.



(a) K=4. (b) K=100.

(c) K=600.

Figure 12: Comparison of validation accuracy at different K, ϵ, and $\delta = 10^{-6}$ between FL, FTL updating full model, FTL updating task-specific sub-model and FbFTL. The accuracy is marked as 0 if the model fails to converge.

Note that it is extremely difficult to analytically evaluate the volume of feature privacy leakage that can be mitigated via shuffling. Howeover, we can compare the performances of FbFTL and FedAvg for a given K via experimental results as done above. One key conclusion we have is the following. In terms of privacy preservation, FbFTL is preferred when K is small (i.e., each client obtains a small set of samples), and gradient-based FedAvg is preferred when K is large.

VI. Conclusion

In this paper, we have presented a novel communication-efficient federated transfer learning method. In this proposed feature-based federated transfer learning (FbFTL), the features and outputs are uploaded rather than the gradient updates. We have provided a thorough description of the system design and the learning algorithm, and compared its theoretical payload with that of federated learning and federated transfer learning. Our results demonstrate substantial reductions in both uplink and downlink payload when using FbFTL. Via experiments, we have further shown the effectiveness of the proposed FbFTL by showing that FbFTL reduces the uplink payload by up to five orders of magnitude compared to that of existing methods. Subsequently, we have demonstrated that FbFTL with small batch size has significantly less packet loss rate than gradient-based frameworks, and illustrated its robustness against data insufficiency and quantization. Finally, we have considered different types of privacy leakage, and analyzed mitigation approaches. Specifically, we have first analyzed label privacy leakage with both statistical knowledge and query (resulting in access to the label outputs). We have identified a condition under which label privacy vanishes. We have also addressed feature privacy and considered a DP mechanism for preserving this privacy. We have shown

that with small K and shuffling that eliminates label privacy leakage, FbFTL also attains good differential feature privacy protection. These characterizations and results render FbFTL a communication-efficient, robust, and privacy-preserving novel federated transfer learning scheme.

REFERENCES

[1] J. Konečnỳ, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," *arXiv preprint arXiv:1610.02527*, 2016.

[2] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, "Advances and open problems in federated learning," *arXiv preprint arXiv:1912.04977*, 2019.

[3] J. Mills, J. Hu, and G. Min, "Communication-efficient federated learning for wireless edge intelligence in iot," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 5986–5994, 2019.

[4] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[5] Y. Cheng, J. Lu, D. Niyato, B. Lyu, J. Kang, and S. Zhu, "Federated transfer learning with client selection for intrusion detection in mobile edge computing," *IEEE Communications Letters*, vol. 26, no. 3, pp. 552–556, 2022.

[6] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, "Federated learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 13, no. 3, pp. 1–207, 2019.

[7] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. Quek, and H. V. Poor, "Federated learning with differential privacy: Algorithms and performance analysis," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3454–3469, 2020.

[8] X. Yin, X. Yu, K. Sohn, X. Liu, and M. Chandraker, "Feature transfer learning for face recognition with under-represented data," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 5704–5713.

[9] Y. Chen, X. Qin, J. Wang, C. Yu, and W. Gao, "Fedhealth: A federated transfer learning framework for wearable healthcare," *IEEE Intelligent Systems*, vol. 35, no. 4, pp. 83–93, 2020.

[10] C. Ju, D. Gao, R. Mane, B. Tan, Y. Liu, and C. Guan, "Federated transfer learning for eeg signal classification," in *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*. IEEE, 2020, pp. 3040–3045.

[11] H. Yang, H. He, W. Zhang, and X. Cao, "Fedsteg: A federated transfer learning framework for secure image steganalysis," *IEEE Transactions on Network Science and Engineering*, 2020.

[12] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[13] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big data*, vol. 3, no. 1, pp. 1–40, 2016.

[14] C. Wang and S. Mahadevan, "Heterogeneous domain adaptation using manifold alignment," in *Twenty-second international joint conference on artificial intelligence*, 2011.

[15] M. Kaya, G. Fidan, and I. H. Toroslu, "Transfer learning using twitter data for improving sentiment classification of turkish political news," in *Information sciences and systems 2013*. Springer, 2013, pp. 139–148.

[16] F. H. Khan, U. Qamar, and S. Bashir, "Enhanced cross-domain sentiment classification utilizing a multi-source transfer learning approach," *Soft Computing*, vol. 23, no. 14, pp. 5431–5442, 2019.

[17] L. Duan, D. Xu, and I. Tsang, "Learning with augmented features for heterogeneous domain adaptation," *arXiv preprint arXiv:1206.4660*, 2012.

[18] B. Kulis, K. Saenko, and T. Darrell, "What you saw is not what you get: Domain adaptation using asymmetric kernel transforms," in *CVPR 2011*. IEEE, 2011, pp. 1785–1792.

[19] Y. Zhu, Y. Chen, Z. Lu, S. J. Pan, G.-R. Xue, Y. Yu, and Q. Yang, "Heterogeneous transfer learning for image classification," in *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[20] M. Shaha and M. Pawar, "Transfer learning for image classification," in *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*. IEEE, 2018, pp. 656–660.

[21] M. Hussain, J. J. Bird, and D. R. Faria, "A study on cnn transfer learning for image classification," in *UK Workshop on computational Intelligence*. Springer, 2018, pp. 191–202.

[22] D. Han, Q. Liu, and W. Fan, "A new image classification method using cnn transfer learning and web data augmentation," *Expert Systems with Applications*, vol. 95, pp. 43–56, 2018.

[23] M. Harel and S. Mannor, "Learning from multiple outlooks," *arXiv preprint arXiv:1005.0027*, 2010.

[24] J. Park, R. J. Javier, T. Moon, and Y. Kim, "Micro-doppler based classification of human aquatic activities via transfer learning of convolutional neural networks," *Sensors*, vol. 16, no. 12, p. 1990, 2016.

[25] N. Agarwal, A. Sondhi, K. Chopra, and G. Singh, "Transfer learning: Survey and classification," in *Smart Innovations in Communication and Computational Sciences*. Springer, 2021, pp. 145–155.

[26] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Information and Software Technology*, vol. 54, no. 3, pp. 248–256, 2012.

[27] J. Nam, W. Fu, S. Kim, T. Menzies, and L. Tan, "Heterogeneous defect prediction," *IEEE Transactions on Software Engineering*, vol. 44, no. 9, pp. 874–896, 2017.

[28] P. Prettenhofer and B. Stein, "Cross-language text classification using structural correspondence learning," in *Proceedings of the 48th annual meeting of the association for computational linguistics*, 2010, pp. 1118–1127.

[29] J. T. Zhou, I. W. Tsang, S. J. Pan, and M. Tan, "Heterogeneous domain adaptation for multiple classes," in *Artificial intelligence and statistics*. PMLR, 2014, pp. 1095–1103.

[30] J. T. Zhou, S. J. Pan, I. W. Tsang, and Y. Yan, "Hybrid heterogeneous transfer learning through deep learning," in *Twenty-eighth AAAI conference on artificial intelligence*, 2014.

[31] M. Chen, D. Gündüz, K. Huang, W. Saad, M. Bennis, A. V. Feljan, and H. V. Poor, "Distributed learning in wireless networks: Recent progress and future challenges," *arXiv preprint arXiv:2104.02151*, 2021.

[32] Y. Yuan, X. Chen, X. Chen, and J. Wang, "Segmentation transformer: Object-contextual representations for semantic segmentation," in *European Conference on Computer Vision (ECCV)*, vol. 1, 2021.

[33] J. Jain, A. Singh, N. Orlov, Z. Huang, J. Li, S. Walton, and H. Shi, "Semask: Semantically masked transformers for semantic segmentation," *arXiv preprint arXiv:2112.12782*, 2021.

[34] Z. Dai, H. Liu, Q. V. Le, and M. Tan, "Coatnet: Marrying convolution and attention for all data sizes," *arXiv preprint arXiv:2106.04803*, 2021.

[35] P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur, "Sharpness-aware minimization for efficiently improving generalization," *arXiv preprint arXiv:2010.01412*, 2020.

[36] Z. Liu, H. Hu, Y. Lin, Z. Yao, Z. Xie, Y. Wei, J. Ning, Y. Cao, Z. Zhang, L. Dong *et al.*, "Swin transformer v2: Scaling up capacity and resolution," *arXiv preprint arXiv:2111.09883*, 2021.

[37] G. Ghiasi, Y. Cui, A. Srinivas, R. Qian, T.-Y. Lin, E. D. Cubuk, Q. V. Le, and B. Zoph, "Simple copy-paste is a strong data augmentation method for instance segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 2918–2928.

[38] I. Yamada, A. Asai, H. Shindo, H. Takeda, and Y. Matsumoto, "Luke: deep contextualized entity representations with entity-aware self-attention," *arXiv preprint arXiv:2010.01057*, 2020.

[39] A. Srivastava, D. Jha, S. Chanda, U. Pal, H. D. Johansen, D. Johansen, M. A. Riegler, S. Ali, and P. Halvorsen, "Msrf-net: A multi-scale residual fusion network for biomedical image segmentation," *arXiv preprint arXiv:2105.07451*, 2021.

[40] Y. Zhang, J. Qin, D. S. Park, W. Han, C.-C. Chiu, R. Pang, Q. V. Le, and Y. Wu, "Pushing the limits of semi-supervised learning for automatic speech recognition," *arXiv preprint arXiv:2010.10504*, 2020.

[41] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[42] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[43] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009, citeseer.

[44] M. Series, "Minimum requirements related to technical performance for imt-2020 radio interface (s)," *Report*, pp. 2410–0, 2017.

[45] H. Sun, S. Li, F. R. Yu, Q. Qi, J. Wang, and J. Liao, "Toward communication-efficient federated learning in the internet of things with edge computing," *IEEE Internet of Things Journal*, vol. 7, no. 11, pp. 11 053–11 067, 2020.

[46] E. Ozfatura, K. Ozfatura, and D. Gündüz, "Time-correlated sparsification for communication-efficient federated learning," in *2021 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2021, pp. 461–466.

[47] Y. He, H.-P. Wang, and M. Fritz, "Cossgd: Communicationefficient federated learning with a simple cosine-based quantization," in *1st NeurIPS Workshop on New Frontiers in Federated Learning (NFFL)*, 2021.

[48] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[49] D. Basu, D. Data, C. Karakus, and S. Diggavi, "Qsparse-local-sgd: Distributed sgd with quantization, sparsification and local computations," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[50] P. Richtárik, I. Sokolov, and I. Fatkhullin, "Ef21: A new, simpler, theoretically better, and practically faster error feedback," *Advances in Neural Information Processing Systems*, vol. 34, pp. 4384–4396, 2021.

[51] D. Li and J. Wang, "Fedmd: Heterogenous federated learning via model distillation," *arXiv preprint arXiv:1910.03581*, 2019.

[52] T. Lin, L. Kong, S. U. Stich, and M. Jaggi, "Ensemble distillation for robust model fusion in federated learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 2351–2363, 2020.

[53] J.-H. Ahn, O. Simeone, and J. Kang, "Wireless federated distillation for distributed edge learning with heterogeneous data," in *2019 IEEE 30th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*. IEEE, 2019, pp. 1–6.

[54] Y. Jiang, S. Wang, V. Valls, B. J. Ko, W.-H. Lee, K. K. Leung, and L. Tassiulas, "Model pruning enables efficient federated learning on edge devices," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[55] J. Liu, S. Tripathi, U. Kurup, and M. Shah, "Pruning algorithms to accelerate convolutional neural networks for edge applications: A survey," *arXiv preprint arXiv:2005.04275*, 2020.

[56] H. Sidahmed, Z. Xu, A. Garg, Y. Cao, and M. Chen, "Efficient and private federated learning with partially trainable networks," *arXiv preprint arXiv:2110.03450*, 2021.

[57] T.-J. Yang, D. Guliani, F. Beaufays, and G. Motta, "Partial variable training for efficient on-device federated learning," in *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2022, pp. 4348–4352.

[58] O. Aygün, M. Kazemi, D. Gündüz, and T. M. Duman, "Hierarchical over-the-air federated edge learning," in *ICC 2022-IEEE International Conference on Communications*. IEEE, 2022, pp. 3376–3381.

[59] K. Yang, T. Jiang, Y. Shi, and Z. Ding, "Federated learning via over-the-air computation," *IEEE Transactions on Wireless Communications*, vol. 19, no. 3, pp. 2022–2035, 2020.

[60] Y. Sun, S. Zhou, Z. Niu, and D. Gündüz, "Time-correlated sparsification for efficient over-the-air model aggregation in wireless federated learning," *arXiv preprint arXiv:2202.08420*, 2022.

[61] O. Shahid, S. Pouriyeh, R. M. Parizi, Q. Z. Sheng, G. Srivastava, and L. Zhao, "Communication efficiency in federated learning: Achievements and challenges," *arXiv preprint arXiv:2107.10996*, 2021.

[62] H. Xu, C.-Y. Ho, A. M. Abdelmoniem, A. Dutta, E. H. Bergou, K. Karatsenidis, M. Canini, and P. Kalnis, "Compressed communication for distributed deep learning: Survey and quantitative evaluation," Tech. Rep., 2020.

[63] A. Reisizadeh, A. Mokhtari, H. Hassani, A. Jadbabaie, and R. Pedarsani, "Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 2021–2031.

[64] N. Shlezinger, M. Chen, Y. C. Eldar, H. V. Poor, and S. Cui, "Federated learning with quantization constraints," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 8851–8855.

[65] S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang, "Domain adaptation via transfer component analysis," *IEEE Transactions on Neural Networks*, vol. 22, no. 2, pp. 199–210, 2011.

[66] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[67] C. Thapa, P. C. M. Arachchige, S. Camtepe, and L. Sun, "Splitfed: When federated learning meets split learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 8, 2022, pp. 8485–8493.

[68] A. Fallah, A. Mokhtari, and A. Ozdaglar, "Personalized federated learning: A meta-learning approach," *arXiv preprint arXiv:2002.07948*, 2020.

[69] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *arXiv preprint arXiv:1806.00582*, 2018.

[70] J. Park, S. Samarakoon, M. Bennis, and M. Debbah, "Wireless network intelligence at the edge," *Proceedings of the IEEE*, vol. 107, no. 11, pp. 2204–2239, 2019.

[71] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[72] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS 2017 Workshop on Autodiff*, 2017. [Online]. Available: https://openreview.net/forum?id=BJJsrmfCZ

[73] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[74] B. Gliwa, I. Mochol, M. Biesek, and A. Wawer, "SAMSum corpus: A human-annotated dialogue dataset for abstractive summarization," in *Proceedings of the 2nd Workshop on New Frontiers in Summarization*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 70–79. [Online]. Available: https://www.aclweb.org/anthology/D19-5409

[75] H. W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, Y. Li, X. Wang, M. Dehghani, S. Brahma *et al.*, "Scaling instruction-finetuned language models," *arXiv preprint arXiv:2210.11416*, 2022.

[76] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Huggingface's transformers: State-of-the-art natural language processing," 2020.

[77] D. Alistarh, T. Hoefler, M. Johansson, N. Konstantinov, S. Khirirat, and C. Renggli, "The convergence of sparsified gradient methods," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[78] S. U. Stich, J.-B. Cordonnier, and M. Jaggi, "Sparsified sgd with memory," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[79] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, "signsgd: Compressed optimisation for non-convex problems," in *International Conference on Machine Learning*. PMLR, 2018, pp. 560–569.

[80] A. Wainakh, F. Ventola, T. Müßig, J. Keim, C. G. Cordero, E. Zimmer, T. Grube, K. Kersting, and M. Mühlhäuser, "User-level label leakage from gradients in federated learning," *Proceedings on Privacy Enhancing Technologies*, vol. 2022, no. 2, pp. 227–244, 2022.

[81] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[82] J. Jeon, K. Lee, S. Oh, J. Ok *et al.*, "Gradient inversion with generative image prior," *Advances in Neural Information Processing Systems*, vol. 34, pp. 29 898–29 908, 2021.

[83] Y. Huang, S. Gupta, Z. Song, K. Li, and S. Arora, "Evaluating gradient inversion attacks and defenses in federated learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 7232–7241, 2021.

[84] L. Longpr, V. Kreinovich, and T. Dumrongpokaphan, "Entropy as a measure of average loss of privacy," *Thai Journal of Mathematics*, pp. 7–15, 2017.

[85] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *Journal of Network and Computer Applications*, vol. 116, pp. 1–8, 2018.

[86] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," *arXiv preprint arXiv:1812.00564*, 2018.

[87] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller, "Inverting gradients-how easy is it to break privacy in federated learning?" *Advances in Neural Information Processing Systems*, vol. 33, pp. 16 937–16 947, 2020.

[88] B. Zhao, K. R. Mopuri, and H. Bilen, "idlg: Improved deep leakage from gradients," *arXiv preprint arXiv:2001.02610*, 2020.

[89] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of cryptography conference*. Springer, 2006, pp. 265–284.

[90] C. Dwork, "A firm foundation for private data analysis," *Communications of the ACM*, vol. 54, no. 1, pp. 86–95, 2011.

[91] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy." *Found. Trends Theor. Comput. Sci.*, vol. 9, no. 3-4, pp. 211–407, 2014.

[92] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 308–318.