PPFL: Privacy-preserving Federated Learning with Trusted Execution Environments

Fan Mo*
Imperial College London

Hamed Haddadi
Imperial College London

Kleomenis Katevas
Telefónica Research

Eduard Marin
Telefónica Research

Diego Perino
Telefónica Research

Nicolas Kourtellis
Telefónica Research

ABSTRACT

We propose and implement a Privacy-preserving Federated Learning (*PPFL*) framework for mobile systems to limit privacy leakages in federated learning. Leveraging the widespread presence of Trusted Execution Environments (TEEs) in high-end and mobile devices, we utilize TEEs on clients for local training, and on servers for secure aggregation, so that model/gradient updates are hidden from adversaries. Challenged by the limited memory size of current TEEs, we leverage greedy layer-wise training to train each model's layer inside the trusted area until its convergence. The performance evaluation of our implementation shows that *PPFL* can significantly improve privacy while incurring small system overheads at the client-side. In particular, *PPFL* can successfully defend the trained model against data reconstruction, property inference, and membership inference attacks. Furthermore, it can achieve *comparable* model utility with fewer communication rounds (0.54×) and a similar amount of network traffic (1.002×) compared to the standard federated learning of a complete model. This is achieved while only introducing up to ~15% CPU time, ~18% memory usage, and ~21% energy consumption overhead in *PPFL*'s client-side.

CCS CONCEPTS

• **Security and privacy** → **Privacy protections**; *Distributed systems security*; • **Computing methodologies** → *Distributed algorithms*.

---

*Work performed while at Telefónica Research.

1 INTRODUCTION

Training deep neural networks (DNNs) on multiple devices locally and building an aggregated global model on a server, namely federated learning (FL), has drawn significant attention from academia (e.g., [17, 27, 42]) and industry, and is even being deployed in real systems (e.g., Google Keyboard [7]). Unlike traditional machine learning (ML), where a server collects all user data at a central point and trains a global model, in FL, users only send the locally updated model parameters to the server. This allows training a model without the need for users to reveal their data, thus preserving their privacy. Unfortunately, recent works have shown that adversaries can execute attacks to retrieve sensitive information from the model parameters themselves [16, 20, 45, 78]. Prominent examples of such attacks are data reconstruction [16, 20] and various types of inference attacks [20, 45]. The fundamental reason why these attacks are possible is because as a DNN learns to achieve their main task, it also learns irrelevant information from users' training data that is inadvertently embedded in the model [73]. Note that in FL scenarios, such attacks can be launched both at server and client sides.

Motivated by these attacks, researchers have recently introduced several countermeasures to prevent them. Existing solutions can be grouped into three main categories depending on whether they rely on: (i) homomorphic encryption (e.g., [2, 42]), (ii) multi-party computation (e.g., [8]), or (iii) differential privacy (e.g., [14, 17, 44]). While homomorphic encryption is practical in both high-end and mobile devices, it only supports a limited number of arithmetic operations in the encrypted domain. Alternatively, the use of fully homomorphic encryption has been employed to allow arbitrary operations in the encrypted domain, thus supports ML. Yet, this comes with too much computational overhead, making it impractical for mobile devices [51, 63]. Similarly, multi-party computation-based solutions incur significant computational overhead. Also, in some cases, differential privacy can fail to provide sufficient privacy as shown in [45]. Furthermore, it can negatively impact the utility and fairness of the model [3, 25], as well as the system performance [66, 68]. Overall, none of the existing solutions meets all requirements, hampering their adoption.

More recently, the use of hardware-based Trusted Execution Environments (TEEs) has been proposed as a promising way to preclude attacks against DNN model parameters and gradients. TEEs allow to securely store data and execute arbitrary code on an untrusted device almost at native speed through secure memory compartments. All these advantages – together with the recent commoditization of TEEs both in high-end and mobile devices – make TEEs a suitable candidate to allow fully privacy-preserving

ML modeling. However, in order to keep the Trusted Computing Base (TCB) as small as possible, current TEEs have limited memory. This makes it impossible to simultaneously place all DNN layers inside the TEE. As a result, prior work has opted for using TEEs to conceal only the most sensitive DNN layers from adversaries, leaving other layers unprotected [18, 49]. While this approach was sufficient to mitigate some attacks against traditional ML where clients obtain only the final model, in FL scenarios the attack surface is significantly larger. FL client devices are able to observe distinct snapshots of the model throughout the training, allowing them to realize attacks at different stages [20, 45]. Therefore, it is of utmost importance to protect all DNN layers using the TEE.

In this paper, we propose Privacy-preserving Federated Learning (PPFL), the first practical framework to *fully* prevent private information leakage at both *server* and *client-side* under FL scenarios. PPFL is based on greedy layer-wise training and aggregation, overcoming the constraints posed by the limited TEE memory, and providing comparable accuracy of complete model training at the price of a tolerable delay. Our layer-wise approach supports sophisticated settings such as training one or more layers (block) each time, which can potentially better deal with heterogeneous data at the client-side and speed up the training process.

To show its feasibility, we implemented and evaluated a full prototype of PPFL system including server-side (with Intel SGX), client-side (with Arm TrustZone) elements of the design, and the secure communication between them. Our experimental evaluation shows that PPFL provides full protection against data reconstruction, property inference, and membership inference attacks, whose outcomes are degraded to random guessing (e.g., white noise images or 50% precision scores). PPFL is practical as it does not add significant overhead to the training process. Compared to regular end-to-end FL, PPFL introduces a 3× or higher delay for completing the training of all DNN layers. However, PPFL achieves comparable ML performance when training only the first few layers, meaning that it is not needed to train all DNN layers. Due to this flexibility of layer-wise training, PPFL can provide a similar ML model utility as end-to-end FL, with fewer communication rounds (0.54×), and a similar amount of network traffic (1.002×), with only ~15% CPU time, ~18% memory usage, and ~21% energy consumption overhead at client-side.

2 BACKGROUND AND RELATED WORK

In this section, we provide the background needed to understand the way TEEs work (Sec. 2.1), existing privacy risks in FL (Sec. 2.2), privacy-preserving ML techniques using TEEs (Sec. 2.3), as well as core ideas behind layer-wise DNN training for FL (Sec. 2.4).

2.1 Trusted Execution Environments (TEE)

A TEE enables the creation of a secure area on the main processor that provides strong confidentiality and integrity guarantees to any data and code it stores or processes. TEEs realize strong isolation and attestation of secure compartments by enforcing a dual-world view where even compromised or malicious system (i.e., privileged) software in the normal world – also known as the Rich Operating System Execution Environment (REE) – cannot gain access to the secure world. This allows for a drastic reduction of the TCB since

only the code running in the secure world needs to be trusted. Another key aspect of TEEs is that they allow arbitrary code to run inside almost at native speed. In order to keep the TCB as small as possible, current TEEs have limited memory; beyond this, TEEs are required to swap pages between secure and unprotected memory, which incurs a significant overhead and hence must be prevented.

Over the last few years, significant research and industry efforts have been devoted to developing secure and programmable TEEs for high-end devices (e.g., servers[1]) and mobile devices (e.g., smartphones). In our work, we leverage Intel Software Guard Extensions (Intel SGX) [13] at the server-side, while in the client devices we rely on Open Portable Trusted Execution Environment (OP-TEE) [40]. OP-TEE is a widely known open-source TEE framework that is supported by different boards equipped with Arm TrustZone. While some TEEs allow the creation of fixed-sized secure memory regions (e.g., of 128MB in Intel SGX), some others (e.g., ARM TrustZone) do not place any limit on the TEE size. However, creating large TEEs is considered to be bad practice since it has proven to significantly increase the attack surface. Therefore, the TEE size must always be kept as small as possible independently of the type of TEEs and devices being used. This principle has already been adopted by industry, e.g., in the HiKey 960 board the TEE size is only 16MiB.

2.2 Privacy Risks in FL

Below we give a brief overview of the three main categories of privacy-related attacks in FL: data reconstruction, property inference, and membership inference attacks.

Data Reconstruction Attack (DRA). The DRA aims at reconstructing original input data based on the observed model or its gradients. It works by inverting model gradients based on generative adversarial attack-similar techniques [2, 16, 78], and consequently reconstructing the corresponding original data used to produce the gradients. DRAs are effective when attacking DNN's early layers, and when gradients have been only updated on a small batch of data (i.e., less than 8) [16, 49, 78]. As the server typically observes updated models of each client in plaintext, it is more likely for this type of leakages to exist at the server. By subtracting updated models with the global model, the server obtains gradients computed w.r.t. clients' data during the local training.

Property Inference Attack (PIA). The goal of PIAs is to infer the value of private properties in the input data. This attack is achieved by building a binary classifier trained on model gradients updated with auxiliary data and can be conducted on both server and client sides [45]. Specifically, property information, which also refers to the feature/latent information of the input data, is easier to be carried in stronger aggregation [47]. Even though clients in FL only observe multiple snapshots of broadcast global models that have been linearly aggregated on participating clients' updates, property information can still be well preserved, providing attack points to client-side adversaries.

Membership Inference Attack (MIA). The purpose of MIAs is to learn whether specific data instances are present in the training dataset. One can follow a similar attack mechanism as PIAs

---

[1] Recently, cloud providers also offer TEE-enabled infrastructure-as-a-service solutions to their customers (e.g., Microsoft Azure Confidential).

to build a binary classifier when conducting MIAs [52], although there are other methods, e.g., using shadow models [64]. The risk of MIAs can exist on both the server and client sides. Moreover, because membership is 'high-level' latent information, adversaries can perform MIAs on the final (well-trained) model and its last layer [52, 64, 73].

2.3 Privacy-preserving ML using TEEs

Running ML inside TEEs can hide model parameters from REE adversaries and consequently preserve privacy, as already used for light data analytics on servers [54, 62] and for heavy computations such as DNN training [18, 23, 49, 70]. However, due to TEEs' limited memory size, previous studies run only part of the model (e.g., sensitive layers) inside the TEE [18, 48, 49, 70]. In the on-device training case, DarkneTZ [49] runs the last layers with a Trusted Application inside TEEs to defend against MIAs, and leaves the first layers unprotected. DarkneTZ's evaluation showed no more than 10% overhead in CPU, memory, and energy on edge-like devices, demonstrating its suitability for client-side model updates in FL. In an orthogonal direction, several works leveraged clients' TEEs for verifying the integrity of local model training [10, 75], but did not consider privacy. Considering a broader range of attacks (e.g., DRAs and PIAs), it is essential to protect all layers instead of the last layers only, something that PPFL does.

2.4 Layer-wise DNN Training for FL

Instead of training the complete DNN model in an end-to-end fashion, one can train the model layer-by-layer from scratch, i.e., *greedy layer-wise training* [6, 35]. This method starts by training a shallow model (e.g., one layer) until its convergence. Next, it appends one more layer to the converged model and trains only this new layer [5]. Usually, for each greedily added layer, the model developer builds a new classifier on top of it in order to output predictions and compute training loss. Consequently, these classifiers provide multiple early exits, one per layer, during the forward pass in inference [29]. Furthermore, recently this method was shown to scale for large datasets such as ImageNet and to achieve performance comparable to regular end-to-end ML [5]. Notably, all previous studies on layer-wise training focused on generic ML.

Contribution. Our work is the first to build a DNN model in a FL setting with privacy-preserving guarantees using TEEs, by leveraging the greedy layer-wise training, and to train each DNN layer inside each FL client's TEE. Thus, PPFL satisfies the constraint of TEE's limited memory while protecting the model from the aforementioned privacy attacks. Interestingly, the classifiers built atop each layer may also provide personalization opportunities for the participating FL clients.

3 THREAT MODEL AND ASSUMPTIONS

Threat model. We consider a standard FL context where multiple client devices train a DNN locally and send their (local) model parameters to a remote, centralized server, which aggregates these parameters to create a global model [7, 27, 43]. The goal of adversaries is to obtain sensitive information embedded in the global model through data reconstruction [16, 78] or inference attacks [45, 52].

We consider two types of (passive) adversaries: (i) users of client devices who have access to distinct snapshots of the global model and (ii) the server's owner (e.g., a cloud or edge provider) who has access to the updated model gradients. Adversaries are assumed to be *honest-but-curious*, meaning that they allow FL algorithms to run as intended while trying to infer as much information as possible from the global model or gradients. Adversaries can have full control (i.e., root privileges) of the server or the client device, and can perform their attacks against any DNN layer. However, attacks against the TEE, such as side-channel attacks (e.g., Voltpillager [12]), physical attacks (e.g., Platypus [41]) and those that exploit weaknesses in TEEs (e.g., [38]) and their SDKs (e.g., [71]) are out of scope for this paper.

Assumptions. We assume that the server and enough participating FL client devices have a TEE whose memory size is larger than the largest layer of the DNN to be trained. This is the case in current FL DNNs. However, in the unlikely case that a layer does not fit in available TEEs, the network design needs to be adjusted with smaller, but more layer(s), or a smaller training batch size. We also assume that there is a secure way to bootstrap trust between the server TEE and each of the client device TEE (e.g., using a slightly modified version of the SIGMA key exchange protocol [32, 77], or attested TLS [30]), and that key management mechanisms exist to update and revoke keys when needed [55]. Finally, we assume that the centralized server will forward data to/from its TEE. Yet, it is important to note that if the server was malicious and would not do this, this would only affect the availability of the system (i.e., the security and privacy properties of our solution remain intact). This type of Denial-of-Service (DoS) attack is hard to defend against and is not considered within the standard TEE threat model.

4 PPFL FRAMEWORK

In this section, we first present an overview of the proposed system and its functionalities (Sec. 4.1), and then detail how the framework employs layer-wise training and aggregation in conjunction to TEEs in FL (Sec. 4.2).

4.1 System Overview

We propose a Privacy-preserving Federated Learning framework which allows clients to collaboratively train a DNN model while keeping the model's layers always inside TEEs during training. Figure 1 provides an overview of the framework and the various steps of the *greedy layer-wise training and aggregation.* In general, starting from the first layer, each layer is trained until convergence, before moving to the next layer. In this way, PPFL aims to achieve full privacy preservation without significantly increasing system cost. PPFL's design provides the following functionalities:

Privacy-by-design Guarantee. PPFL ensures that layers are always protected from adversaries while they are being updated. Privacy risks depend on the aggregation level and frequency with which they happen, when exposing the model or its layers [16, 45, 47]. In PPFL, lower-level information (i.e., original data and attributes) is not exposed because updated gradients during training are not accessible from adversaries (they happen inside the TEEs). This protects against DRAs and PIAs. However, when one of such
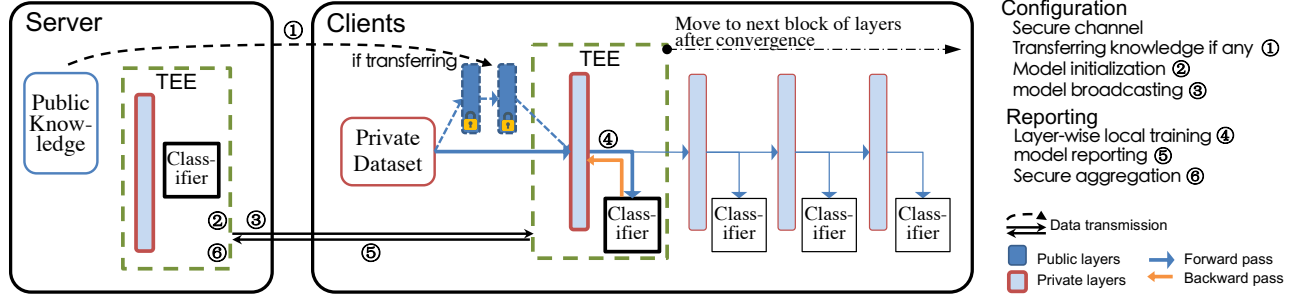
Figure 1: A schematic diagram of the PPFL framework. The main phases follow the system design in [7].

layers is exposed after convergence, there is a risk of MIAs. We follow a more practical approach based on the observation that membership-related information is only sensitive in the *last DNN layer*, making it vulnerable to MIAs as indicated in previous research [47, 49, 52, 59]. To avoid this risk on the final model, PPFL can keep the last layer inside the clients TEEs after training.

Device Selection. After the server and a set of TEE-enabled clients agree on the training of a DNN model via FL, clients inform the server about their TEE's memory constraints. The server then (re)constructs a DNN model suitable for this set of clients and selects the clients that can accommodate the model layers within their TEE. In each round, the server can select new clients and the device selection algorithm can follow existing FL approaches [21, 53].

Secure Communication Channels. The server establishes two secure communication channels with each of its clients: (i) one from its REE to the client's REE (e.g., using TLS) to exchange data with clients and (ii) a logical one from its TEE to the client's TEE for securely exchanging private information (e.g., model layer training information). In the latter case, the transmitted data is encrypted using cryptographic keys known only to the server and client TEEs and is sent over the REE-REE channel. It is important to note that the secure REE-REE channel is only an additional security layer. All privacy guarantees offered by PPFL are based on the hardware-backed cryptographic keys stored inside TEEs.

Model Initialization and Configuration. The server configures the model architecture, decides the layers to be protected by TEEs, and then initializes model parameters inside the TEE (step ②, Fig. 1). The latter ensures clients' local training starts with the same weight distribution [43, 72]. In addition, the server configures other training hyper-parameters such as learning rate, batch size, and epochs, before transmitting such settings to the clients (step ③, Fig. 1).

In cases of typical ML tasks such as image recognition where public knowledge is available such as pre-trained DNN models or public datasets with features similar to the client private data, the server can transfer this knowledge (especially in cross-device FL [27]) in order to bootstrap and speed up the training process. In both cases, this knowledge is contained in the first layers. Thus, the clients leave the first layers frozen and only train the last several layers of the global model. This training process is similar to the concept of transfer learning [9, 56, 69], where, in our case, public knowledge is transferred in a federated manner.

In PPFL, the server can learn from *public models*. Thus, during initialization, the server first chooses a model pre-trained on public data that have a similar distribution with private data. The server keeps the first layers, removes the last layer(s), and assembles new layer(s) atop the reserved first ones. These first layers are transferred to clients and are always kept frozen (step ①, Fig. 1). New layers, attached to the reserved layers, are trained inside each client's TEE, and then aggregated inside the server's TEE (steps ②~⑥, Fig. 1). In learning from *public datasets*, the server first performs an initial training to build the model based on public datasets.

Local Training. After model transmission and configuration using secure channels, each client starts local training on their data on each layer via a model partitioned execution technique (step ④, Fig. 1). We detail this step in Sec. 4.2.

Reporting and Aggregation. Once local training of a layer is completed inside TEEs, all participating clients report the layer parameters to the server through secure channels (step ⑤, Fig. 1). Finally, the server securely aggregates the received parameters within its TEE and applies FedAvg [43], resulting in a new global model layer (step ⑥, Fig. 1).

4.2 Layer-wise Training and Aggregation

In order to address the problem of limited memory inside a TEE when training a DNN model, we modify the greedy layer-wise learning technique proposed in [6] for general DNN training [5], to work in the FL setting. The procedure of layer-wise training and aggregation is detailed in the following Algorithms 1 and 2.

Algorithm 1. This algorithm details the actions taken by PPFL on the server side. When not specified, operations are carried out outside the TEE (i.e., in the REE). First, the server initializes the global DNN model with random weights or public knowledge (steps ①-②, Fig. 1). Thus, each layer l to be trained is initialized (θ_l) and prepared for broadcast. The server checks all available devices and constructs a set of participating clients whose TEE is larger than the required memory usage of l. Then, it broadcasts the model's layer to these participating clients (step ③, Fig. 1), via ClientUpdate() (see Algorithm 2). Upon receiving updates from all participating clients, the server decrypts the layer weights, performs secure layer aggregation and averaging inside its TEE (step ⑥), and broadcasts the new version of l to the clients for the next FL round. Steps ②~⑥ are repeated until the training of l converges, or a fixed number of

---

Algorithm 1: PPFL-Server with TEE

Input:

- Number of all clients: N
- TEE memory size of Client n: $S^{(n)}$
- Memory usage of layers $\{1, ..., L\}$ in training (forward and backward pass in total): $\{S_1, ..., S_L\}$
- Communication rounds: R

Output: Aggregated final parameters: $\{\theta_1^0, ..., \theta_L^0\}$

% *Layer-wise client updates*
for $l \in \{1, ..., L\}$ **do**

   % *Select clients with enough TEE memory*
   Initialize participating client list $\mathbf{J} = \{\}$
   for $n \in \{1, ..., N\}$ **do**
      if $S^{(n)} > S_l$ **then**
         $\mathbf{J} \leftarrow \mathbf{J} \cup \{n\}$
   Initialize θ_l (parameters of layers l) in TEE

   for $r \in \{1, ..., R\}$ **do**
      for $j \in \mathbf{J}$ **do**
         % *clients' local updating: see Algorithm 2*
         $\theta_l^{(j)} = $**ClientUpdate**$(l, \theta_l)$
      % *FedAvg with Secure Aggregation*
      $\theta_l = \frac{1}{\text{size}(\mathbf{J})} \sum_{j \in \mathbf{J}} \theta_l^{(j)}$ in TEE

   Save θ_l from TEE as θ_l^0 in REE

return $\{\theta_1^0, ..., \theta_L^0\}$

---

rounds are completed. Then, this layer is considered fully trained (θ_l^0), it is passed to the REE, and is broadcasted to all clients to be used for training the next layer. Interestingly, PPFL also allows grouping *multiple layers into blocks* and training each block inside client TEEs in a similar fashion as the individual layers. This option allows for better utilization of the memory space available inside each TEE and reduces communication rounds for the convergence of more than one layer at the same time.

Algorithm 2. This algorithm details the actions taken by PPFL on the client side. Clients load the received model parameters from the server and decrypt and load the target training layer l inside their TEEs. More specifically, in the front, this new layer l connects to the previous pre-trained layer(s) that are frozen during training. In the back, the clients attach on l their *own derived classifier*, which consists of fully connected layers and a softmax layer as the model exit. Then, for each epoch, the training process iteratively goes through batches of data and performs both *forward and backward passes* [36] to update both the layer under training and the classifier inside the TEE (step ④, Fig. 1). During this process, a *model partitioned execution* technique is utilized, where intermediate representations of the previously trained layers are passed from the REE to the TEE via shared memory in the forward pass. After local training is completed (i.e., all batches and epochs are done), each client sends via the secure channel the (encrypted) layer's weights from its TEE to the server's TEE (step ⑤).

Algorithm 2: ClientUpdate(l, θ_l) with TEEs

Initialization:

- Local dataset $\mathcal{X}$: data $\{x\}$ and labels $\{y\}$
- Trained final parameters of all previous layers, i.e., $\theta_1^0, \theta_2^0, ..., \theta_{l-1}^0$
- Number of local training epochs: E
- Activation function: $\sigma()$ and loss function: ℓ
- Classifier: $C()$

Input:

- Target layer: l
- Broadcast parameters of layer l: θ_l

Output: Updated parameters of layer l: θ_l

% *Weights and biases of layers* $1, ..., (l-1)$ *and* l
for $i \in \{1, ..., l-1\}$ **do**
   $\{W_i, b_i\} \leftarrow \theta_i^0$
$\{W_l, b_l\} \leftarrow \theta_l$ in TEE

% *Training process*
for $e \in \{1, .., E\}$ **do**
   for $\{x, y\} \in \mathcal{X}$ **do**

      % *Forward pass*
      Intermediate representation $T_0 = x$
      for $i \in \{1, ..., l-1\}$ **do**
         $T_i = \sigma(W_i T_{i-1} + b_i)$

      $T_l = \sigma(W_l T_{l-1} + b_l)$
      $\ell \leftarrow \ell(C(T_l), y)$

      % *Backward pass*
      $\frac{\partial \ell}{\partial C}$ to update parameters of C
      % *Updating layer* l
      $W_l \leftarrow W_l + \frac{\partial \ell}{\partial W_l}$; $b_l \leftarrow b_l + \frac{\partial \ell}{\partial b_l}$

$\{\text{in TEE}\}$ (brace spanning forward and backward pass)

$\theta_l = \{W_l, b_l\}$ in TEE

return θ_l

Model Partitioned Execution. The above learning process is based on a technique that conducts model training (including both forward and backward passes) across REEs and TEEs, namely model partitioned execution. The transmission of the forward activations (i.e., intermediate representation) and updated parameters happens between the REE and the TEE via *shared memory*. On a high level, when a set of layers is in the TEE, activations are transferred from the REE to the TEE (see Algorithm 2). Assuming global layer l is under training, the layer with its classifier $C(.)$ are executed in the TEE, and the previous layers (i.e., 1 to $l-1$) are in the REE.

Before training, layer l's parameters are loaded and decrypted securely within the TEE. During the *forward pass*, local data x are inputted, and the REE processes the previous layers from 1 to $l-1$ and invokes a command to transfer the layer $l-1$'s activations (i.e., T_{l-1}) to the secure memory through a buffer in shared memory. The TEE switches to the corresponding invoked command in order to receive layer $l-1$'s activations and processes the forward pass of layer l and classifier $C(.)$ in the TEE.

During the *backward pass*, the TEE computes the $C(.)$'s gradients based on received labels $\mathbf{y}$ and outputs of $C(.)$ (produced in the forward pass) and uses them to compute the gradients of the layer l in the TEE. The training of this batch of data (i.e., $\mathbf{x}$) finishes here, and there is no need to transfer l's errors from the TEE to the REE via shared memory, as previous layers are frozen outside the TEE. After that, the parameters of layer l are encrypted and passed to the REE, ready to be uploaded to the server, corresponding to the *FedSGD* [11]. Further, *FedAvg* [43] which requires multiple batches to be processed before updating, repeats the same number of forward and backward passes across the REE and the TEE for each batch of data.

Algorithmic Complexity Analysis. Next, we analyze the algorithmic complexity of PPFL and compare it to standard end-to-end FL. For the global model's layers $l \in \{1, \ldots, L\}$, we denote the forward and backward pass cost on layer l as F_l and B_l, respectively. The corresponding cost on the classifier is denoted as F_c and B_c. Then, in end-to-end FL, the total training cost for one client is:

$$\left(\sum_{l=1}^{L} (F_l + B_l) + F_c + B_c \right) \cdot S \cdot E \qquad (1)$$

where S is the number of steps in one epoch (i.e., number of samples inside local datasets divided by the batch size). As in PPFL all layers before the training layer l are kept frozen, the cost of training layer l is $(\sum_{k=1}^{l} F_k + F_c + B_l + B_c) \cdot S \cdot E$. Then, by summation, we get the total cost of all layers as:

$$\left(\sum_{l=1}^{L} \sum_{k=1}^{l} F_k + \sum_{l=1}^{L} B_l + L \cdot (F_c + B_c) \right) \cdot S \cdot E \qquad (2)$$

By comparing Equations 1 and 2, we see the overhead of PPFL comes from: (i) repeated forward pass in previous layers ($l \in \{1, \ldots, l-1\}$) when training layer l, and (ii) repeated forward and backward pass for the classifier atop layer l.

5 IMPLEMENTATION & EVALUATION SETUP

In this section, we first describe the implementation of the PPFL system (Sec. 5.1), and then detail how we assess its performance on various DNN models and datasets (Sec. 5.2) using different metrics (Sec. 5.3). We follow common setups of past FL systems [43, 72] and on-device TEE works [1, 49].

5.1 PPFL Prototype

We implement the *client-side* of PPFL by building on top of DarkneTZ [49], in order to support on-device FL with Arm TrustZone. In total, we changed 4075 lines of code of DarkneTZ in C. We run the client-side on a HiKey 960 Board, which has four ARM Cortex-A73 and four ARM Cortex-A53 cores configured at 2362MHz and 533MHz, respectively, as well as a 4GB LPDDR4 RAM with 16MiB TEE secure memory (i.e., TrustZone). Since the CPU power/frequency setting can impact the TrustZone's performance [1], we execute the on-device FL training with full CPU frequency. In order to emulate multiple device clients and their participation in FL rounds, we use the HiKey board in a repeated, iterative fashion, one time per client device. We implement the *server-side* of PPFL on generic Darknet ML framework [58] by adding 751 lines of C code

Table 1: DNNs used in the evaluation of PPFL.

DNN	Architecture
LeNet [37, 43]	C20-MP-C50-MP-FC500-FC10
AlexNet [5, 34]	C128×3-AP16-FC10
VGG9 [65, 72]	C32-C64-MP-C128×2-MP-D0.05-C256×2 -MP-D0.1-FC512×2-FC10
VGG16 [65]	C64×2-MP-C128×2-MP-C256×3-C512×3 -MP-FC4096×2-FC1000-FC10
MobileNetv2 [61]	68 layers, unmodified refer to [61] for details

Architecture notation: Convolution layer (C) with a given number of filters; filter size is 5×5 in LeNet and 3×3 in AlexNet, VGG9, and VGG16. Fully Connected (FC) with a given number of neurons. All C and FC layers are followed by ReLU activation functions. MaxPooling (MP). AveragePooling (AP) with a given stride size. Dropout layer (D) with a given dropping rate.

based on Microsoft OpenEnclave [46] with Intel SGX. For this, an Intel Next Unit of Computing (ver.NUC8BEK, i3-8109U CPU, 8GB DDR4-2400MHz) was used with SGX-enabled capabilities.

Besides, we developed a set of bash shell scripts to control the *FL process* and create the *communication channels*. For the communication channels between server and client to be secure, we employ standard cryptographic-based network protocols such as SSH and SCP. All data leaving the TEE are encrypted using the Advanced Encryption Standard (AES) in Cipher Block Chaining (CBC) mode with random Initialization Values (IV) and 128-bit cryptographic keys. Without loss of generality, we opted for manually hardcoding the cryptographic keys inside the TEEs ourselves. Despite key management in TEE-to-TEE channels being an interesting research problem, we argue that establishing, updating, and revoking keys do not happen frequently and hence the overhead these tasks introduce is negligible compared to one from the DNN training.

The implementation of PPFL server and client is available for replication and extension: https://github.com/mofanv/PPFL.

5.2 Models and Datasets

We focus on Convolutional Neural Networks (CNNs) since the privacy risks we consider (Sec. 3 and 4.1) have been extensively studied on such DNNs [45, 52]. Also, layer-based learning methods mostly aim at CNN-like DNNs [5]. Specifically, in our PPFL evaluation, we employ DNNs commonly used in the relevant literature (Table 1).

For our experimental analysis, we used *MNIST* and *CIFAR*10, two datasets commonly employed by FL researchers. Note that in practice, FL training needs labeled data locally stored at the clients' side. Indeed, the number of labeled examples expected to be present in a real setting could be fewer than what these datasets may allocate per FL client. Nonetheless, using them allows comparison of our results with state-of-art end-to-end FL methods [16, 39, 72].

Specifically, LeNet is tested on MNIST [37] and all other models are tested on CIFAR10 [33]. The former is a handwritten digit image (28×28) dataset consisting of 60k training samples and 10k test samples with 10 classes. The latter is an object image (32×32×3) dataset consisting of 50k training samples and 10k test samples with 10 classes. We follow the setup in [43] to partition training datasets into 100 parts, one per client, in two versions: i) Independent and Identically Distributed (IID) where a client has samples of all classes; ii) Non-Independent and Identically Distributed (Non-IID) where a client has samples only from two random classes.

5.3 Performance Metrics

The evaluation of PPFL prototype presented in the next section focuses on assessing the framework from the point of view of (i) privacy of data, (ii) ML model performance, and (iii) client-side system cost. Although ML computations (i.e., model training) have the same precision and accuracy no matter in REEs or TEEs, PPFL changes the FL model training process into a layer-based training. This affects ML accuracy and the number of communication rounds needed for the model to converge (among others). Thus, we devise several metrics and perform extensive measurements to assess overall PPFL performance. We conduct system cost measurements only on client devices since their computational resources are more limited compared to the server. All experiments are done with 10% of the total number of clients (i.e., 10 out of 100) participating in each communication round. We run FL experiments on our PPFL prototype (Sec. 5.1) to measure the system cost. To measure privacy risks and ML model performance, we perform simulations on a cluster with multiple NVIDIA RTX6000 GPUs (24GB) nodes running PyTorch v1.4.0 under Python v3.6.0.

Model Performance. We measure three metrics to assess the performance of the model and PPFL-related process:

(1) *Test Accuracy*: ML accuracy of test data on a given FL model, for a fixed number of communication rounds.
(2) *Communication Rounds*: Iterations of communication between server and clients needed to achieve a particular test accuracy.
(3) *Amount of communication*: Total amount of data exchanged to reach a test accuracy. Transmitted data sizes may be different among communication rounds when considering different layers' sizes in layer-wise training.

Privacy Assessment. We measure privacy risk of PPFL by applying three FL-applicable, privacy-related attacks:

(1) Data Reconstruction Attack (DRA) [78]
(2) Property Inference Attack (PIA) [45]
(3) Membership Inference Attack (MIA) [52]

We follow the proposing papers and their settings to conduct each attack on the model trained in FL process.

Client-side System Cost. We monitor the efficiency of client on-device training, and measure the following device costs for PPFL-related process information:

(1) *CPU Execution Time (s)*: Time the CPU was used for processing the on-device model training, including time spent in REE and the TEE's user and kernel time, which is reported by using function `getrusage(RUSAGE_SELF)`.
(2) *Memory Usage (MB)*: We add REE memory (the maximum resident set size in RAM, accessible by `getrusage()`) and allocated TEE memory (accessible by `mdbg_check(1)`) to get the total memory usage.
(3) *Energy Consumption (J)*: Measured by all energy used to perform one on-device training step when the model runs with/without TEEs. For this, we use the *Monsoon High Voltage Power Monitor* [50]. We configure the power to HiKey board as 12V voltage while recording the current in a 50Hz sampling rate. Training

Table 2: Results of three privacy-related attacks (DRA, PIA and MIA) on PPFL vs. end-to-end (E2E) FL. Average score reported with 95% confidence interval in parenthesis.

Learning Method	Model	Privacy-related Attack		
		DRA, in MSE $^\alpha$	PIA, in AUC $^\delta$	MIA, in Prec. $^\epsilon$
E2E	AlexNet	0.017 (0.01)	0.930 (0.03)	0.874 (0.01)
	VGG9	0.008 (<0.01)	0.862 (0.05)	0.765 (0.04)
PPFL	AlexNet	~1.3	~0.5	0.506 (0.01)
	VGG9			0.507 (<0.01)

$^\alpha$MSE (mean-square error) measures the difference between constructed images and target images (range is $[0, \infty)$, and the lower MSE is, the more privacy loss); $^\delta$AUC refers to the area under receiver operating curve; $^\epsilon$Prec. refers to Precision. The range of both AUC and Prec. is $[0.5, 1]$ (assuming 0.5 is for random guesses), and the higher AUC or Prec. is, the more privacy loss).

with a high-performance power setting can lead to high temperature and consequently under-clocking. Thus, we run each trial with 2000 steps continuously, starting with 120s cooling time.

6 EVALUATION RESULTS

In this section, we present the experimental evaluation of PPFL aiming to answer a set of key questions.

6.1 How Effectively does PPFL Thwart Known Privacy-related Attacks?

To measure the exposure of the model to known privacy risks, we conduct data reconstruction, property inference, and membership inference attacks (i.e., DRAs, PIAs, and MIAs) on the PPFL model. While training AlexNet and VGG9 models on CIFAR10 in an IID setting. We compare the exposure of PPFL to these attacks against a standard, end-to-end FL-trained model. Table 2 shows the average performance of each attack in the same way it is measured in literature [45, 52, 78]: Mean-Square-Error (MSE) for the DRA, Area-Under-Curve (AUC) for the PIA, and Precision for the MIA.

From the results, it becomes clear that, while these attacks can successfully disclose private information in regular end-to-end FL, they fail in PPFL. As DRAs and PIAs rely on intermediate training models (i.e., gradients) that remain protected, PPFL can fully defend against them. The DRA can only reconstruct a fully noised image for any target image (i.e., an MSE of ~1.3 for the specific dataset), while the PIA always reports a random guess on private properties (i.e., an AUC of ~0.5). Regarding the MIA on final trained models, as PPFL keeps the last layer and its outputs always protected inside the client's TEE, it forces the adversary to access only previous layers, which significantly drops the MIA's advantage (i.e., Precision≈0.5). Thus, PPFL fully addresses privacy issues raised by such attacks.

6.2 What is the PPFL Communication Cost?

Predefined ML Performance. Next, we measure PPFL's communication cost to complete the FL process, when a specific ML performance is desired. For this, we first execute the standard end-to-end FL without TEEs for 150 rounds and record the achieved ML performance. Subsequently, we set the same test accuracy as a requirement, and measure the number of communication rounds

and amount of communication required by PPFL to achieve this ML performance.

In this experiment, we set the number of local epochs at clients as 10. We use SGD as the optimization algorithm and set the learning rate as 0.01, with a decay of 0.99 after each epoch. Momentum is set to 0.5 and the batch size to 16. When training each layer locally, we build one classifier on top of it. The classifier's architecture follows the last convolutional (Conv) layer and fully-connected (FC) layers of the target model (e.g., AlexNet or VGG9). Thus, the training of each global model's layer progresses until all Conv layers are finished. We choose AlexNet and VGG9 on CIFAR10, because MNIST is too simple for testing. Then, the classifier atop all Conv layers is finally trained to provide outputs for the global model. Note that we also aggregate the client classifiers while training one global layer to provide the test accuracy after each communication round. We perform these experiments on IID and Non-IID data.

Overall, the results in Table 3 show that, while trying to reach the ML performance achieved by the standard end-to-end FL system, PPFL adds small communication overhead, if any, to the FL process. In fact, in some cases, it can even reduce the communication cost, while preserving privacy when using TEEs. As expected, using Non-IID data leads to lower ML performance across the system, which also implies less communication cost for PPFL as well.

The reason why in many cases PPFL has reduced communication cost, while still achieving comparable ML performance, is that training these models on datasets such as CIFAR10 may not require training the complete model. Instead, during the early stage of PPFL's layer-wise training (e.g., first global layer+classifier), it can already reach good ML performance, and in some cases even better than training the entire model. We explore this aspect further in the next subsection. Consequently, and due to the needed rounds being fewer, the amount of communication is also reduced.

The increased cost when training VGG9 is due to the large number of neurons in the classifier's FC layer connected to the first Conv layer. Thus, even if the number of total layers considered (one global layer + classifier) is smaller compared to the latter stages (multiple global layers + classifier), the model size (i.e., number of parameters) can be larger.

Indeed, we are aware that by training any of these models on CIFAR10 [43] for more communication rounds, either the PPFL or the regular end-to-end FL can reach higher test accuracy such as 85% with standard *FedAvg*. However, the training rounds used here are sufficient for our needs, as our goal is to evaluate the performance of PPFL (i.e., what is the cost for reaching the same accuracy), and not to achieve the best possible accuracy on this classification task.

Communication Duration of FL Phases. In the next experiment, we investigate the wall-clock time needed for running PPFL's phases in one communication round: broadcast of the layer from server to clients, training of the layer at the client device, upload the layer to the server, aggregate all updates from clients and apply *FedAvg*. Depending on each layer's size and TEE memory size, batch size can start from 1 and go as high as the TEE allows. However, since our models are uneven in layer sizes (with VGG9 being the largest), we set the batch size to 1 to allow comparison, and also capture an upper bound on the possible duration of each phase in

Table 3: Communication overhead (rounds and amount) of PPFL to reach the same accuracy as end-to-end FL system.

Model	Data	Baseline Acc.$^\alpha$	Comm. Rounds	Comm. Amount
LeNet	IID	98.93%	56 $(0.37\times)^\delta$	0.38 $\times$
	Non-IID	97.06%$^\epsilon$	-	-
AlexNet	IID	68.50%	97 (0.65$\times$)	0.63 $\times$
	Non-IID	49.49%	79 (0.53$\times$)	0.53 $\times$
VGG9	IID	63.09%	171 (1.14$\times$)	2.87 $\times$
	Non-IID	46.70%	36 (0.24$\times$)	0.60 $\times$

$^\alpha$Acc.: Test accuracy of 150 communication rounds in end-to-end FL;
$^\delta$1$\times$ refers to no overhead; $^\epsilon$PPFL reaches a maximum of 95.99%.

Table 4: Time duration of FL phases in *one communication round*, when training LeNet, AlexNet and VGG9 models with PPFL and end-to-end (E2E) FL.

Model	Method	Duration of FL phases (s)				
		B.cast$^\alpha$	Training	Upload	Aggr.$^\delta$	Total
LeNet	E2E	4.520	2691.0	6.645	0.064	2702.3
	PPFL	18.96	6466.2	7.535	1.887	6496.5
	- *layer 1*	4.117	1063.3	1.488	0.426	1069.8
	- *layer 2*	4.670	2130.6	1.627	0.692	2138.3
	- *layer 3*	5.332	2315.2	1.745	0.676	2323.6
	- *clf.*$^\epsilon$	4.845	957.16	2.675	0.093	964.87
AlexNet	E2E	14.58	3772.0	6.122	0.061	3792.8
	PPFL	57.24	14236	16.89	3.290	14316
	- *layer 1*	16.20	2301.8	4.690	0.129	2322.9
	- *layer 2*	12.56	4041.1	4.777	0.174	4058.8
	- *layer 3*	10.31	4609.4	5.388	0.243	4625.6
	- *clf.*	18.17	3283.8	2.033	2.744	3309.5
VGG9	E2E	14.10	2867.1	8.883	0.067	2890.2
	PPFL	353.5	21389	173.8	4.066	21924
	- *layer 1*	127.5	4245.7	95.58	0.375	4469.5
	- *layer 2*	77.22	2900.6	24.82	0.207	3003.1
	- *layer 3*	79.18	3703.1	24.84	0.223	3807.6
	- *layer 4*	27.05	2987.9	12.15	0.235	3027.6
	- *layer 5*	21.47	2404.4	9.137	0.347	2435.7
	- *layer 6*	10.95	2671.0	4.768	0.571	2687.9
	- *clf.*	10.11	2476.4	2.478	2.108	2493.2

$^\alpha$B.cast: Broadcast; $^\delta$Aggr.: Aggregation; $^\epsilon$clf.: Classifier.

each model training. Indeed, we confirmed that increasing batch size for small models that allow it (e.g., AlexNet with batch size=16), incrementally reduces the duration of phases.

Table 4 shows the break-down of time taken for each phase, for three models and two datasets (LeNet on MNIST; AlexNet and VGG9 on CIFAR10) and IID data. As expected, layer-wise FL increases the total time compared to end-to-end FL because each layer is trained separately, but the previously trained and finalized layers still need to be processed in the forward pass. In fact, these results are in line with the complexity analysis shown earlier in Sec. 4.2, i.e., to finish the training of all layers, layer-wise training introduces a 3$\times$ or higher delay, similar to the number of layers. On the one hand, we argue that applications can tolerate this additional delay if they are to be protected from privacy-related attacks, despite the execution time increase being non-negligible and up to a few hours

of training. Indeed, models can be (re)trained on longer timescales (e.g., weekly, monthly), and rounds can have a duration of 10s of minutes, while being executed in an asynchronous manner. On the other hand, training one layer in PPFL costs similar time to the end-to-end FL training of the complete model. This highlights that the minimum client contribution time is the same as end-to-end FL: clients can choose to participate in portions of an FL round, and in just a few FL rounds. For example, a client may contribute to the model training for only a few layers in any given FL round.

Among all FL phases, local training costs the most, while the time spent in server aggregation and averaging is trivial, regardless if it is non-secure (i.e., end-to-end FL) or secure (PPFL). Regarding VGG9, layer-wise training of early layers significantly increases the communication time in broadcast and upload, because the Conv layers are with a small number of filters and consequently the following classifier's FC layer has a large size. This finding hints that selecting suitable DNNs to be trained in PPFL (e.g., AlexNet vs. VGG9) is crucial for practical performance. Moreover, and according to the earlier FL performance results (also see Table 2), it may not be necessary to train all layers to reach the desired ML utility.

6.3 Is the PPFL ML Performance Comparable to State-of-art FL?

In these experiments, we reduce the number of communication rounds that each layer in PPFL is trained to 50, finish the training process per layer, and compare its performance with centralized layer-wise training, as well as regular end-to-end FL. The latter trains the full model for all rounds up to that point. For example, if PPFL trains the first layer for 50 rounds, and then the second layer for 50 rounds, the end-to-end FL will train all the model (end-to-end) for 100 rounds.

As shown in Figure 2, training LeNet on the "easy" task of MNIST data (IID or not) leads quickly to high ML performance, regardless of the FL system used. Training AlexNet on IID and Non-IID CIFAR10 data can lead to test accuracy of 74% and 60.78%, respectively, while centralized training reaches 83.34%. Training VGG9, which is a more complex model on IID and Non-IID CIFAR10 data leads to lower performances of 74.60% and 38.35%, respectively, while centralized training reaches 85.09%. We note the drop of performance in PPFL when every new layer is considered into training. This is to be expected, since PPFL starts from scratch with the new layer, leading to a significant performance drop in the first FL rounds. Of course, towards the end of the 50 rounds, PPFL performance matches and in some cases surpasses that of end-to-end FL.

In general, with more layers being included in the training, the test accuracy increases. Interestingly, in more complex models (e.g., VGG9) with Non-IID data, PPFL can lead to a drop in ML performance when the number of layers keeps increasing. In fact, in these experiments, it only reaches ∼55% after finishing the second layer and drops. One possible reason for this degradation is that the first layers of VGG9 are small and maybe not capable of capturing heterogeneous features among Non-IID data, which consequently has a negative influence on the training of latter layers. On the other hand, this reminds us that we can have early exits for greedy layer-wise PPFL on Non-IID data. For example, clients that do not have enough data, or already have high test accuracy after
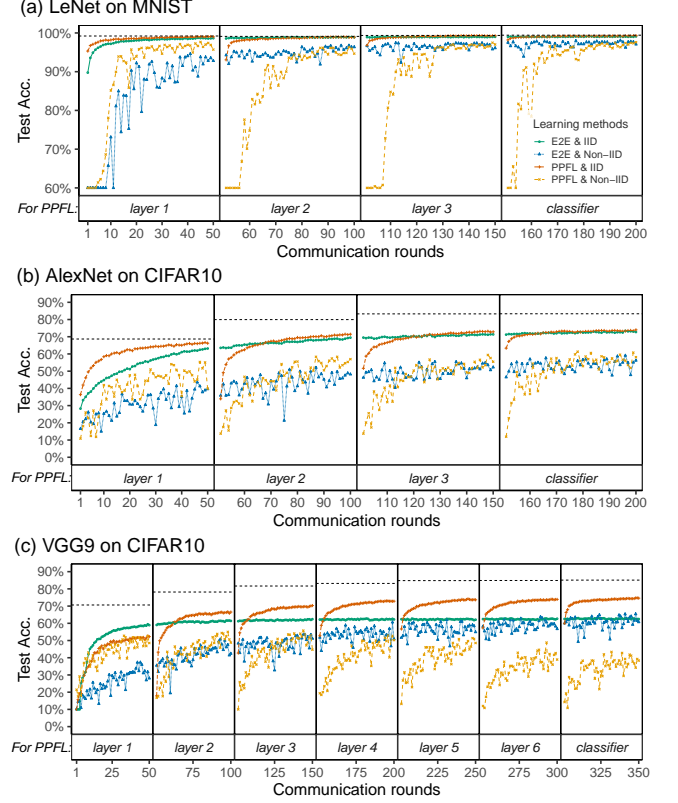

Figure 2: Test accuracy of training LeNet, AlexNet, and VGG9 models on IID and Non-IID datasets when using PPFL. Horizontal dashed lines refer to the accuracy that the centralized training reaches after every 50 epochs. Note: end-to-end (E2E) FL trains the complete model rather than each layer, and the 'Layer No.' at x-axis are *only* applicable to PPFL.

training the first layers can quit before participating in further communication rounds. Overall, the layer-wise training outperforms end-to-end FL during the training of the first or second layer.

We further discuss possible reasons for PPFL's better ML performance compared to end-to-end FL. On the one hand, this could be due to some DNN architectures (e.g., VGG9) being more suitable for layer-wise FL. For example, training each layer separately may allow PPFL to overcome possible local optima at which the backward propagation can "get stuck" in end-to-end FL. On the other hand, hyper-parameter tuning may help improve performance in both layer-wise and end-to-end FL, always with the risk of overfitting the data. Indeed, achieving the best ML performance possible was not our focus, and more in-depth studying is needed in the future, to understand under what setups layer-wise can perform better than end-to-end FL.

6.4 What is the PPFL Client-Side System Cost?

We further investigate the system performance and costs on the client devices with respect to CPU execution time, memory usage, and energy consumption. Figure 3 shows the results for all three
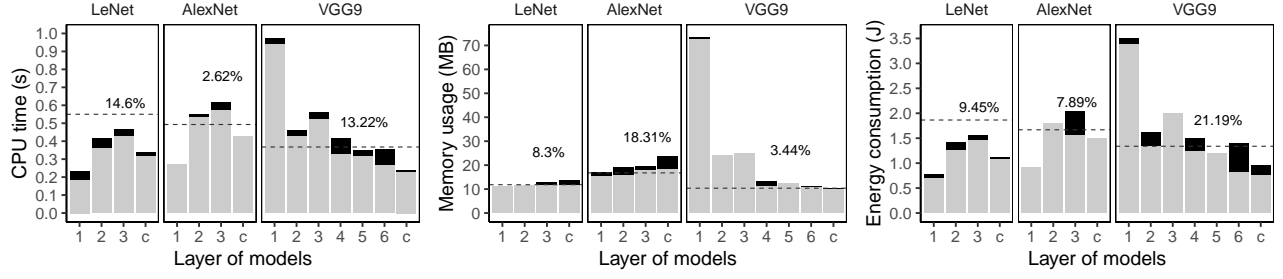
Figure 3: System performance of the client devices when training LeNet, AlexNet, and VGG9 using PPFL, measured on *one step of training* (i.e., one batch of data). The light grey bar (▢) refers to learning without TEEs, and the black bar (■) refers to overhead when the layer under training is inside the TEE. Percentage (%) of the overhead (averaged on one model) is shown above these bars. Horizontal dashed lines signify the cost of end-to-end FL. In x-axis, 'c' refers to 'classifier'.
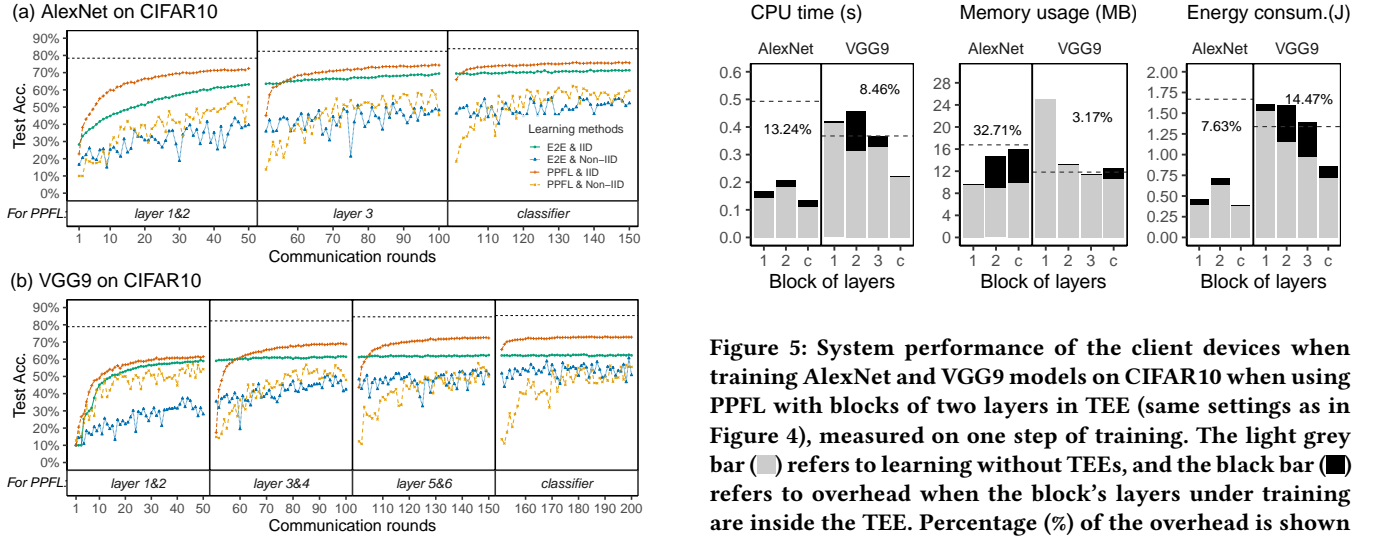

Figure 4: Test accuracy of training AlexNet and VGG9 models on CIFAR10 (IID and Non-IID) when using PPFL with blocks of two layers in TEE (Note: horizontal dashed lines refer to the accuracy that the end-to-end (E2E) FL reaches after 50 communication rounds).

Figure 5: System performance of the client devices when training AlexNet and VGG9 models on CIFAR10 when using PPFL with blocks of two layers in TEE (same settings as in Figure 4), measured on one step of training. The light grey bar (▢) refers to learning without TEEs, and the black bar (■) refers to overhead when the block's layers under training are inside the TEE. Percentage (%) of the overhead is shown above these bars. Horizontal dashed lines refer to the cost of end-to-end FL. 'c' refers to 'classifier'.

6.5 What is the PPFL ML and System Costs if Blocks of Layers were Trained in Clients?

As explained in Algorithm 1 of Sec. 4.2, if the TEEs can hold more than one layers, it is also possible to put a block of layers inside the TEE for training. Indeed, heterogeneous devices and TEEs can have different memory sizes, thus supporting a wide range of block sizes. For these experiments, we assume all devices have the same TEE size and construct 2-layer blocks, and measure the system's test accuracy and ML performance on CIFAR10. The performance of three or more layers inside TEEs could be measured in a similar fashion (if the TEE's memory can fit them). We do not test LeNet on MNIST because it can easily reach high accuracy (around 99%) as shown earlier and in previous studies [43, 72].

Results in Figure 4 indicate that training blocks of layers can reach similar or even better ML performance compared to training each layer separately (i.e., see Fig. 2). It can also improve the test accuracy of complex models such as VGG9, for which we noted a degradation of ML performance caused by the first layer's small size

metrics, when training LeNet on MNIST, AlexNet and VGG9 on CIFAR10, on IID data. The metrics are computed for one step of training (i.e., one batch of data). More training steps require analogously more CPU time and energy, but do not influence memory usage since the memory allocated for the model is reused for all subsequent steps. Here, we compare PPFL with layer-wise training without TEEs, to measure the overhead of using the TEE. Among the trained models, the maximum overhead is 14.6% for CPU time, 18.31% for memory usage, and 21.19% for energy consumption. In addition, when training each layer, PPFL has comparable results with end-to-end training (i.e., horizontal dashed lines in Figure 3).

Table 5: Reduction of communication rounds and amount when training 2-layer instead of 1-layer blocks.

| Model	Data	Comm. Rounds	Comm. Amount
AlexNet	IID	$0.65\times \rightarrow 0.18\times$	$0.63\times \rightarrow 0.27\times$
	Non-IID	$0.53\times \rightarrow 0.29\times$	$0.53\times \rightarrow 0.44\times$
VGG9	IID	$1.14\times \rightarrow 0.43\times$	$2.87\times \rightarrow 1.07\times$
	Non-IID	$0.24\times \rightarrow 0.11\times$	$0.60\times \rightarrow 0.27\times$

and incapacity to model the data (see Fig. 2). In addition, compared to training one layer at a time, training 2-layer blocks reduces the total required communication to reach the desired ML performance. In fact, while aiming to reach same baseline accuracy as in Table 3, training 2-layer blocks requires half or less of communication cost than 1-layer blocks see Table 5. Also, layer-wise training outperforms end-to-end FL for similar reasons as outlined for Figure 2.

Regarding the system cost, results across models show that the maximum overhead is 13.24% in CPU time, 32.71% in memory usage, and 14.47% in energy consumption (see Fig. 5). Compared to training one layer at a time, training layer blocks does not always increase the overhead. For example, overhead when running VGG9 drops from 13.22% to 8.46% in CPU, from 2.44% to 3.17% in memory usage, and from 21.19% to 14.47% in energy consumption. One explanation is that combining layers into blocks amortizes the cost of "expensive" with "cheap" layers. Interestingly, PPFL still has a comparable cost with end-to-end FL training.

6.6 Can Bootstrapping the PPFL with Public Knowledge Help?

We investigate how the backend server of PPFL can use existing, public models to bootstrap the training process for a given task. For this purpose, we leverage two models (MobileNetv2 and VGG16) pre-trained on ImageNet to the classification task on CIFAR10. Because these pre-trained models contain sufficient knowledge relevant to the target task, training the last few layers is already adequate for a good ML performance. Consequently, we can freeze all Conv layers and train the last FC layers within TEEs, thus protecting them as well. By default, MobileNetv2 has one FC layer, and VGG16 has three FC layers at the end. We test both cases that one and three FC layers are attached and re-trained for these two models, respectively. CIFAR10 is resized to 224×224 in order to fit with the input size of these pre-trained models. We start with a smaller learning rate of 0.001 to avoid divergence and a momentum of 0.9 because the feature extractors are well-trained.

Test Accuracy. Figure 6 shows that the use of pre-trained first layers (i.e., feature extractors) to bootstrap the learning process can help the final PPFL models reach test accuracy similar to centralized training. Interestingly, transferring pre-trained layers from VGG16 can reach higher test accuracy than MobileNetv2. This is expected because VGG16 contains many more DNN parameters than MobileNetv2, which provides better feature extraction capabilities. Surprisingly, attaching and training more FC layers at the end of any of the models does not improve test accuracy. This can be due to the bottleneck of the transferred feature extractors, which since they are frozen, they do not allow the model to *fully* capture the variability of the new data.
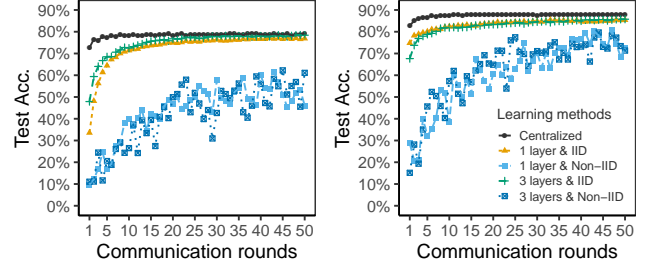
(a) Transfer from MobileNetv2   (b) Transfer from VGG16

Figure 6: Test accuracy of training on CIFAR10 (IID and Non-IID) with public models MobileNetv2 and VGG16, pre-trained on ImageNet). Both models are trained and tested with 1 and 3 FC layers attached at the end of each model.
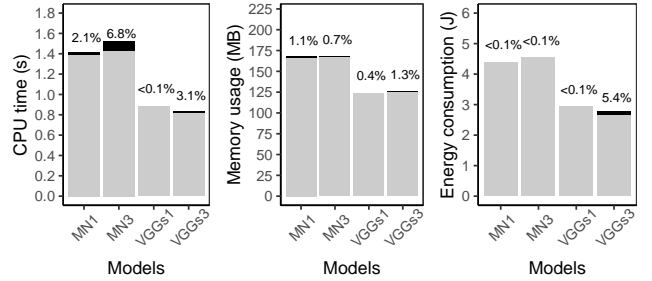

Figure 7: System performance of client devices when training with transferred public models on CIFAR10, measured on 1 step of training. Light grey bar (▢): learning without TEEs; Black bar (■): overhead when layers under training are in TEE. Percentage (%) of overhead shown above bars. MN1: MobileNetv2 with one layer for training (i.e., '1 layer' in Figure 6a). VGGs: a small size of VGG16.

Client-side System Cost. In order to measure client-side cost under this setting, we need to do some experimental adjustments. The VGG16 (even the last FC layers) is too large to fit in TEEs. Thus, we reduce the batch size to 1 and proportionally scale down all layers (e.g., from 4096 to 1024 neurons for one FC layer). Indeed, scaling layers may lead to biases in results, but the actual performance cannot be worse than this estimation. As shown in [49], larger models have less overhead because the last layers are relatively smaller compared to the complete size of the model.

Interestingly, results shown in Figure 7 indicate that when we train and keep the last FC layers inside the client's on-device TEEs, there is only a small overhead incurred in terms of CPU time (6.9%), memory usage (1.3%), and energy consumption (5.5%) in either model. These results highlight that transferring knowledge can be a good alternative for bootstrapping PPFL training and keep system overhead low. In addition, we note that when the server does not have suitable public models, it is possible to first train a model on public datasets that have similar distribution with local datasets. We refer to Appendix A.1 for more details on experimental results.

7 DISCUSSION & FUTURE WORK

Key Findings. PPFL's experimental evaluation showed that:

- Protecting the training process (i.e., gradient updates) inside TEEs, and exposing layers only after convergence can thwart data reconstruction and property inference attacks. Also, keeping a model's last layer inside TEEs mitigates membership inference attacks.
- Greedy layer-wise FL can achieve comparable ML utility with end-to-end FL. While layer-wise FL increases the total of communication rounds needed to finish all layers, it can reach the same test accuracy as end-to-end FL with fewer rounds ($0.538\times$) and amount of communication ($1.002\times$).
- Most PPFL system cost comes from clients' local training: up to ~15% CPU time, ~18% memory usage, and ~21% energy consumption in client cost when training different models and data, compared to training without TEEs.
- Training 2-layer blocks decreases communication cost by at least half, and slightly increases system overhead (i.e., CPU time, memory usage, energy consumption) in cases of small models.
- Bootstrapping PPFL training process with pre-trained models can significantly increase ML utility, and reduce overall cost in communications and system overhead.

Dishonest Attacks. The attacks tested here assume the classic 'honest-but-curious' adversary [57]. In FL, however, there are also dishonest attacks such as backdoor [4, 67] or poisoning attacks [15], whose goal is to actively change the global model behavior, e.g., for surreptitious unauthorized access to the global model [26]. In the future, we will investigate how TEEs' security properties can defend against such attacks.

Privacy and Cost Trade-off. PPFL guarantees 'full' privacy by keeping layers inside TEEs. However, executing computations in secure environments inevitably leads to system costs. To reduce such costs, one can relax their privacy requirements, potentially increasing privacy risks due inference attacks with higher "advantage" [76]. For example, clients who do not care about high-level information leakages (i.e., learned model parameters), but want to protect the original local data, can choose to hide only the first layers of the model in TEEs. We expect that by dropping clients already achieving good performance when training latter layers, we could gain better performance. This may further benefit personalization and achieve better privacy, utility, and cost trade-offs.

Model Architectures. The models tested in our layer-wise FL are linear links cross consecutive layers. However, our framework can be easily extended to other model architectures that have been studied in standard layer-wise training. For example, one can perform layer-wise training on (i) Graph Neural Networks by disentangling feature aggregation and feature transformation [74], and (ii) Long Short-Term Memory networks (LSTMs), by adding hidden layers [60]. There are other architectures that contain skipping connections to jump over some layers such as ResNet [19]. No layer-wise training has been investigated for ResNets, but training a block of layers could be attempted by including the jumping shortcut inside a block.

Accelerating Local Training. PPFL uses only the CPU of client devices for local training. Training each layer does not introduce parallel processing on a device. Indeed, more effective ways to perform this compute load can be devised. One way is that clients could use specialized processors (i.e., GPUs) to accelerate training. PPFL's design can integrate such advances mainly in two ways. First, the client can outsource the first, well-trained, but non-sensitive layers, to specialized processors that can share computation and speed-up local training. Second, recently proposed GPU-based TEEs can support intensive deep learning-like computation in high-end servers [22, 24]. Thus, such TEEs on client devices can greatly speed-up local training. However, as GPU-TEE still requires small TCB to restrict attack surface, PPFL's design can provide a way to leverage limited TEE space for privacy-preserving local training.

Federated Learning Paradigms. PPFL was tested with *FedAvg*, but there are other state-of-art FL paradigms that are compatible with PPFL. PPFL leverages greedy layer-wise learning but does not modify the hyper-parameter determination and loss function (which have been improved in *FedProx* [39]) or aggregation (which is neuron matching-based in *FedMA* [72]). Compared with PPFL that trains one layer until convergence, *FedMA*, which also uses layer-wise learning, trains each layer for one round, and then moves to the next layer. After finishing all layers, it starts again from the first. Thus, *FedMA* is still vulnerable because gradients of one layer are accessible to adversaries. PPFL could leverage *FedMA*'s neuron-matching technique when dealing with heterogeneous (i.e., Non-IID) data [28]. Besides, our framework is compatible with other privacy-preserving techniques (e.g., differential privacy) in FL. This is useful during the model usage phase where some users may not have TEEs. PPFL can also be useful to systems such as FLaaS [31] that enable third-party applications to build collaborative ML models on the device shared by said applications.

8 CONCLUSION

In this work, we proposed PPFL, a practical, privacy-preserving federated learning framework, which protects clients' private information against known privacy-related attacks. PPFL adopts greedy layer-wise FL training and updates layers always inside Trusted Execution Environments (TEEs) at both server and clients. We implemented PPFL with mobile-like TEE (i.e., TrustZone) and server-like TEE (i.e., Intel SGX) and empirically tested its performance. For the first time, we showed the possibility of fully guaranteeing privacy and achieving comparable ML model utility with regular end-to-end FL, without significant communication and system overhead.

9 ACKNOWLEDGMENTS

REFERENCES

[1] Amacher, J., and Schiavoni, V. On the performance of arm trustzone. In *IFIP International Conference on Distributed Applications and Interoperable Systems* (2019), Springer, pp. 133–151.

[2] Aono, Y., Hayashi, T., Wang, L., Moriai, S., et al. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security 13*, 5 (2017), 1333–1345.

[3] Bagdasaryan, E., Poursaeed, O., and Shmatikov, V. Differential privacy has disparate impact on model accuracy. In *Advances in Neural Information Processing Systems* (2019), pp. 15479–15488.

[4] Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D., and Shmatikov, V. How to backdoor federated learning. In *International Conference on Artificial Intelligence and Statistics* (2020), PMLR, pp. 2938–2948.

[5] Belilovsky, E., Eickenberg, M., and Oyallon, E. Greedy layerwise learning can scale to imagenet. In *International conference on machine learning* (2019), PMLR, pp. 583–593.

[6] Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. Greedy layer-wise training of deep networks. *Advances in neural information processing systems 19* (2006), 153–160.

[7] Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konečný, J., Mazzocchi, S., McMahan, H. B., et al. Towards federated learning at scale: System design. In *Conference on Machine Learning and Systems* (2019).

[8] Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., Ramage, D., Segal, A., and Seth, K. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (2017), pp. 1175–1191.

[9] Brownlee, J. *A Gentle Introduction to Transfer Learning for Deep Learning*, 2019 (accessed November 11, 2020).

[10] Chen, H., Fu, C., Rouhani, B. D., Zhao, J., and Koushanfar, F. Deepattest: An end-to-end attestation framework for deep neural networks. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)* (2019), IEEE, pp. 487–498.

[11] Chen, J., Pan, X., Monga, R., Bengio, S., and Jozefowicz, R. Revisiting distributed synchronous sgd. In *ICLR Workshop Track* (2016).

[12] Chen, Z., Vasilakis, G., Murdock, K., Dean, E., Oswald, D., and Garcia, F. D. Voltpillager: Hardware-based fault injection attacks against intel SGX enclaves using the SVID voltage scaling interface. In *30th USENIX Security Symposium* (Vancouver, B.C., Aug. 2021).

[13] Costan, V., and Devadas, S. Intel sgx explained. *IACR Cryptol. ePrint Arch. 2016*, 86 (2016), 1–118.

[14] Dwork, C., Roth, A., et al. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science 9*, 3-4 (2014), 211–407.

[15] Fang, M., Cao, X., Jia, J., and Gong, N. Local model poisoning attacks to byzantine-robust federated learning. In *29th USENIX Security Symposium* (2020), pp. 1605–1622.

[16] Geiping, J., Bauermeister, H., Dröge, H., and Moeller, M. Inverting gradients–how easy is it to break privacy in federated learning? *arXiv preprint arXiv:2003.14053* (2020).

[17] Geyer, R. C., Klein, T., and Nabi, M. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557* (2017).

[18] Gu, Z., Huang, H., Zhang, J., Su, D., Jamjoom, H., Lamba, A., Pendarakis, D., and Molloy, I. Yerbabuena: Securing deep learning inference data via enclave-based ternary model partitioning. *arXiv preprint arXiv:1807.00969* (2018).

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

[20] Hitaj, B., Ateniese, G., and Perez-Cruz, F. Deep models under the gan: information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (2017), pp. 603–618.

[21] Huang, T., Lin, W., Wu, W., He, L., Li, K., and Zomaya, A. Y. An efficiency-boosting client selection scheme for federated learning with fairness guarantee. *arXiv preprint arXiv:2011.01783* (2020).

[22] Hunt, T., Jia, Z., Miller, V., Szekely, A., Hu, Y., Rossbach, C. J., and Witchel, E. Telekine: Secure computing with cloud gpus. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI'20)* (2020), pp. 817–833.

[23] Hunt, T., Song, C., Shokri, R., Shmatikov, V., and Witchel, E. Chiron: Privacy-preserving machine learning as a service. *arXiv preprint arXiv:1803.05961* (2018).

[24] Jang, I., Tang, A., Kim, T., Sethumadhavan, S., and Huh, J. Heterogeneous isolated execution for commodity gpus. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (2019), pp. 455–468.

[25] Jayaraman, B., and Evans, D. Evaluating differentially private machine learning in practice. In *28th USENIX Security Symposium (USENIX Security 19)* (Santa Clara, CA, Aug. 2019), USENIX Association, pp. 1895–1912.

[26] Jere, M. S., Farnan, T., and Koushanfar, F. A taxonomy of attacks on federated learning. *IEEE Security & Privacy* (2020), 0–0.

[27] Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N.,

[28] Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977* (2019).

[28] Katevas, K., Bagdasaryan, E., Waterman, J., Safadieh, M. M., Birrell, E., Haddadi, H., and Estrin, D. Policy-based federated learning. *arXiv preprint arXiv:2003.06612* (2021).

[29] Kaya, Y., Hong, S., and Dumitras, T. Shallow-deep networks: Understanding and mitigating network overthinking. In *International Conference on Machine Learning* (2019), PMLR, pp. 3301–3310.

[30] Knauth, T., Steiner, M., Chakrabarti, S., Lei, L., Xing, C., and Vij, M. Integrating remote attestation with transport layer security. *arXiv preprint arXiv:1801.05863* (2018).

[31] Kourtellis, N., Katevas, K., and Perino, D. Flaas: Federated learning as a service. In *Workshop on Distributed ML* (2020), ACM CoNEXT.

[32] Krawczyk, H. Sigma: The 'sign-and-mac'approach to authenticated diffie-hellman and its use in the ike protocols. In *Annual International Cryptology Conference* (2003), Springer, pp. 400–425.

[33] Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. *Citeseer* (2009).

[34] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Communications of the ACM 60*, 6 (2017), 84–90.

[35] Larochelle, H., Bengio, Y., Louradour, J., and Lamblin, P. Exploring strategies for training deep neural networks. *Journal of machine learning research 10*, 1 (2009).

[36] LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *nature 521*, 7553 (2015), 436–444.

[37] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (1998), 2278–2324.

[38] Lee, J., Jang, J., Jang, Y., Kwak, N., Choi, Y., Choi, C., Kim, T., Peinado, M., and Kang, B. B. Hacking in Darkness: Return-Oriented Programming against Secure Enclaves. In *Proceedings of the 26th USENIX Conference on Security Symposium* (2017), pp. 523–539.

[39] Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., and Smith, V. Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127* (2018).

[40] Linaro.org. *Open Portable Trusted Execution Environment*, 2020 (accessed September 3, 2020).

[41] Lipp, M., Kogler, A., Oswald, D., Schwarz, M., Easdon, C., Canella, C., and Gruss, D. PLATYPUS: Software-based Power Side-Channel Attacks on x86. In *2021 IEEE Symposium on Security and Privacy (SP)* (2021), IEEE.

[42] Liu, Y., Kang, Y., Xing, C., Chen, T., and Yang, Q. A secure federated transfer learning framework. *IEEE Intelligent Systems* (2020).

[43] McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics* (2017), PMLR, pp. 1273–1282.

[44] McMahan, H. B., Ramage, D., Talwar, K., and Zhang, L. Learning differentially private recurrent language models. In *International Conference on Learning Representations* (2018).

[45] Melis, L., Song, C., De Cristofaro, E., and Shmatikov, V. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy (SP)* (2019), IEEE, pp. 691–706.

[46] Microsoft. *Open Enclave SDK*, 2020 (accessed Decemenber 4, 2020).

[47] Mo, F., Borovykh, A., Malekzadeh, M., Haddadi, H., and Demetriou, S. Layer-wise characterization of latent information leakage in federated learning. *ICLR Distributed and Private Machine Learning workshop* (2021).

[48] Mo, F., and Haddadi, H. Efficient and private federated learning using tee. In *EuroSys* (2019).

[49] Mo, F., Shamsabadi, A. S., Katevas, K., Demetriou, S., Leontiadis, I., Cavallaro, A., and Haddadi, H. Darknetz: towards model privacy at the edge using trusted execution environments. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services* (2020), pp. 161–174.

[50] Monsoon. Monsoon solutions inc. home page. https://www.msoon.com/, 2020 (accessed November 12, 2020).

[51] Naehrig, M., Lauter, K., and Vaikuntanathan, V. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop* (2011), pp. 113–124.

[52] Nasr, M., Shokri, R., and Houmansadr, A. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE Symposium on Security and Privacy (SP)* (2019), IEEE, pp. 739–753.

[53] Nishio, T., and Yonetani, R. Client selection for federated learning with heterogeneous resources in mobile edge. In *IEEE International Conference on Communications (ICC)* (2019), IEEE, pp. 1–7.

[54] Ohrimenko, O., Schuster, F., Fournet, C., Mehta, A., Nowozin, S., Vaswani, K., and Costa, M. Oblivious multi-party machine learning on trusted processors. In *25th USENIX Security Symposium* (2016), pp. 619–636.

[55] Paladi, N., Karlsson, L., and Elbashir, K. Trust Anchors in Software Defined Networks. In *Computer Security* (2018), pp. 485–504.

[56] Pan, S. J., and Yang, Q. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering 22*, 10 (2009), 1345–1359.

[57] Paverd, A., Martin, A., and Brown, I. Modelling and automatically analysing privacy properties for honest-but-curious adversaries. *Tech. Rep.* (2014).

[58] Redmon, J. Darknet: Open source neural networks in c. http://pjreddie.com/darknet/, 2013–2016.

[59] Sablayrolles, A., Douze, M., Schmid, C., Ollivier, Y., and Jégou, H. White-box vs black-box: Bayes optimal strategies for membership inference. In *International Conference on Machine Learning* (2019), pp. 5558–5567.

[60] Sagheer, A., and Kotb, M. Unsupervised pre-training of a deep lstm-based stacked autoencoder for multivariate time series forecasting problems. *Scientific reports 9*, 1 (2019), 1–16.

[61] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2018), pp. 4510–4520.

[62] Schuster, F., Costa, M., Fournet, C., Gkantsidis, C., Peinado, M., Mainar-Ruiz, G., and Russinovich, M. Vc3: Trustworthy data analytics in the cloud using sgx. In *2015 IEEE Symposium on Security and Privacy* (2015), IEEE, pp. 38–54.

[63] Microsoft SEAL (release 3.5). https://github.com/Microsoft/SEAL, Apr. 2020. Microsoft Research, Redmond, WA.

[64] Shokri, R., Stronati, M., Song, C., and Shmatikov, V. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)* (2017), IEEE, pp. 3–18.

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

[66] Subramani, P., Vadivelu, N., and Kamath, G. Enabling fast differentially private sgd via just-in-time compilation and vectorization. *arXiv preprint arXiv:2010.09063* (2020).

[67] Sun, Z., Kairouz, P., Suresh, A. T., and McMahan, H. B. Can you really backdoor federated learning? *arXiv preprint arXiv:1911.07963* (2019).

[68] Testuggine, D., and Mironov, I. *Introducing Opacus: A high-speed library for training PyTorch models with differential privacy*, 2020 (accessed January 1, 2021).

[69] Torrey, L., and Shavlik, J. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI global, 2010, pp. 242–264.

[70] Tramèr, F., and Boneh, D. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. In *International Conference on Learning Representations (ICLR)* (2019).

[71] Van Bulck, Jo and Oswald, David and Marin, Eduard and Aldoseri, Abdulla and Garcia, Flavio D. and Piessens, Frank. A Tale of Two Worlds: Assessing the Vulnerability of Enclave Shielding Runtimes. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)* (2019), pp. 1741–1758.

[72] Wang, H., Yurochkin, M., Sun, Y., Papailiopoulos, D., and Khazaeni, Y. Federated learning with matched averaging. *arXiv preprint arXiv:2002.06440* (2020).

[73] Yeom, S., Giacomelli, I., Fredrikson, M., and Jha, S. Privacy risk in machine learning: Analyzing the connection to overfitting. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)* (2018), IEEE, pp. 268–282.

[74] You, Y., Chen, T., Wang, Z., and Shen, Y. L2-gcn: Layer-wise and learned efficient training of graph convolutional networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), pp. 2127–2135.

[75] Zhang, X., Li, F., Zhang, Z., Li, Q., Wang, C., and Wu, J. Enabling execution assurance of federated learning at untrusted participants. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications* (2020), IEEE, pp. 1877–1886.

[76] Zhao, B. Z. H., Kaafar, M. A., and Kourtellis, N. Not one but many tradeoffs: Privacy vs. utility in differentially private machine learning. In *Cloud Computing Security Workshop* (2020), ACM CCS.

[77] Zhao, S., Zhang, Q., Qin, Y., Feng, W., and Feng, D. Sectee: A software-based approach to secure enclave architecture using tee. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (2019), pp. 1723–1740.

[78] Zhu, L., Liu, Z., and Han, S. Deep leakage from gradients. In *Advances in Neural Information Processing Systems* (2019), pp. 14774–14784.

A APPENDIX

A.1 Transferring Public Datasets

The server can potentially gather data that have a similar distribution to clients' private data. In initialization, the server trains a global model based on the gathered data rather than using one existing model. Then, the server broadcasts the trained model to clients' devices. Clients feed their private data into the model but update only the last layers inside the TEE during local training. Also, only the last layers being trained are uploaded to the server for secure aggregation. Because the server holds public data, we expect it to retrain the complete model before each communication round in order to keep fine-tuning the first layers. Here, we fix the communication rounds to 20 and measure only the test accuracy. We expect the system cost to be similar to transferring from models because, similarly, only the last layers are trained at the client-side.

(a) Transfer from public MNIST, to train LeNet.

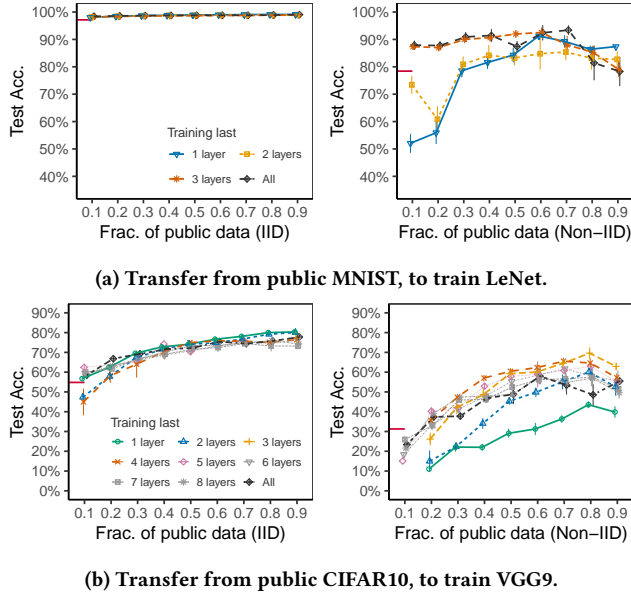(b) Transfer from public CIFAR10, to train VGG9.

Figure 8: Test accuracy when learning with public datasets. The short red line (—) starting from y-axis refers to end-to-end FL. Each trail runs for 10 times, and error bars refer to 95% confidence interval (Note: In the top left figure, test accuracy is very high and almost the same, as the range of y-axis is set as the same for the same dataset (i.e., MNIST here). In the bottom right figure (i.e., for CIFAR10), several trails fail to train and thus corresponding points are not plotted).

Test accuracy results are shown in Figure 8. It is indicated that in general when the server holds more public data, the final global model can reach a higher test accuracy. This is as expected since the server gathers a larger part of the training datasets. With complete training datasets, this process will finally become centralized training. Nevertheless, this indication is not always held. For example, in the IID case (see the two left plots in Figure 8), when training all layers, servers with public data of 0.1 fraction outperform servers without public data, i.e., the end-to-end FL, while regarding Non-IID of CIFAR10, servers with 0.1 fraction cannot outperform that without public data (see right plots in Figure 8b). One reason for it is that the first layers, which are trained on public datasets, cannot represent all features of privacy datasets. We also observe that when the server does not have enough public data (e.g., 0.1 fraction), training only the last 1 or 2 layers can lead to extremely low performance or even failure. Still, this is because the first layers cannot represent sufficiently the clients' datasets.

Another observation is that the number of training last layers does not have a significant influence on test accuracy in terms of IID cases, especially when the server holds more public data. This is because learning from IID public data is able to represent the feature space of the complete (private) training datasets. However, the results change when it comes to the Non-IID case. The number of training last layers has a significant influence on test accuracy. For instance, regarding VGG9, training only the last 1 or 2 layers at the client-side performs much worse compared to training 3 or 4 layers (see right plots in Figure 8b). Moreover, training 3 or 4 layers tend to have better test accuracy than training more layers (e.g., all layers). One explanation is that the feature extraction capability of first layers is good enough when the server has many public data, so fine-tuning these first layers at the client (e.g., training all layers) may destroy the model and consequently drop the accuracy.

Overall, by training only the last several layers at the client-side, PPFL with public datasets can guarantee privacy, and in the meanwhile, achieve better performance than that of training all layers.