SplitFed: When Federated Learning Meets Split Learning

Chandra Thapa[1*], **Pathum Chamikara Mahawaga Arachchige**[1], **Seyit Camtepe**[1], **Lichao Sun**[2*]

[1]CSIRO Data61, Sydney, Australia
[2]Lehigh University, Bethlehem, Pennsylvania, USA
{chandra.thapa, chamikara.arachchige, seyit.camtepe}@data61.csiro.au, lis221@lehigh.edu

Abstract

Federated learning (FL) and split learning (SL) are two popular distributed machine learning approaches. Both follow a model-to-data scenario; clients train and test machine learning models without sharing raw data. SL provides better model privacy than FL due to the machine learning model architecture split between clients and the server. Moreover, the split model makes SL a better option for resource-constrained environments. However, SL performs slower than FL due to the relay-based training across multiple clients. In this regard, this paper presents a novel approach, named *splitfed learning (SFL)*, that amalgamates the two approaches eliminating their inherent drawbacks, along with a refined architectural configuration incorporating differential privacy and PixelDP to enhance data privacy and model robustness. Our analysis and empirical results demonstrate that (pure) SFL provides similar test accuracy and communication efficiency as SL while significantly decreasing its computation time per global epoch than in SL for multiple clients. Furthermore, as in SL, its communication efficiency over FL improves with the number of clients. Besides, the performance of SFL with privacy and robustness measures is further evaluated under extended experimental settings.

Introduction

Distributed Collaborative Machine Learning (DCML) is popular due to its default data privacy benefits (Kairouz, McMahan, and et al. 2019). Unlike the conventional approach, where the data is centrally pooled and accessed, DCML enables machine learning without having to transfer data from data custodians to any untrusted party. Moreover, analysts have no access to raw data; instead, the machine learning (ML) model is transferred to the data curator for processing. Besides, it enables computation on multiple systems or servers and distributed devices.

The most popular DCML approaches are federated learning (Konecný, McMahan, and Ramage 2015; McMahan et al. 2017) and split learning (Gupta and Raskar 2018). Federated learning (FL) trains a full ML model on the distributed clients with their local data and later aggregates the locally trained full ML models to form a global model in a server. The main advantage of FL is that it allows parallel, hence efficient, ML model training across many clients.

Computational requirement at the client-side and model privacy during ML training in FL. The main disadvantage of FL is that each client needs to run the full ML model, and resource-constrained clients, such as available in the Internet of Things, could not afford to run the full model. This case is prevalent if the ML models are deep learning models. Besides, there is a privacy concern from the model's privacy perspective during training because the server and clients have full access to the local and global models.

To address these concerns, split learning (SL) was introduced. SL splits the full ML model into multiple smaller network portions and train them separately on a server, and distributed clients with their local data. Assigning only a part of the network to train at the client-side reduces processing load (compared to that of running a complete network as in FL), which is significant in ML computation on resource-constrained devices (Vepakomma et al. 2018). Besides, a client has no access to the server-side model and vice-versa.

Training time overhead in SL. Despite the advantages of SL, there is a primary issue. The relay-based training in SL makes the clients' resources idle because only one client engages with the server at one instance; causing a significant increase in the training overhead with many clients.

To address these issues in FL and SL, this paper proposes a novel architecture called *splitfed learning (SFL)*. SFL considers the advantages of FL and SL, while emphasizing on data privacy, and robustness of the model. Refer to Table 1 for its abstract comparison with FL and SL. Our contributions are mainly two-fold: Firstly, we are the first to propose SFL. Data privacy and model's robustness are enhanced at the architectural level in SFL by the differential privacy-based measures (Abadi et al. 2016) and PixelDP (Lecuyer et al. 2019). Secondly, to demonstrate the feasibility of SFL, we present comparative performance measurements of FL, SL, and SFL by considering four standard datasets and four popular models. Based on our analyses and empirical results, SFL provides an excellent solution that offers better model privacy than FL, and it is faster than SL with a similar performance to SL in model accuracy and communication efficiency.

Overall, SFL is beneficial for resource-constrained environments where full model training and deployment are not

*Corresponding author.

	SL	FL	SFL
Model aggregation	No	Yes	Yes
Model privacy advantage by splitted model	Yes	No	Yes
Client-side training	Sequential	Parallel	Parallel
Distributed computing	Yes	Yes	Yes
Access to raw data	No	No	No

Table 1: An abstract comparison of split learning (SL), federated learning (FL), and splitfed learning (SFL).

feasible, and fast model training time is required to periodically update the global model based on a continually updating dataset over time (*e.g.*, data stream). These environments characterize various domains, including health, *e.g.*, real-time anomaly detection in a network with multiple Internet of Medical Things[1] connected via gateways, and finance, *e.g.*, privacy-preserving credit card fraud detection.

Background and Related Works

Federated learning (Konecný, McMahan, and Ramage 2015; McMahan et al. 2017; Bonawitz et al. 2019) trains a complete ML network/algorithm at each client on its local data in parallel for a certain number of local epochs, and then the local updates are sent to the server for aggregation (McMahan et al. 2017). This way, the server forms a global model and completes one global epoch[2]. The learned parameters of the global model are then sent back to all clients to train for the next round. This process continues until the algorithm converges. In this paper, we consider the federated averaging (FedAvg) algorithm (McMahan et al. 2017) for model aggregations in FL. FedAvg considers a weighted average of the gradients for the model updates.

Split learning (Vepakomma et al. 2018; Gupta and Raskar 2018) splits a deep learning network $\mathbf{W}$ into multiple portions, and these portions are processed and computed on different devices. In a simple setting, $\mathbf{W}$ is split into two portions $\mathbf{W}^{\mathrm{C}}$ and $\mathbf{W}^{\mathrm{S}}$, called client-side network and server-side network, respectively. The clients, where the data reside, commit only to the client-side portion of the network, and the server commits only to the server-side portion of the network. The communication involves sending activations, called *smashed data*, of the split layer, called *cut layer*, of the client-side network to the server, and receiving the gradients of the smashed data from the server-side operations. The synchronization of the learning process with multiple clients is done either in a centralized mode or peer-to-peer mode in SL (Gupta and Raskar 2018).

[1]The examples of the Internet of Medical Things include glucose monitoring devices, open artificial pancreas systems, wearable electrocardiogram (ECG) monitoring devices, and smart lenses.

[2]When forward propagation and back-propagation are completed for all available datasets across all participating clients for one cycle, it is called one global epoch.

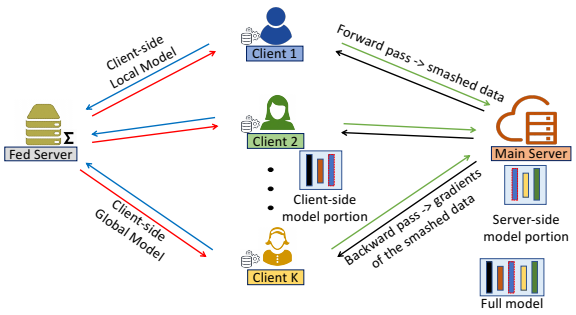


Figure 1: Overview of splitfed learning (SFL) system model.

Differential privacy (DP) is a privacy model that defines privacy in terms of stochastic frameworks (Dwork and Roth 2014; Dwork et al. 2016). DP is formally defined as follows:

Definition 1 *A mechanism $\mathcal{M}$ is considered to be (ϵ, δ)-differential private if, for all adjacent datasets, x and y, and for all possible subsets of results, R of the mechanism, the following holds:*

$$\mathbb{P}[\mathcal{M}(x) \in R] \leq e^{\epsilon} * \mathbb{P}[\mathcal{M}(y) \in R] + \delta.$$

Practically, the values of ϵ (privacy budget) and δ (probability of failure) should be kept as small as possible to maintain a high level of privacy. However, the smaller the values of ϵ and δ, the higher the noise applied to the input data by the DP algorithm.

The Proposed Framework

The framework SFL is presented in this section. We first give the overview of SFL. Then we detail three key modules: (1) the differentially private knowledge perturbation, (2) the PixelDP for robust learning, and (3) total cost analysis of SFL.

Overall Structure

SFL combines the primary strength of FL, which is parallel processing among distributed clients, and the primary strength of SL, which is network splitting into client-side and server-side sub-networks during training. Refer to Figure 1 for a representation of the SFL architecture. Unlike SL, all clients carry out their computations in parallel and engage with the main server and fed server. A client can be a hospital or an Internet of Medical Things with low computing resources, and the main server can be a cloud server or a researcher with high-performance computing resources. The fed server is introduced to conduct FedAvg on the client-side local updates. Moreover, the fed server synchronizes the client-side global model in each round of network training. The fed server's computations, which is mainly computing FedAvg, are not costly. Hence, the fed server can be hosted within the local edge boundaries. Alternatively, if we implement all operations at the fed server over encrypted information, i.e., homomorphic encryption-based client-side model aggregation, then the main server can perform the operations of the fed server.

SFL workflow. All clients perform forward propagation on their client-side model in parallel, including its

Algorithm 1: Splitfed Learning (SFL)

Notations: (1) S_t is a set of K clients at t time instance, (2) $\mathbf{A}_{k,t}$ is the smashed data of client k at t, (3) $\mathbf{Y}_k$ and $\hat{\mathbf{Y}}_k$ are the true and predicted labels, respectively, of the client k, (4) $\triangledown \ell_k$ is the gradient of the loss for the client k, (5) n and n_k are the total sample size and sample size at a client k, respectively.

`/* Runs on Main Server */`
EnsureMainServer executes:
  if time instance *t=0* **then**
    Initialize $\mathbf{W}_t^{\mathrm{S}}$ (global server-side model)
  else
    for each client $k \in S_t$ in parallel **do**
      while *local epoch* $e \neq E$ **do**
        $(\mathbf{A}_{k,t}, \mathbf{Y}_k) \leftarrow$ ClientUpdate$(\mathbf{W}_{k,t}^{\mathrm{C}})$
        Forward propagation with $\mathbf{A}_{k,t}$ on $\mathbf{W}_t^{\mathrm{S}}$, compute $\hat{\mathbf{Y}}_k$
        Loss calculation with $\mathbf{Y}_k$ and $\hat{\mathbf{Y}}_k$
        Back-propagation calculate $\triangledown \ell_k(\mathbf{W}_t^{\mathrm{S}}; \mathbf{A}_t^{\mathrm{S}})$
        Send $d\mathbf{A}_{k,t} := \triangledown \ell_k(\mathbf{A}_t^{\mathrm{S}}; \mathbf{W}_t^{\mathrm{S}})$ (i.e., gradient of the $\mathbf{A}_{k,t}$) to client k for ClientBackprop$(d\mathbf{A}_{k,t})$
      end
    end
    Server-side model update:
    $\mathbf{W}_{t+1}^{\mathrm{S}} \leftarrow \mathbf{W}_t^{\mathrm{S}} - \eta \frac{n_k}{n} \sum_{i=1}^{K} \triangledown \ell_i(\mathbf{W}_t^{\mathrm{S}}; \mathbf{A}_t^{\mathrm{S}})$
  end

`/* Runs on Fed Server */`
EnsureFedServer executes:
  if *t=0* **then**
    Initialize $\mathbf{W}_t^{\mathrm{C}}$ (global client-side model)
    Send $\mathbf{W}_t^{\mathrm{C}}$ to all K clients for ClientUpdate$(\mathbf{W}_{k,t}^{\mathrm{C}})$
  else
    for each client $k \in S_t$ in parallel **do**
      $\mathbf{W}_{k,t}^{\mathrm{C}} \leftarrow$ ClientBackprop$(d\mathbf{A}_{k,t})$
    end
    Client-side global model updates:
    $\mathbf{W}_{t+1}^{\mathrm{C}} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} \mathbf{W}_{k,t}^{\mathrm{C}}$
    Send $\mathbf{W}_{t+1}^{\mathrm{C}}$ to all K clients for ClientUpdate$(\mathbf{W}_{k,t}^{\mathrm{C}})$
  end

noise layer, and pass their smashed data to the main server. Then the main server processes the forward propagation and back-propagation on its server-side model with each client's smashed data separately in (somewhat) parallel. It then sends the gradients of the smashed data to the respective clients for their back-propagation. Afterward, the server updates its model by FedAvg, i.e., weighted averaging of gradients that it computes during the back-propagation on each client's smashed data. At the client's side, after receiving the gradients of its smashed data, each client performs the back-propagation on their client-side local model and computes its gradients. A DP mechanism is used to make these gradients private and send them to the fed server. The fed server conducts the FedAvg of the client-side local updates and sends them back to all participating clients.

Variants of Splitfed Learning. There can be several variants of SFL. We broadly divide them into two categories in the following:

Based on Server-side Aggregation. This paper proposes two variants of SFL. The first one is called *splitfedv1 (SFLV1)*, which is depicted in Algorithm 1 and 2. The next algorithm is called *splitfedv2 (SFLV2)*, and it is motivated by the intuition of the possibility to increase the model accuracy by removing the model aggregation part in the server-side computation module in Algorithm 1. In Algorithm 1, the server-side models of all clients are executed separately in parallel and then aggregated to obtain the global server-side model at each global epoch. In contrast, SFLV2 processes the forward-backward propagations of the server-side model sequentially with respect to the client's smashed data (no FedAvg of the server-side models). The client order is chosen randomly in the server-side operations, and the model gets updated in every single forward-backward propagation. Besides, the server receives the smashed data from all participating clients synchronously. The client-side operation remains the same as in the SFLV1; the fed server conducts the FedAvg of the client-side local models and sends the aggregated model back to all participating clients. These operations are not affected by the client order as the local client-side models are aggregated by the weighted averaging method, i.e., FedAvg. Some other SFL versions are available in the literature, but they are developed after and influenced by our approach (Han, amd Jungmoon Lee, and Moon 2021; Gao et al. 2021).

Based on Data Label Sharing. Due to the split ML models in SFL, we can carry out ML in the two settings; (1) sharing the data labels to the server and (2) without sharing any data labels to the server. Algorithm 1 considers SFL with data label sharing. In cases without sharing data labels, the ML model in SFL can be partitioned into three parts, assuming a simple setup. Each client will process two client-side model portions; one with the first few layers of $\mathbf{W}$, and another with the last few layers of $\mathbf{W}$ and loss calculations. The remaining middle layers of $\mathbf{W}$ will be computed at the server-side. All possible configurations of SL, including vertically partitioned data, extended vanilla, and multi-task SL (Vepakomma et al. 2018), can be carried out similarly in SFL as its variants.

Privacy Protection

The inherent privacy preservation capabilities of SFL are due to two reasons: firstly, it adopts the model-to-data approach, and secondly, SFL conducts ML over a split network. A network split in ML learning enables the clients/fed server and the main server to maintain the full model privacy by not allowing the main server to get the client-side model updates and vice-versa. The main server has access only to the smashed data (i.e., activation vectors of the cut layer). The curious main server needs to invert all the client-side model parameters, i.e., weight vectors, to infer data and client-side model. The possibility of inferring the client-side model parameters and raw data is highly unlikely if we configure the client-side ML networks' fully connected

Algorithm 2: ClientUpdate

```
/* Runs on Client k */
EnsureClientUpdate(W^C_{k,t}):
    Model updates W^C_{k,t} ← FedServer()
    Set A_{k,t} = φ
    for each local epoch e from 1 to E do
        Forward propagation with data X_k up to a layer
          L ≥ 1 in W^C_{k,t}
        Noise layer: Perturbs the outputs of the layer L
          based on Equation (5)
        With the output from the noise layer, continue forward
        propagation to the remaining layers of W^C_{k,t}, and get
        the activations of its final layer A_{k,t} (smashed data)
        Y_k is the true labels of X_k
        Send A_{k,t} and Y_k to the main server
        Wait for the completion of ClientBackprop(dA_{k,t})
    end

/* Runs on Client k */
EnsureClientBackprop(dA_{k,t}):
    while local epoch e ≠ E do
        dA_{k,t} ← MainServer()
        Back-propagation, calculate gradients ▽ℓ_k(W^C_{k,t})
          with dA_{k,t}
        ℓ2-norm of each gradient is clipped and a
          calibrated noise is added based on Equation (2)
          and (3) to calculate g̃_{k,t}
        Update W^C_{k,t} ← W^C_{k,t} − η g̃_{k,t}
    end
    Send W^C_{k,t} to the Fed server
```

layers with sufficiently large numbers of nodes (Gupta and Raskar 2018). However, for a smaller client-side network, the possibility of this issue can be high. This issue can be controlled by modifying the loss function at the client-side (Vepakomma et al. 2019). Due to the same reasons, the clients (having access only to the gradients of the smashed data from the main server) and the fed server (having access only to the client-side updates) cannot infer the server-side model parameters. Since there is no network split and separate training on the client-side and server-side in FL, SFL provides superior architectural configurations for enhanced privacy for an ML model during training compared to FL.

Privacy Protection at the Client-side. We discuss the inherent privacy of the proposed model in the previous section. However, there can be an advanced adversary exploiting the underlying information representations of the shared smashed data or parameters (weights) to violate data owners' privacy. This can happen if any server/client becomes curious though still honest. To avoid these possibilities, we apply two measures in our studies; (i) differential privacy to the client-side model training and (ii) PixelDP noise layer in the client-side model.

Privacy Protection on Fed Server. Considering Algorithm 2, we present the process for implementing differential privacy at a client k. We assume the following: σ represents the noise scale, and C' represents the gradient norm bound. Now, firstly, after t time, the client k receives the gradients $d\mathbf{A}_{k,t}$ from the server, and with this, it calculates client-side gradients $\triangledown\ell_k(\mathbf{W}^{\mathrm{C}}_{k,i,t})$ for each of its local sample x_i, and

$$\mathbf{g}_{k,t}\left(x_i\right) \leftarrow \triangledown\ell_k(\mathbf{W}^{\mathrm{C}}_{k,i,t}). \tag{1}$$

Secondly, the ℓ_2-norm of each gradient is clipped according to the following equation:

$$\overline{\mathbf{g}}_{k,t}\left(x_i\right) \leftarrow \mathbf{g}_{k,t}\left(x_i\right) / \max\left(1, \frac{\left\|\mathbf{g}_{k,t}\left(x_i\right)\right\|_2}{C'}\right). \tag{2}$$

Thirdly, calibrated noise is added to the average gradient:

$$\tilde{\mathbf{g}}_{k,t} \leftarrow \frac{1}{n_k}\sum_i \left(\overline{\mathbf{g}}_{k,t}\left(x_i\right) + \mathcal{N}\left(0, \sigma^2 C'^2 \mathbf{I}\right)\right). \tag{3}$$

Finally, the client-side model parameters of client k are updated as follows; $\mathbf{W}^{\mathrm{C}}_{k,t+1} \leftarrow \mathbf{W}^{\mathrm{C}}_{k,t} - \eta_t \tilde{\mathbf{g}}_{k,t}$.

We apply calibrated noise iteratively until the model converges or reaches a specified number of iterations. As the iterations progress, the final convergence will exhibit a privacy level of (ε, δ)- differential privacy, where (ε, δ) is the overall privacy cost of the client-side model.

Differential privacy is used to enforce strict privacy to the client-side model training algorithm based on Abadi et al.'s approach (Abadi et al. 2016). Equation 2 (norm clipping) guarantees that $\left\|\mathbf{g}_{k,t}\left(x_i\right)\right\|_2$ is preserved when $\left\|\mathbf{g}_{k,t}\left(x_i\right)\right\|_2 \leq C'$. This step also guarantees that $\left\|\mathbf{g}_{k,t}\left(x_i\right)\right\|_2$ scaled down to C' when $\left\|\mathbf{g}_{k,t}\left(x_i\right)\right\|_2 > C'$. This step also helps clipping out the effect of Equation 5 on the gradients. Hence, norm clipping step allows bounding the influence of each individual example on $\mathbf{g}_{k,t}$ in the process of guaranteeing differential privacy. It was shown that, the corresponding noise addition (refer to Equation 3) provides (ϵ, δ)-DP for each step of b ($b = n_k/batch\_size$), if we choose σ (noise scale) to be $\sqrt{2\log\frac{1.25}{\delta}}/\varepsilon$ (Dwork and Roth 2014). Hence, at the end of b steps, this will result in $(b\epsilon, b\delta)$-DP. As shown by Abadi et al., for any $\varepsilon < c_1 b^2 T$ and $\delta > 0$, by choosing $\sigma \geq c_2 \frac{b\sqrt{T\log(1/\delta)}}{\varepsilon}$, the privacy can be maintained at (ϵ, δ)-DP (Abadi et al. 2016). Moments accountant (a privacy accountant) is used to track and maintain (ϵ, δ). Hence, at the end of b, a client model guarantees (ϵ, δ)-DP. With the strict assumption that all clients work on IID data, we can confirm that all clients maintain and guarantee (ϵ, δ)-DP while client-side model training and synchronization.

Privacy Protection on Main Server. The above DP measures do not stop potential leakage from the smashed data to the main server though it has some effect on the smashed data after the first global epoch. Thus, to avoid privacy leakage and further strengthen data privacy and model robustness against potential adversarial ML settings, we integrate a noise layer in the client-side model based on the concepts of PixelDP (Lecuyer et al. 2019).

This extended measure utilizes the noise application mechanism involved in differential privacy to add a calibrated noise to the output (*e.g.,* activation vectors) of a layer at the client-side model while maintaining utility. In this process, firstly, we calculate the sensitivity of the process. The sensitivity of a function $\mathbf{A}$ is defined as the maximum

change in output that can be produced by a change in the input, given some distance metrics for the input and output (p-norm and q-norm, respectively):

$$\Delta I_{p,q} = \Delta I^{\mathbf{A}}_{p,q} = \max_{i,j,i\neq j} \frac{\|\mathbf{A}_{k,i} - \min \mathbf{A}_{k,j}\|_q}{\|x_i - x_k\|_p} \quad (4)$$

Secondly, Laplacian noise with scale $\frac{\Delta I^{\mathbf{A}}_{p,q}}{\varepsilon'}$ is applied to randomize any data as follows:

$$\mathbf{A}^{\mathrm{P}}_{k,i} = \mathbf{A}_{k,i} + \mathrm{Lap}\left(\frac{\Delta I^{A}_{p,q}}{\varepsilon'}\right), \quad (5)$$

where, $\mathbf{A}^{\mathrm{P}}_{k,i}$ represents a private version of $\mathbf{A}_{k,i}$, and ϵ' is the privacy budget used for the Laplacian noise. This method enables forwarding private versions of the smashed data to the main server; hence, preserving the privacy of smashed data. The private version of the smashed data is due to the post-processing immunity of the DP mechanism applied at the noise layer in the client-side model. The noisy smashed data is more private than the original data due to the calibrated noise. Moreover, PixelDP not only can provide privacy for smashed data, but also can improve the robustness of the model against adversarial examples. However, detailed analysis and mathematical guarantees are kept for future work to preserve the main focus of the proposed work.

Robustness via PixelDP. The primary intuition behind using random DP mechanism to robust ML against adversarial examples is to create a DP scoring function. For example, feeding any data sample through the DP scoring function, the outputs are DP with regards to the features of the input. Then, stability bounds for the expected output of the DP function are given by the following Lemma (Lecuyer et al. 2019):

Lemma 1 *Suppose a randomized function $\mathcal{M}$, with bounded output $\mathcal{M} \in [0, b], b \in \mathbb{R}^+$, satisfies (ϵ, δ)-DP. Then the expected value of its output meets the following property:*

$$\forall \alpha \in B_p(1). \mathbb{E}(\mathcal{M}(x)) \leq e^{\epsilon} \cdot \mathbb{E}(\mathcal{M}(x+\alpha)) + b\delta, \quad (6)$$

where $B_p(r) := \alpha \in \mathbb{R}^n : \|\alpha\|_p \leq r$ is the p-norm ball, and the expectation is taken over the randomness in $\mathcal{M}$.

Combined with Equation, $\forall \alpha \in B_p(L), k = f(x).\ y_k(x+\alpha) > \max_{i:i\neq k} y_i(x+\alpha)$, the bounds provide a rigorous certification for robustness to adversarial examples.

Total Cost Analysis

This section analyzes the total communication cost and model training time for FL, SL, and SFL under a uniform data distribution. Assume K be the number of clients, p be the total data size, q be the size of the smashed layer, R be the communication rate, T be the time taken for one forward and backward propagation on the full model with dataset of size p (for any architecture), T_{fedavg} is the time required to perform the full model aggregation (let $\frac{T_{\text{fedavg}}}{2}$ be the aggregation time for an individual server), $|\mathbf{W}|$ be the size of the full model, and β be the fraction of the full model's size available in a client in SL/SFL, i.e., $|\mathbf{W}^{\mathrm{C}}| = \beta|\mathbf{W}|$. The term $2\beta|\mathbf{W}|$ in communication per client is due to the download and upload of the client-side model updates before and after training, respectively, by a client. The result is presented in Table 2. As shown in the table, SL can become inefficient when there is a large number of clients. Besides, we see that when K increases, the total training time cost increases in the order of SFLV2<SFLV1<SL. Also, we observe this in our empirical results[3].

Experiments

Experiments are carried out on uniformly distributed and horizontally partitioned image datasets among clients. All programs are written in python 3.7.2 using the PyTorch library (PyTorch 1.2.0). For quicker experiments and developments, we use the High-Performance Computing (HPC) platform that is built on Dell EMC's PowerEdge platform with partner GPUs for computation and InfiniBand networking. We run clients and servers on different computing nodes of the cluster provided by HPC. We request the following resources for one slurm job on HPC: 10GB of RAM, one GPU (Tesla P100-SXM2-16GB), one computing node with at most one task per node. The architecture of the nodes is x86_64. In our setup, we consider that all participants update the model in each global epoch (i.e., $C = 1$ during training). We choose ML network architectures and datasets based on their performance and their need to include proportionate participation in our studies. The learning rate for LeNet is maintained at 0.004 and 0.0001 for the remainder of network architectures (AlexNet, ResNet, and VGG16). We choose the learning rate based on the models' performance during our initial observations. For example, for LeNet on FMNIST, we observed train and test accuracy of 94.8% and 92.1% with a learning rate of 0.004, whereas 87.8% and 87.3% with a learning rate of 0.0001 in 200 global epochs. We set up a similar computing environment for our analysis.

We use four public image datasets in our experiments, and these are summarized in Table 3. HAM10000 dataset is a medical dataset, i.e., the Human Against Machine with 10000 training images (Tschandl 2018). It consists of colored images of pigmented skin lesions, and has dermatoscopic images from different populations, acquired and stored by different modalities. It has seven cases of important diagnostic categories of lesions: Akiec, bcc, bkl, df, mel, nv, and vasc. MNIST, Fashion MNIST, and CIFAR10 are standard datasets, all with 10 classes.

In regard to ML models, we consider four popular architectures in our experiments. These four architectures fall under Convolutional Neural Network (CNN) architectures and are summarized in Table 4. We restrict our experiments to CNN architectures to maintain the cohesiveness of our work proposed in this paper. We will conduct further experimental evaluations on other architectures such as recurrent neural networks in future work.

For all experiments in SL, SFLV1, and SFLV2, the network layers are split at the following layer: second layer of LeNet (after 2D MaxPool layer), second layer of AlexNet

[3]Empirical results are provided in (Thapa, Chamikara, and Camtepe 2020).

Method	Comms. per client	Total comms.	Total model training time
FL	$2\|\mathbf{W}\|$	$2K\|\mathbf{W}\|$	$\{T + 2\frac{\|\mathbf{W}\|}{R} + T_{\text{fedavg}}\}$
SL	$(\frac{2p}{K})q + 2\beta\|\mathbf{W}\|$	$2pq + 2\beta K\|\mathbf{W}\|$	$T + 2\frac{pq}{R} + 2\beta\frac{\|\mathbf{W}\|}{R}K$
SFLV1	$(\frac{2p}{K})q + 2\beta\|\mathbf{W}\|$	$2pq + 2\beta K\|\mathbf{W}\|$	$T + 2\frac{pq}{KR} + 2\frac{\beta\|\mathbf{W}\|}{R} + T_{\text{fedavg}}$
SFLV2	$(\frac{2p}{K})q + 2\beta\|\mathbf{W}\|$	$2pq + 2\beta K\|\mathbf{W}\|$	$T + 2\frac{pq}{KR} + 2\frac{\beta\|\mathbf{W}\|}{R} + \frac{T_{\text{fedavg}}}{2}$

Table 2: Total cost analysis of the four DCML approaches for one global epoch.

Dataset	Training samples	Testing samples	Image size
HAM10000 (Tschandl 2018)	9,013	1,002	600×450
MNIST (LeCun, Cortes, and Burges 2010)	60,000	10,000	28×28
FMNIST (Xiao, Rasul, and Vollgraf 2017)	60,000	10,000	28×28
CIFAR10 (Krizhevsky, Nair, and Hinton 2009)	50,000	10,000	32×32

Table 3: Datasets

(after 2D MaxPool layer), fourth layer of VGG16 (after 2D MaxPool layer), and third layer (after 2D BatchNormalization layer) of ResNet18. For a fair comparison, while performing the comparative evaluations of SFLV1 and SFLV2 with FL and SL, we do not consider the addition of differential privacy-based measures and PixelDP in SFLV1 and SFLV2[4].

Performance of FL, SL, SFLV1 and SFLV2

We consider the results under normal learning (centralized learning) as our benchmark. Table 5 summarizes our first result, where the observation window is 200 global epochs with one local epoch, batch size of 1024, and five clients for DCML. The tables show the best accuracy observed within 200 global epochs. Moreover, the test accuracy is averaged over all clients in the DCML setup at each global epoch.

As presented in Table 5, SL and SFL (both versions) performed well under the proposed experimental setup. However, we also observed that among DCML, FL shows better learning performance in most cases, possibly due to the FedAvg of the full models at each global epoch. Based on the results, we can observe that SFLV1 and SFLV2 have inherited the characteristics of SL. In a separate experiment, we noticed that VGG16 on CIFAR10 did not converge in SL, which was the same for both versions of splitfed learning, although there were around 66% and 67% of training and testing accuracies, respectively, for FL. We assume that this was because of the unavailability of certain other factors such as hyper-parameters tuning or change in data distribution or additional regularization terms in the loss function, which are beyond the scope of this paper.

Further diving into individual cases, as an example, we present the performance of ResNet18 on the HAM10000 dataset for normal (centralized learning), FL, SL, SFLV1, and SFLV2, under similar settings. For ResNet18 on HAM10000, the test accuracy convergence was almost the same for FL, SL, SFLV1, and SFLV2, and they reached around 76% in the observation window of 200 global epochs (refer to Figure 2). However, SFLV1 and SFLV2 struggled to converge if SL failed to converge. This was observed for the case of VGG16 on CIFAR10 in our separate experiments.

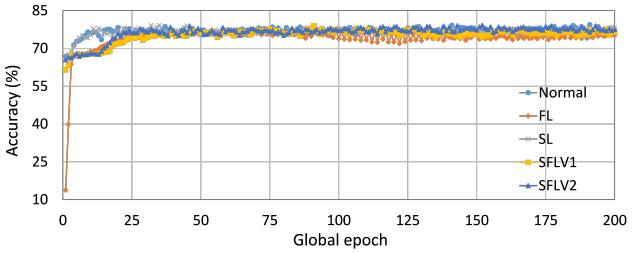


Figure 2: Testing convergence of ResNet18 on HAM10000 under various learning with five clients.

So far, we considered the testing mean accuracy in our results. Figure 4 illustrates the variations of the performance (i.e., accuracy) over five clients at each global epoch. In this regard, we compute the coefficient of variation (CV), which is a ratio of the standard deviation to the mean, and it measures the dispersion. Moreover, we calculate the CV over the five accuracies generated by the five clients at each global epoch. Based on our results for ResNet18 on HAM10000, the CVs for SL, FL, SFLV1, and SFLV2 are bounded between 0.06 and 2.63 while training, and 0.54 and 6.72 while testing after epoch 2; at epoch 1, the CV is slightly higher. The results indicate uniform individual client-level performance across the clients, as the CV coefficient values below 10 are considered a good range in literature.

In some datasets and architectures, the training/testing accuracy of the model was still improving and showing better performance at higher global epochs than 200. For ex-

[4]Some source codes are available at https://github.com/chandra2thapa/SplitFed-When-Federated-Learning-Meets-Split-Learning.

Architecture	# Parameters	Layers	Kernel size
LeNet (Lecun et al. 1998)	60 thousands	5	(5×5), (2×2)
AlexNet (Krizhevsky, Sutskever, and Hinton 2012)	60 million	8	(11×11),(5×5),(3×3)
VGG16 (Simonyan and Zisserman 2015)	138 million	16	(3×3)
ResNet18 (He et al. 2016)	11.7 million	18	(7×7), (3×3)

Table 4: Model Architecture

Dataset	Architecture	Normal	FL	SL	SFLV1	SFLV2
HAM10000	ResNet18	79.3%	77.5%	79.1%	79%	79.2%
HAM10000	AlexNet	80.1%	75 %	73.8%	70.5%	74.9%
FMNIST	LeNet	92.7%	91.9 %	90.4%	89.6%	90.4%
FMNIST	AlexNet	90.5%	89.7%	84.7%	86%	81%
CIFAR10	LeNet	72.1%	69.4 %	62.7%	62.6%	63.8%
MNIST	AlexNet	98.8%	98.7 %	95.1%	96.9%	92%
MNIST	ResNet18	99.3%	99.2 %	99.2%	99%	99.2%

Table 5: Test Results (five clients for DCML)

ample, going from 200 epochs to 400 epochs, we noticed training and testing accuracy increment from around 83% to around 86% for FL with LeNet on FMNIST with 100 users. However, we limited our observation window to 100 or 200 global epochs as some network architecture such as AlexNet on HAM10000 in FL was taking an extensive amount of training time on the HPC (a shared resource).

Effect of Number of Users on the Performance

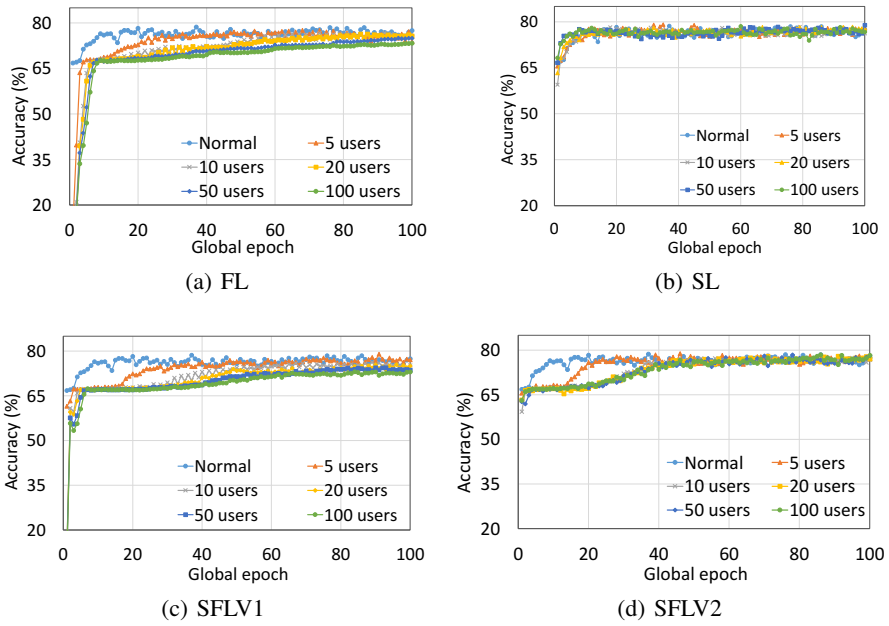
This section presents the analysis of the effect of the number of users for ResNet18 on HAM10000. We observed that up to 100 clients (clients ranging from 5 to 100), the training and testing curves for all numbers of clients followed a similar pattern in each plot. Moreover, they achieved a similar level of accuracy within each of our DCMLs. We got comparative test accuracies of 74% (FL), 77% (SL), 75% (SFLV1), and 77% (SFLV2) at 100 global epochs. While training, only SL and SFLV2 achieved the centralized training (normal learning) accuracy at around 100 global epochs. In contrast, FL and SFLV1 could not achieve this result even at 200 global epochs. The experimental results for clients ranging from 5 to 100 showed a negligible effect on the performance due to the increase in the number of clients in FL, SL, SFLV1, and SFLV2 (for example, refer to Figure 3. However, this observation was not the case in general. For LeNet on FMNIST with fewer clients, the testing performances of FL and SL were close to the normal learning. Moreover, for SL with AlexNet on HAM10000, the performance degraded and even failed to converge with the increase in the number of clients, and we saw a similar effect on the SFLV2. Overall, the convergence of the learning and performance slowed down (sometimes failed to progress) with the increase in the number of clients due to the resource limitations and other constraints, such as the change in data distribution among the clients with the increase in its number, and a regular global model aggregation to synchronize the model across the multiple clients.

SFL with Differential Privacy at the Client-side Model with a PixelDP Noise Layer

We implemented the differential privacy measures as described in Section "Privacy Protection." For illustration, experiments were performed for SFLV1 with AlexNet on MNIST data distributed over five clients. For 50 global epochs with 5 local epochs at each client per global epoch, the testing accuracy curves converged as shown in Figure 5. Besides, as for illustration, we change the values of ε', which is the privacy budget used by the PixelDP noise layer placed after the first convolution layer of AlexNet, to see the effect on the overall performance. Moreover, we maintain ε at 0.5 (privacy budget of client-side model training) during all experiments to examine the behavior of SFLV1 under strict client-side model privacy. As expected, the convergence of accuracy curves with DP measures is gradual and slow compared to non-differentially private training. Besides, testing accuracy of around 40%, 64%, 73%, 77%, and 78% are observed at global epoch 50 for ε' equal to $0.5, 1, 2, 5$, and no PixelDP, respectively. Clearly, the accuracy increases with the increase in the privacy budget, i.e., $\varepsilon + \varepsilon'$. Overall, the utility is decreased with a decrease in the privacy budget. As the client-side architecture in SFLV2 is the same as in SFLV1, the application of differential privacy in SFLV2 can be done in the same way as in SFLV1.

Conclusion

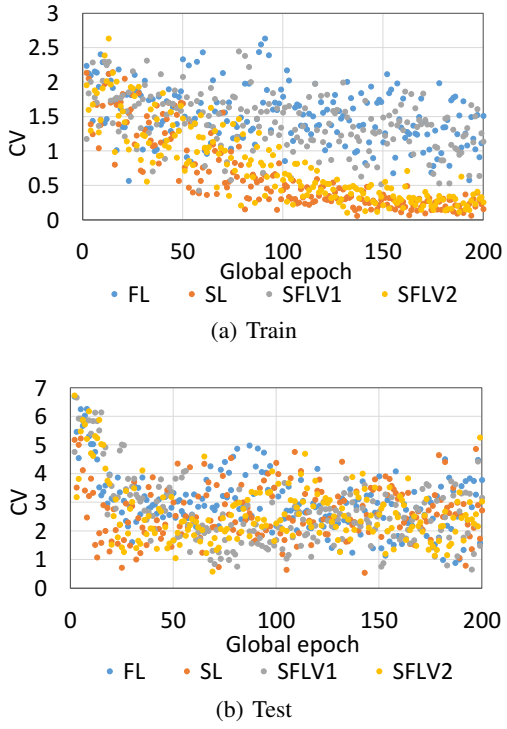
By bringing federated learning (FL) and split learning (SL) together, we proposed a novel distributed machine learning approach, named splitfed learning (SFL). SFL offered model privacy by network splitting and differential private client-side model updates. It is faster than SL by performing parallel processing across clients. Our results demonstrate that SFL provides similar performance in terms of model accuracy compared to SL. Thus, being a hybrid approach, it supports machine learning with resource-constrained de-



(a) FL (b) SL

(c) SFLV1 (d) SFLV2

Figure 3: Effect of the number of client/users on testing accuracy for ResNet18 on HAM10000.



(a) Train

(b) Test

Figure 4: Coefficient of variation (CV) of ResNet18 on HAM10000 under various learning settings with five clients.

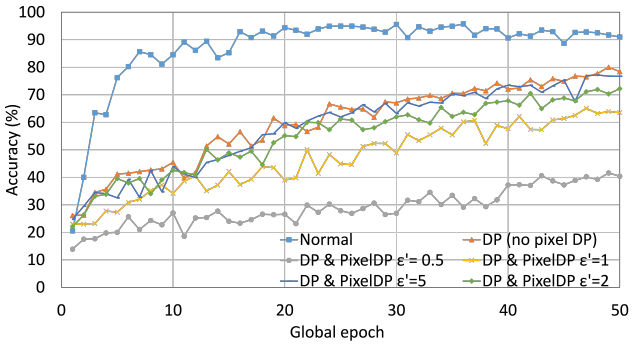


Figure 5: SFLV1 testing convergence of AlexNet on MNIST with five clients, sensitivity $\delta = 1e^{-5}$, $\varepsilon = 0.5$, $\sigma = 1.3$ (DP), and under various choices of ε' (PixelDP).

vices (enabled by network splitting as in SL) and fast training (enabled by handling clients in parallel as in FL). The performance of SFL with privacy and robustness measures based on differential privacy and PixelDP was further analyzed to investigate its feasibility towards data privacy and model robustness. Studies related to the detailed trade-off analysis of privacy and utility, and integration of homomorphic encryption (Gentry 2009) for guaranteed data privacy are left for future works.

References

Abadi, M.; Chu, A.; Goodfellow, I.; McMahan, H. B.; Mironov, I.; Talwar, K.; and Zhang, L. 2016. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Se-*

curity, 308–318.

Bonawitz, K.; Eichner, H.; Grieskamp, W.; Huba, D.; Ingerman, A.; Ivanov, V.; Kiddon, C.; Konecný, J.; Mazzocchi, S.; McMahan, H. B.; Overveldt, T. V.; Petrou, D.; Ramage, D.; and Roselander, J. 2019. Towards Federated Learning at Scale: System Design. In *Proc. SysML Conference*, 1–15.

Dwork, C.; McSherry, F.; Nissim, K.; and Smith, A. D. 2016. Calibrating Noise to Sensitivity in Private Data Analysis. *J. Priv. Confidentiality*, 7(3): 17–51.

Dwork, C.; and Roth, A. 2014. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4): 211–407.

Gao, Y.; Kim, M.; Thapa, C.; Abuadbba, S.; Zhang, Z.; Camtepe, S.; Kim, H.; and Nepal, S. 2021. Evaluation and Optimization of Distributed Machine Learning Techniques for Internet of Things. *CoRR*, abs/2103.02762.

Gentry, C. 2009. *A fully homomorphic encryption scheme*. Ph.D. thesis, Stanford University, Stanford, California.

Gupta, O.; and Raskar, R. 2018. Distributed learning of deep neural network over multiple agents. *J. Network and Computer Applications*, 116: 1–8.

Han, D.-J.; amd Jungmoon Lee, H. I. B.; and Moon, J. 2021. Accelerating Federated Learning with Split Learning on Locally Generated Losses. In *Proc. FL-ICML*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *Proc. IEEE CVPR*, 770–778.

Kairouz, P.; McMahan, H. B.; and et al. 2019. Advances and Open Problems in Federated Learning. *CoRR*, abs/1912.04977.

Konecný, J.; McMahan, B.; and Ramage, D. 2015. Federated Optimization: Distributed Optimization Beyond the Datacenter. *CoRR*, abs/1511.03575.

Krizhevsky, A.; Nair, V.; and Hinton, G. 2009. CIFAR-10 (Canadian Institute for Advanced Research). Http://www.cs.toronto.edu/ kriz/cifar.html.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Proc. NIPS'12 - Vol. 1*, 1097–1105. USA.

Lecun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proc. of the IEEE*, 86(11): 2278–2324.

LeCun, Y.; Cortes, C.; and Burges, C. 2010. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2.

Lecuyer, M.; Atlidakis, V.; Geambasu, R.; Hsu, D.; and Jana, S. 2019. Certified robustness to adversarial examples with differential privacy. In *2019 IEEE Symposium on Security and Privacy (SP)*, 656–672. IEEE.

McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; and y Arcas, B. A. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proc. AISTATS*, 1273–1282.

Simonyan, K.; and Zisserman, A. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *Proc. 3rd ICLR*.

Thapa, C.; Chamikara, M. A. P.; and Camtepe, S. 2020. SplitFed: When Federated Learning Meets Split Learning. *CoRR*, abs/2004.12088.

Tschandl, P. 2018. The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. *Harvard Dataverse*. Doi:10.7910/DVN/DBW86T.

Vepakomma, P.; Gupta, O.; Dubey, A.; and Raskar, R. 2019. Reducing leakage in distributed deep learning for sensitive health data. In *Proc. ICLR AI for social good workshop*.

Vepakomma, P.; Gupta, O.; Swedish, T.; and Raskar, R. 2018. Split learning for health: Distributed deep learning without sharing raw patient data. *arxiv*. Http://arxiv.org/abs/1812.00564.

Xiao, H.; Rasul, K.; and Vollgraf, R. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *CoRR*, abs/1708.07747.