Privacy and Efficiency of Communications in Federated Split Learning

Zongshun Zhang, Andrea Pinto, Valeria Turina, Flavio Esposito, and Ibrahim Matta, *Senior Member, IEEE*

***Abstract*—Every day, large amounts of sensitive data are distributed across mobile phones, wearable devices, and other sensors. Traditionally, these enormous datasets have been processed on a single system, with complex models being trained to make valuable predictions. Distributed machine learning techniques such as Federated and Split Learning have recently been developed to protect user data and privacy better while ensuring high performance. Both of these distributed learning architectures have advantages and disadvantages. In this article, we examine these tradeoffs and suggest a new hybrid Federated Split Learning architecture that combines the efficiency and privacy benefits of both. Our evaluation demonstrates how our hybrid Federated Split Learning approach can lower the amount of processing power required by each client running a distributed learning system, and reduce training and inference time while keeping a similar accuracy. We also discuss the resiliency of our approach to deep learning privacy inference attacks and compare our solution to other recently proposed benchmarks.**

***Index Terms*—Data communications, federated learning, machine learning, privacy, split learning.**

I. Introduction

CENTRALIZED machine learning (ML) training is becoming unsustainable [1]. Aside from the advantages of re-training often to optimize revenues [2], several learning applications need to run their processes at the edge of the network, not in the core of a data center, for multiple reasons, including end-to-end latency minimization by running machine learning algorithms locally on an end-device, and privacy concerns of trusting third-party clouds [3]. Several Machine Learning (ML) models trade user experience improvements on mobile devices for sensible data exploitation; see e.g., text recommendation in keyboards [4], [5] or vocal assistants [6]. In these and other applications, a decentralized learning approach may be preferable to a centralized system since sensitive data may remain locally within a client and not transferred over a computer network.

Despite its benefits in several use cases, running machine learning training and inference jobs within local devices has several limitations: computing capacity is often limited, battery drains faster with intensive processing, and the mobile or other end devices have limited memory and storage capabilities. For example, our experiments show that to fine-tune a VGG-16 [7] Neural Network, pre-trained on ImageNet [8] with Cifar-10 [9], tens of minutes are needed to reach 90% accuracy on an NVIDIA V100 GPU.

Different distributed neural network architectures have been proposed to preserve privacy and guarantee timely convergence – for example, Federated Learning (FL) [1], [10], [11], [12], Split Learning (SL) [13], or hybrid approaches [14], [15], [16]. FL, [10], averages the weights of the learned Neural Network model on each edge device to create a single model, and updates the local ones. Previous research has shown that this strategy can achieve higher accuracy than considering only a local model [10], [17], [18], [19] and at the same time can preserve the privacy of the data. SL splits the entire NN into partitions of layers. Then each partition is executed on a different entity (i.e., edge and cloud), and all edge NN partitions can sequentially pair with one cloud partition. Thus, this approach utilizes distributed datasets while keeping user data private.

On the one hand, given enough resources to meet training Service Level Objectives (SLOs), FL can scale to many devices. However, it is impractical in resource-constrained edge training/inference settings. On the other hand, Split Learning can train with limited resources, but it does not scale to many devices well since it is not parallel. Especially, when we pair different edge devices holding non-independent and identically distributed (Non-IID) data with the cloud, training may not converge at all [20]. Furthermore, the intermediate data in between partitions can be costly to transmit and store [21], [22], [23], [24], and leads to privacy concerns, as it is derived from the source data.

To cope with the inefficiencies of the existing distributed learning systems, we propose Federated Split Learning (FSL) [25], which combines the benefits of FL and SL while mitigating their drawbacks. The FSL model is characterized by multiple edge client – server pairs. Such pairs train their copies of the NN simultaneously, providing the parallelism of federated learning and the practicality of split learning by partitioning the NN across the client and (edge) server. After some client-edge server pairs have completed certain training epochs, a "Parameter Server" in the cloud only averages the (edge) server NN weights Fig. 1(e), an approach adapted from FedAVG in FL [10].

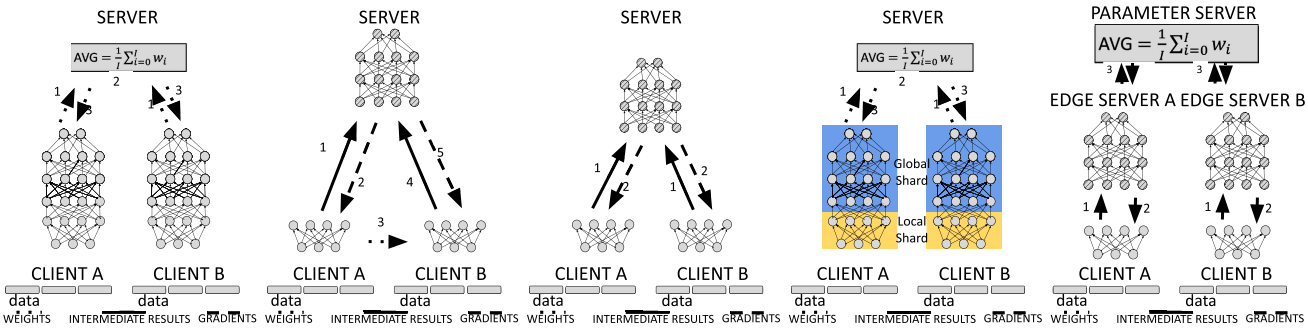
FSL is generally better than the similar techniques, i.e., Parallel Split Learning (PSL [16] or SplitFed [15]), and Federated

Manuscript received 1 February 2023; revised 14 May 2023; accepted 15 May 2023. Date of publication 29 May 2023; date of current version 1 September 2023. This work has been supported by National Science Foundation under Grants CNS-1908574 and CNS-1908677. Recommended for acceptance by S. Mohanty. *(Corresponding author: Zongshun Zhang.)*

Zongshun Zhang and Ibrahim Matta are with the Department of Computer Science, Boston University, Boston, MA 02215 USA (e-mail: zhangzs@bu.edu; matta@bu.edu).

Andrea Pinto, Valeria Turina, and Flavio Esposito are with the Department of Computer Science, Saint Louis University, Saint Louis, MO 63103 USA (e-mail: andrea.pinto.1@slu.edu; valeria.turina@slu.edu; espositof@slu.edu).

Digital Object Identifier 10.1109/TBDATA.2023.3280405



(a) Federated Learning (FL) (b) Split Learning (SL) (c) Parallel Split Learning (PSL) (d) Federated Reconstruction (FRC) (e) Federated Split Learning (FSL)

Fig. 1. Distributed NN Training Architectures: *(a) Federated Learning:* The NN is in the client. The parameter server calculates the average weights among clients and overrides the local weights. *(b) Split Learning:* The server partition sequentially trains with each of the clients. Client weights are shared with the next training client. *(c) Parallel Split Learning:* The server trains clients' output in batches in parallel, but the client's weights are kept private. *(d) Federated Reconstruction:* Multiple clients train local and global shards of weights alternatively in parallel and only the global shards are preserved and averaged in between epochs. *(e) Federated Split Learning:* Multiple Edge Server and Client pairs train simultaneously. The Edge Servers' weights are averaged by a Parameter Server. The clients weights are kept private.

Reconstruction (FRC) [26], in latency and privacy. Intuitively, while PSL only use one cloud server node, FSL allows parallel server computations leading to lower latency. Our naïve FSL also halves the processing time of FRC, though FRC has a better privacy level (Section IV-A). FSL, PSL, and other SL-based systems are prone to privacy concerns as they transmit the hidden variables (intermediate data) extracted based on client source data from the client to the (edge) server over the Internet. To mitigate this limitation, we further proposed a Client-based Privacy Approach (CPA [25]) to reduce the vulnerable features in intermediate data, e.g., by avoiding having client layers extract unnecessary features, or by training using privacy-preserving client weights. In this article, we further optimize this CPA approach to reduce overhead by considering the training pipeline and model partitioning plan.

Contributions: In this article, we evaluate the generality and benefits of our FSL architecture with different NN models and tasks: from image classifications to an Internet traffic classification [27]. In particular, we evaluate the privacy-performance tradeoff and discuss insights to enhance the privacy guarantee of FSL using an improved Client-based Privacy Approach (CPA) and novel neural network partitioning approaches. Particularly, we realized that certain ways of partitioning NN could reduce transmission delay and enhance privacy together. Our experimental results show that by combining different privacy approaches and NN partitioning methods, our FSL can achieve low training time, high privacy guarantees, and high accuracy.

The rest of the paper is organized as follows: Section II, introduces the background and related works for FSL. Then, we describe our FSL system in Section III. In Section IV we discuss the Client-based Privacy Approach (CPA) and NN partitioning at the edge for any split learning-based architecture. Our evaluation with training metrics is presented in Section V-B, while the privacy evaluation is presented in Section V-C.

II. Distributed Learning Background

Federated Learning [1], [17], [18], [19], [28] is a decentralized deep learning technique that trains neural network models using data sources "owned" by multiple clients Fig. 1(a). A logically centralized parameter server holds the latest neural network model, and orchestrates the sharing of its weights between all clients.

The parameter server first sends the same randomly initialized neural network weights to each client where a local model is trained using the local dataset. Until the client models reach a given accuracy threshold, the parameter server repeatly retrieves, averages and overwrites the client weights (Fig. 1(a) – steps 1 to 3). Thus, the global model could be trained with the privately-owned (client) datasets which do not get transmitted over the Internet. Moreover, FL has hyper-parameters to specify the frequency to average the weights of a group of clients, which balance s Internet traffic and model accuracy. Thus, FL is considered scalable in terms of the number of clients, as long as such clients have enough computation power and storage to meet the training constraints, or Service Level Objectives (SLOs).

Split Learning [13] is a distributed machine learning technique that is characterized by a computational split of the neural network model into two partitions. Each partition could run on a separate computing node, hence splitting the computational resource demand. During forward propagation, the output hidden variables of the client partition are sent to the server node — step 1 Fig. 1(b). Then those outputs are used as inputs to the second NN partition and continue the forward propagation. After calculating the loss, the server begins the backward propagation. Once the gradients of the server inputs are calculated, they are sent and used as the gradient of client outputs to finish the backward propagation — step 2. Finally, the complete NN weights are updated with those gradients. Similarly, the server's NN partition can also pair with any other client's NN partition. First, the trained weights in the last client are moved to the new client — step 3. Then the server trains with the new client as mentioned above — steps 4 and 5. The SL algorithm preserves data privacy but suffers from a long convergence time with Non-Independent and Identically Distributed (non-IID) data sources (Fig. 2), and large intermediate data to transmit. Moreover, training with more than one client is sequential, hence poorly scalable. Some remedies attempted to mitigate the

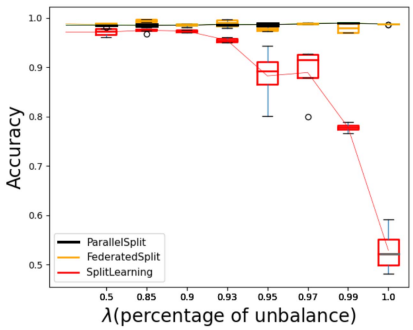


Fig. 2. Accuracy degradation with Non-IID data.

transmission delay of such intermediate data by compressing the model and intermediate data with Knowledge Distillation [24] and BottleNeck layers [21], [22], or avoiding computing the complete model with Early Exits [23].

A. Hybrid Split-Federated Learning

In this article, we consider hybrid split-federated learning systems to combine benefits and minimize drawbacks of Federated Learning and Split Learning.

Combining Split and Federated Learning: Parallel Split Learning (PSL [16]) and SpliltFed [15] (Fig. 1(c)) train multiple client NN partitions in parallel for low latency. During Forward Propagation, different clients' outputs are sent to a single remote server as one batch or sequentially. Then for Backward Propagation, the gradients of the server inputs are sliced (matching the outputs of multiple clients) and sent to each client sequentially.

Model Specific Approaches: FedSL [14] is one example for Recurrent Neural Networks (RNN). They unroll the RNN's feedback loop and split the recurrent NN partition to different nodes holding the sequence data segments. After each epoch, devices average and overwrite their weights. This approach improves training efficiency for RNNs by using extra computation resources to increase parallelism.

FL Extensions: Prior work has focused on extending the FL paradigm. For example, Federated Reconstruction (FRC) [26] prioritize s user data privacy while trading off training efficiency. Their model is partitioned into *global* and *local* shards which are both deployed and alternatively trained in each edge device, i.e., first each client device trains the randomly initialized local shard with the frozen trained global shard; Second, it trains the global shard with the frozen trained local shard. A parameter server then retrieves the updated global shards for averaging, disregards the local shards, and sends the global shards to the clients for the next epoch. This design makes the training stateless, local, and highly scalable for storage.

B. Source Data Privacy

One advantage of these edge training systems is the high perceived level of source data privacy: a user's data does not leave the client device. However, previous work have explored threats based on weights and hidden variables.

Weight Attack: The adversaries can learn the source data by applying gradient descent on a random noise input given a NN and label [29], [30]. Our FSL and the previous PSL and FRC do not share the client weights, so they can better mitigate this leakage.

Hidden Variables Attack: The adversaries can also use the hidden variables to reconstruct client source data with an autoencoder NN [31]. Edge intelligence systems that partition Neural Networks and transmit intermediate data in between partitions are vulnerable to this threat. Previous works proposed different mitigations. NoPeek [31] adds a Distance Correlation (DC) loss to the original loss function, so that they can maximize accuracy as well as the difference between the partition's source data and output (intermediate data) by solving a Multi-Objective Optimization Problem. Differential Privacy (DP-SGD) is another widely used lightweight algorithmic approach [32], [33]. They add (gaussian) noise to gradients when training a NN, so that the outputs of a trained privacy-preserving NN partition can confuse the adversary.

Compared with Homomorphic Encryption (HE) or Secure Multi-Party Computation (SMPC), training with DC or DP-SGD fits better in our client and edge server setup. Encryption and decryption steps cause long latency overhead and can drain battery quickly. There are also other methods, i.e., model compression ([21], [24], [34]), that can potentially enhance privacy guarantee and consume power less than HE or SMPC but higher than training with DC or DP-SGD.

III. Privacy-Oblivious FSL

In this section, we detail our Federated Split Learning (FSL) architecture, which we originally proposed in [25]. FSL is a hybrid approach that combines the advantages of SL and FL. It avoids sending users' source data or sharing the complete NN parameters through the network while being scalable.

Our FSL architecture shown in Fig. 1(e) has three types of entities: (i) edge servers, (ii) clients, and (iii) parameter servers. To train a NN with FSL, we first set up an authentication protocol [35], [36] among the entities to pair each client with one edge server, and edge servers with a parameter server. After pairs are found, the communications are sent without encryption. Then, in each client and server pair, we partition the complete NN into the client's partition and (edge) server's partition.

FSL has three training steps. In step 1, the client forward propagates with source data and transmits the intermediate data to the edge server. The server then finishes propagation and calculates loss. In step 2, the server backward propagates to the client source data. In step 3, after a few epochs, the *parameter server* averages the weights in the edge servers.

Comparison With Other Hybrid Methods: Consequently, FSL will have multiple advantages compared to the other approaches discussed. FSL is more practical at the edge as its clients have lower computation and memory demand, i.e., fewer NN layers to train, compared with FL. Also, FSL only averages the weights in the edge servers, instead of the complete weights in FedAVG. Therefore, FSL reduces the risk of suffering model inversion attacks [29], [30]. FSL also provides better scalability than SL, since client and server pairs can train independently. Compared with FRC [26], FSL is more efficient in training time. FRC updates one weight shard per forward and backward propagation pipeline and runs multiple times to update the

full model. Compared with PSL [16], FSL has four potential advantages. *Scalability:* FSL allows Client-Edge Server pairs to train independently, which allows it to scale to more clients. On the other hand, the PSL server can either work with each client sequentially, or enlarge the batch size accommodating the number of clients and waiting for all intermediate data. In the worst case, when n clients arrive at the same time, the lower bound on waiting time for each PSL client to begin backward propagation is $O(n)$. *Robustness:* We also observe that FSL has to maintain fewer states in each independent pair. The PSL server would temporarily store multiple batches of intermediate data simultaneously, so it needs a sophisticated logging and compaction storage system to avoid or recover from failures. Consequently, FSL is more robust than PSL. *Resilience:* In FSL, failure in one pair would not prevent other pairs from training or inference. However, the PSL design may suffer from resiliency problems when the single server failed. *Bandwidth Congestion:* For FSL, each edge server only processes intermediate data from its client, which leads to lower communication overhead. Meanwhile in PSL, since all intermediate data will be sent to and processed by the single server, bandwidth and computation resources at the server node may get overwhelmed. Our findings are presented in Section V.

IV. Privacy-Aware FSL

We have discussed the efficiency and resilience of FSL. In this section, we consider instead the privacy-preserving properties and propose our privacy-aware FSL. In particular, we discuss how to complement general split learning-based architectures to mitigate the privacy concerns of sharing hidden variables or weights over an honest but curious network. We first give a formal definition of our privacy attacker model, and then we discuss how a Client-based Privacy Approach and certain ways of partitioning Neural Networks help to avoid such attacks.

A. Privacy Attacker Model and Assumptions

We assume that the attacker knows the structure of the client NN ($Client$), the *Intermediate Data* ($Client(x_{Sou})$) transmitted from the client to the edge server in plaintext and an auxiliary dataset (x_{Aux}) with features similar to the client source data (x_{Sou}). Then, following (1), the adversary can train an AutoEncoder NN with an Encoder (Enc) and a Decoder (Dec) [37] to reconstruct x_{Aux} by minimizing a Mean Squared Error (MSE) loss comparing x_{Aux} and the AutoEncoder output $Dec \circ Enc(x_{Aux})$ where Enc has the same NN structure of $Client$.

$$\arg\min_{Dec,Enc}(MSE(x_{Aux}, Dec \circ Enc(x_{Aux}))) \tag{1}$$

And then the trained Decoder (Dec) can reconstruct the client source data (x_{Sou}) based on the intermediate data generated by client model ($Client(x_{Sou})$), as shown in (2).

$$x_{Rec} = Dec \circ Client(x_{Sou}) \tag{2}$$

Details of using this attack are discussed in Section V-C2.

B. Attack Resilience

Given the attacker model, in order to compare the level of privacy guarantee among different privacy approaches, we define an *Attack Resilience* metric (τ) as:

$$\tau = 1 - \frac{\|correct\|}{\|reconstructed\|} \tag{3}$$

It measures the misclassification rate. $\|correct\|$ counts the number of reconstructed images, which can be correctly classified by a trained classifier (Section V-C3). And $\|reconstructed\|$ is the total number of reconstructed images. Then we can compare the misclassification rate of the trained classifier based on the reconstructed datasets. The higher value means the reconstructed images have fewer features to be correctly recognized by the attacker's classifier, which is an indicator for higher attack resilience.

C. Client-Based Privacy Approach in Distributed Setting Via Distance Correlation (CPA-DC)

Motivated by NoPeek [31], where they minimize Cross-Entropy while maximizing Distance Correlation (DC) in one loss function, we optimize the two loss functions alternatively in two rounds, as shown in equation (4): At epoch e, if $e \, mod \, \mathrm{F} == 0$, given *Frequency* F, the *regular round* minimizes the cross-entropy loss function $Loss(\cdot)$ with results of the inference model (server NN $g(\cdot)$ takes outputs of client NN $f(\cdot)$ based on input x) and ground truth $label$s in the (edge) server. Otherwise, the *DC round* maximizes the DC loss function comparing client source data x and output $f(x)$ in the client node. Then by balancing the two rounds with *loss multiplier* m and *Frequency* F, we achieve high accuracy and privacy guarantee as only important features are preserved for training.

$$L = \begin{cases} Loss(g \circ f(x), label) & \text{if } e \bmod F == 0 \\ m \cdot DC(x, f(x)) & \text{otherwise,} \end{cases} \tag{4}$$

Our Client-Based Privacy Approach (CPA) can also work with other loss functions or methods teaching NNs to focus on important features. In Section V-C, we evaluated the trade-off among training time, accuracy and attack resilience, comparing DC loss and Differential Privacy (DP-SGD).

D. How to Partition a Neural Network?

In this section, we discuss the problem of selecting how many layers need to be assigned for each NN partition, i.e., client NN depth. This tradeoff will tune training time (processing and transmission delay), privacy, and accuracy. Capturing the tradeoff between all these metrics is challenging. To illustrate, consider the tradeoff between processing and transmission delays. The output size of different layers can be quite different, so a few partitioning policies may lead to significant transmission overhead, increasing training time and hence diminishing the gain of the hybrid FSL compared to the original Federated Learning architecture. For example, in VGG-16 [7], the output size of the first convolutional layer is two times the size of the second convolutional layer. Thus, a system with a model cut after the second convolutional layer can trade off the extra processing

delay at low-capacity clients while yielding a lower transmission delay. We evaluate this effect in Section V-B2.

Analytically, this effect is captured by solving Problem 5, where α and β are developer-specified parameters that represent positive weights for the transmission delay of intermediate data (I) and computation delay (C), the parameter d represents the depth of the client neural network, and b represents the bandwidth, measured periodically. To efficiently solve this problem, similar to Neurosurgeon [38], we consider using two regression models to predict the delays I and C given the available bandwidth, by profiling the model layers for output sizes and processing times offline using the resources for training.

$$\min_{d}(\alpha I(d \mid b) + \beta C(d)) \tag{5}$$

The solution of Problem 5 is optimal with respect to delays, however, it can be sub-optimal with respect to privacy and accuracy. As Section V-B2 shows, the client processing and transmission delay of FSL reaches the minimum when the client NN depth is between 7 and 16. In Fig. 7(a), instead, the client NN needs more than 16 layers to be above 90% attack resilience. Thus, we conclude that an optimal NN partitioning decision should balance different objectives and constraints, including transmission delay, processing time, privacy, and accuracy, as shown in our Problem formulation 6. In such a problem, W represents the model weight vector, (I', C', A, R) is the tuple representing the observations for transmission delay, computation delay, accuracy, and resilience, (γ, κ) are new user-specified positive weights, and d is the client NN depth.

$$\max_{W,d} \quad (-\alpha I'(W, d \mid b) - \beta C'(W, d) + \gamma A(W, d) + \kappa R(W, d)) \tag{6}$$

To solve Problem 6, we have to train W for each d until convergence and then find the best d. This brute force method is inefficient. A more efficient approach would rely on predicting the delays, accuracy, and privacy without the full training of the model. Extending the approach in [38] to go beyond profiling delays, is challenging. This is because the accuracy and attack resilience for each client and edge server pair are harder to profile and predict. Specifically, their profiling depends on the weights trained on other pairs, the distribution of source data among clients, number of clients, number of layers to average in SerAVG, and training epochs (Sec. Sections V-B4 and V-C). Another work can predict the model accuracy [39], but it is based on the already trained model. Therefore, for our FSL architecture with SerAVG, a prediction method for partitioning remains an open question for future work. In this article, we experimentally demonstrate the best model partitioning that balances requirements on training time, accuracy, and privacy.

V. Evaluation Results

In this section, we describe the evaluation results for our Privacy-Oblivious FSL and Privacy-Aware FSL architectures with our privacy-aware approaches (CPA in Section IV-C and Neural Network Partitioning in Section IV-D).[1] Our evaluation demonstrates the advantages of FSL over PSL and FRC in terms of training time, memory usage, and convergence rate. Moreover, we also show that our privacy-aware approaches can prevent the reconstruction of source images from intermediate data in the Split Learning-based systems. We first discuss our experimental setup, then present our evaluation results of Privacy-Oblivious FSL and Privacy-Aware FSL in Sections V-B and V-C.

A. Experimental Setup

This experiment set studies the convergence for Privacy-Oblivious FSL and the privacy guarantee of Privacy-Aware FSL across different hardware and applications with different NNs and datasets.

For the hardware, we used two types of nodes on Chameleon Cloud [40]. One has an RTX6000 GPU, two Intel Xeon Gold 6126 CPUs and 187 GB memory. The other one has four NVIDIA V100 GPUs, two Intel Xeon Gold 6230 CPUs and 128 GB of memory. We emulated the computer network among our distributed learning entities on the localhost interface on a physical machine, and each experiment was set to use a single GPU, so that we can ignore the network bandwidth bottlenecks.

For the applications, we considered three classification tasks and implemented them with PyTorch [41]. Then the distributed communication among entities of the systems was handled by PySyft [42] and PyGrid [43] and no encryption was applied to the transmission. The first application runs a general image classification task with a VGG-16 [7] Convolutional Neural Network (CNN). The model was pre-trained using Imagenet [8] and then fine-tuned with the CIFAR-10 dataset [9]. We run this task on 5 clients running an NVIDIA V100 GPU. The second task uses a LENET [44] CNN to recognize handwritten numbers in the MNIST dataset [44] on 20 clients running an NVIDIA RTX6000 GPU. The third task classifies traffic, not images. In particular, we decomposed a one-dimensional-CNN, trained with the ISCX VPN-nonVPN (ISCX) traffic dataset [45], using 5 clients running on an RTX6000 GPU. We partitioned the dataset and assign ed among different clients with Independent and Identically Distributed (IID) probabilities and all our plots show 95% confidence intervals unless otherwise specified. Our goal is to verify that FSL can always converge, with different tasks, different NNs, different devices, and different data distributions. We verified the advantages in delay or privacy of FSL over existing solutions.

B. Evaluation Results for Privacy-Oblivious FSL

This section illustrates the methodology and draws observations of our experiments. Overall, our evaluations show that Privacy-Oblibious FSL has less overhead and similar accuracy compared to existing solutions. In particular, we evaluate training time (Sec. Section V-B2), memory consumption (Sec. Section V-B3), and learner accuracy (Sec. Section V-B4).

1) Experiment Design: With different datasets and numbers of clients, to reach at least 90% accuracy, the neural networks used for image classification needed 20 epochs. And the traffic classification model needed 80 epochs.

2) Training Time Evaluation: To evaluate training time, let us consider the experiment whose results are reported in Figs. 3 and 4. The x-axis indicates the *Cut Index*, i.e., the index

[1] https://github.com/HEECMA-BU/FSL

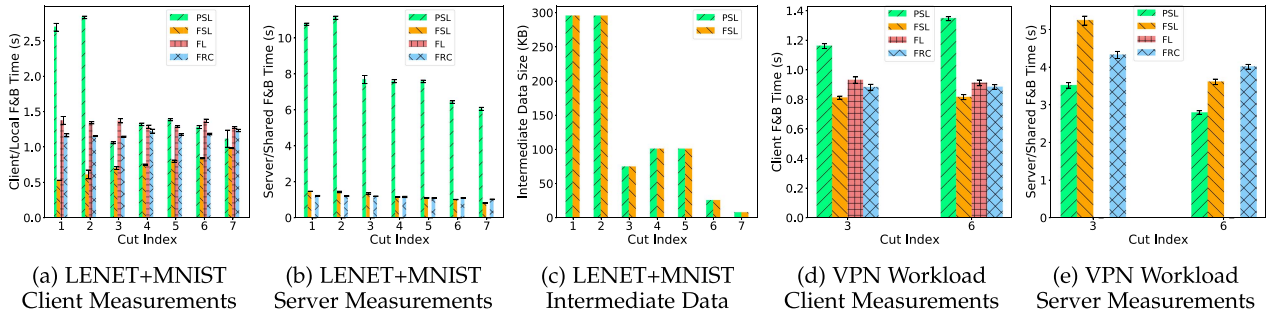


Fig. 3. *LeNet+MNIST:* (a) Client Time, (b) Server Time, (c) Intermediate data size. Observations: 1) Intermediate data size is correlated with the times taken by the PSL architecture while having little correlation under FSL. Notice that since FSL and PSL do not modify the NN structure other than splitting the NN, the sizes of intermediate data generated by an FSL client and a PSL client at one cut index with the same input image are the same; 2) FRC has almost twice the overall training time as FL; 3) Plots are obtained by averaging 20 clients' results. The intermediate data size is under the batch size of 16 and each image was resized to (1,32,32). *VPN Workload:* (d) Client Time (e) Server Time. Similar considerations are valid for the VPN dataset. Tested with 5 clients with a one dimensional NN with an input size of (1,784). Still, FSL has the shortest Client F&B time compared to the other settings.

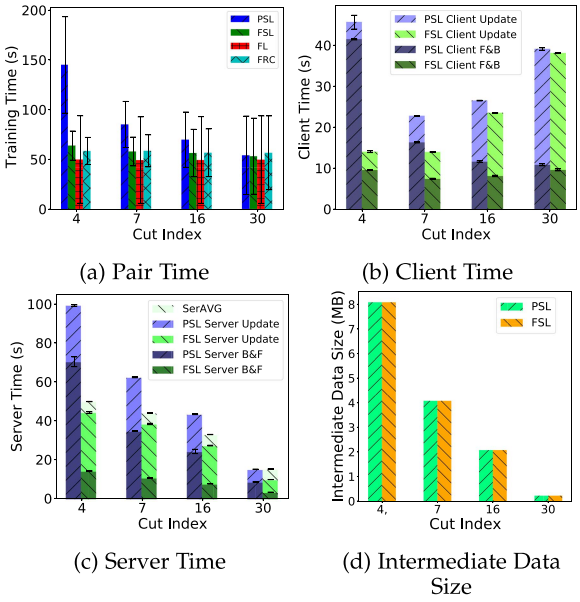


Fig. 4. VGG + CiFar10: Plots show the effect of Cut Index over (a) averaged overall training time, (b) time spent in clients (5 clients average), (c) time spent in server, (d) intermediate data size. The transmission delay caused by intermediate data size can dominate the training time (occurring during F&B propagation).

of the last layer in the client/local part of the NN. With LENET and MNIST, we can observe from Fig. 3(a) and (b) that FSL has the shortest "Client Forward and Backward Propagation" (Client F&B) time among all other distributed architectures. The Client F&B time includes transmission time for gradients and computing both activations and gradients in Client NN. And Server F&B time includes transmission time for hidden variables from client to server and computation in Server NN. Notice that the weight update time is separately counted by "Client Update Time" and "Server Update Time". Moreover, we note that PSL is more vulnerable than FSL to limited bandwidth across splits. PSL is consistently the slowest, due to its inefficient server design; the server has to synchronize and process all batches of intermediate data in each training epoch stacked as a big batch or sequentially.

Observing FRC and FL, we see the F&B times do not change along with the Cut Index (Fig. 3(a) and (b)). Note also that FRC is not training time efficient. It updates its complete model with two almost full forward and backward steps [26], which is shown in the same figures: the FRC total F&B time for local and shared weights is almost double compared to the FL training time.

We were able to obtain similar results comparing the F&B times in another predicting scenario: the 1D CNN implemented by [45] (Fig. 3(d) and (e)). Due to the limited size of this neural network (with only two convolutional layers), we evaluated the architectures with merely two Cut Indexes: at layer 3 and layer 6 of the NN. Even in this experiment, we observe how our FSL still has the shortest Client F&B time. PSL is the worst performant at each cut, and FL keeps performing better than FRC.

FSL consistently Uses Less Time in Each Training Epoch Than the Other Analyzed Architectures: We found that PSL perform worse than FSL because of the single-server architecture. PSL has similar results when comparing its client F&B time with FL and FRC. FRC is not training time efficient. Its total F&B time almost doubles compared to FL.

When evaluating the training time on the CIFAR-10 scenario, we found a different trend (Fig. 4(a)): PSL had the longest training time, except for a cut index of 30. Moreover, FSL did not always perform the best. When most of the layers run within the client, FL has a shorter training time. This is because the size of intermediate data changes as the cut moves, and with smaller data to send, the overall training time can be shorter. Fig. 4(b) and (c) show the extra F&B time during training. And existing works for SL have discussed the similar behavior [21], [22], [23], [24].

In particular, we show that FSL outperforms PSL as PSL F&B time is more vulnerable to intermediate data transmission. In Fig. 3, Client F&B time of FSL keeps decreasing with a smaller Cut Index, while that of PSL still increases at Cut Index 6, 5, and 3, 2, although the intermediate data in this experiment is much smaller than using VGG-16. This behavior is caused by the single server bottleneck. Thus, FSL is more scalable in terms of training time. Such observation also explains why both client and server F&B times of PSL are consistently larger than FSL.

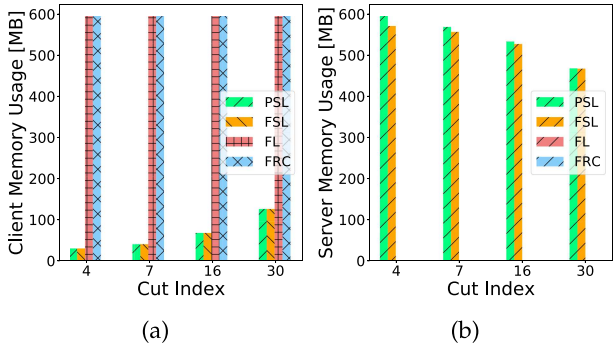


Fig. 5. VGG + CiFar10: Plots show (a) Client and (b) Server memory demands (size of model weights and hidden variables), with batch size 32 and image size (3,32,32). The FSL edge server uses slightly less memory, as less batches of hidden variables are present simultaneously compared to PSL. Also, the FSL client uses less memory compared to FRC/FL as only the Client NN partition is deployed versus the whole model. Notice that the clients of FSL and PSL have the same memory demand as the client models and output sizes are the same given the same cut index and image size.

The Intermediate Data and Gradients Can Cause Significant Network Overhead: Such overhead, however, is better mitigated by our FSL than PSL. Existing work [21] [23] for Split Learning, as well as our partitioning strategies (Section IV-D) can further mitigate the communication overhead. Thus, the additional delay in FSL is not considered a severe bottleneck compared to those systems training at the edge, like FL.

3) Memory Consumption Evaluation: Memory usage of each entity in the edge training and inference systems limits the scope of devices that can join the system. To compare which system is more flexible to deploy in terms of memory capacity on devices, we show the memory demands in FSL, PSL, FL, and FRC systems.

Real-world memory utilization can be highly variable as it depends on several implementation factors, such as libraries used and the Remote Procedure Calls (RPCs) implemented. However, the size of a model and its hidden variables are known. The following results show that FSL clients consume less memory compared to FL and FRC, and the FSL edge server occupies less memory than PSL.

The two plots in Fig. 5 show the memory demands computed at the client and the server for each architecture. The x-axis shows the Cut Index and the y-axis represents the corresponding expected memory usage in MB. Note that FL and FRC do not split the NN, so their memory demands during training are only shown in the left plot.

As shown in Fig. 5(a), the sizes of each client NN's weight and hidden variables at each layer are the same in FSL and PSL. Since FRC and FL compute the full NN in the client during training, they require more than five times the memory, for Cut Index 4 to 30. Fig. 5(b) instead shows that the server memory demand decreases with the cut index, as expected. However, notice that the PSL server need s extra memory to hold intermediate data from different clients, which leads to slightly higher memory demand than FSL.

Memory Usage of FSL Compared to Other Systems: To conclude, we found that FSL's servers are lightweight compared to PSL. Thus, state management would be easier in FSL. Also, FSL's clients are lightweight compared to FRC and FL, during training, so they are more suitable at client devices.

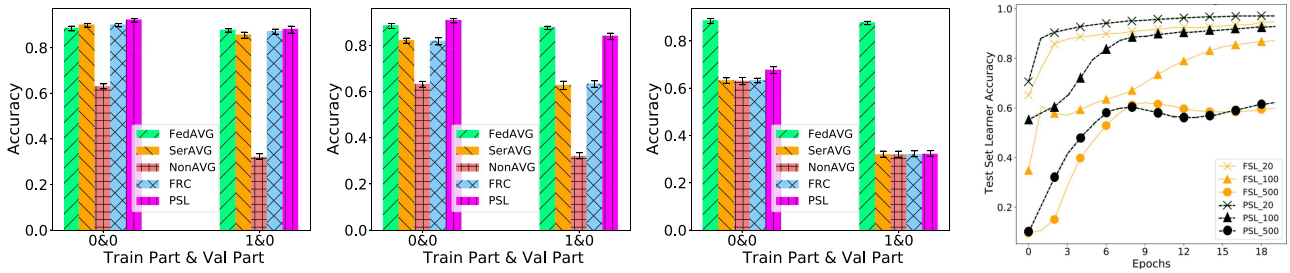
4) Learner Accuracy Evaluation: The distributed training method in FSL, SerAVG, is similar to FedAVG (FL) but only averages the model weights of the NN in the (edge) servers (Section III). In this subsection, we evaluate the convergence speed of SerAVG based on source data distribution, cut indices and data size at each client. Notice that we expect a higher accuracy with further hyper-parameters tuning, given prior results in similar contexts [46], [47]. While our accuracy results show 95% confidence intervals, parameter tuning is out of the scope of this article.

Tradeoff among Non-IID Data, Cut Index and Accuracy: In this experiment set, we evaluate the model accuracy given Non-IID source data and different numbers of feature layers to average. We split the data into two parts, namely part 0 and part 1. Each part includes data corresponding to half of the labels in the MNIST dataset and is divided into a trainset and a testset. Note that the aforementioned way of splitting the dataset is an extreme Non-IID case. For example, a model trained on part 0 of the training set has no knowledge of the labels in part 1 of the training set. To assess the systems on more realistic data distributions, we further add 10% samples, uniformly and randomly selected from the complete MNIST dataset to both parts of datasets.

Consider Fig. 6(a), (b) and (c), the x-axis represents the data partitions that trainset or testset belongs to, i.e., 0 & 1 means trainset of part 0 and testset of part 1 were used. The y-axis shows the validation accuracy. We show the results with cut indices 3, 5, and 7 in a modified LeNet, which correspond to the three convolutional (feature extraction) layers. From left to right, the accuracy decreases from the level of FedAVG (full weights sharing) to NonAVG (no weights are shared) when the server/shared NN for FSL, PSL, and FRC is shallower. When the Cut Index is 3, SerAVG, FedAVG, and PSL perform equally well. When the Cut Index is 5, SerAVG is worse than FedAVG but still higher than NonAVG. FRC has similar accuracy to SerAVG. PSL maintains similar high accuracy as FedAVG. Further, when the Cut Index is 7, all hybrid methods have similar accuracy as NonAVG. We conclude that SerAVG can enhance the accuracy in the Non-IID source data set, while worse than FL and PSL depending on the number of layers to average.

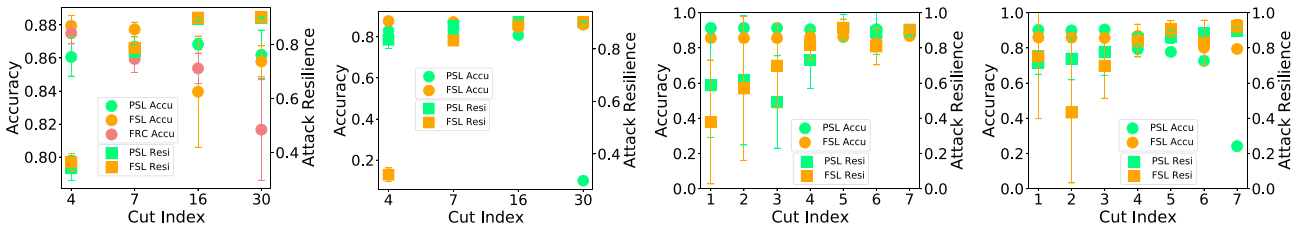
Comparing FedAVG and SerAVG, FedAVG averages all weights, while each FSL client NN is trained with a smaller batch of gradients ignoring other clients, which explains why a shallower (edge) server NN leads to lower accuracy. On the other hand, comparing PSL and FSL, the PSL server optimizes for minimal loss using all clients' batch output. Thus, the gradients applied with SerAVG will be less accurate compared to PSL, as the FSL clients are not updated considering the direction of other clients' gradients.

Note also a similar but smoother drop in accuracy based on Cut Indexes with IID CIFAR10 sources and VGG16 in Fig. 7(a). The two experiments with MNIST and CIFAR10 datasets suggest that SerAVG can achieve high accuracy by utilizing data of all clients when applied to suitable NN models and Cut Indexes. Specifically, when the Cut Index is small, SerAVG achieves high accuracy while allowing deployments over resource-constrained client devices.



(a) Accuracy (Cut Index = 3) (b) Accuracy (Cut Index = 5) (c) Accuracy (Cut Index = 7) (d) Accuracy (different number of clients)

Fig. 6. From plots (a), (b) and to (c), the accuracy of SerAVG, PSL and FRC decreases from the FedAVG level of accuracy to NonAVG, as less convolutional layers are averaged. PSL minimizes loss based on the hidden variables from all clients, so it achieves a little higher accuracy compared to SerAVG and FRC. However, FSL can achieve better privacy as shown in Section V-C. SerAVG: Average (edge) server NN's weights; NonAVG: Each pair trains on its own; FedAVG: Average the complete NN's weights. In plot (d), we use cut index of 3 and the FSL and PSL accuracies converge to similar values.



(a) Privacy Oblivious (VGG16+Cifar-10) (b) DC 1 & Muiltiplier=2.0 (VGG16+Cifar-10) (c) Privacy Oblivious (Lenet+MNIST) (d) DC 1 & Multiplier=0.3 (Lenet+MNIST)

Fig. 7. Accuracy and Attack Resilience for Privacy Oblivious and Privacy-Aware Architectures based on Cut Index. Note: DC 1 in captions refer to $DC\_Frequency = 1$. These figures show that to reach high accuracy and attack resilience, the Cut Index cannot be too big or too small, and loss multiplier is another way to enhance attack resilience. Furthermore, the loss multiplier can be more practical than DC_Frequency, since it doesn't introduce overhead to the training steps.

Tradeoff Between Resource Demand and Accuracy: In this experiment, we evaluated the accuracy of FSL with small resource demand, i.e., small input data size. The dataset is IID across 20, 100, and 500 clients yielding different input data size at each client, and we use cut index of 3 to isolate the accuracy drop caused by SerAVG. Our results are shown in Fig. 6(d). The x-axis indicates the index of epochs, and the y-axis shows the corresponding test set accuracy after a certain number of epochs. We found that with LENET and MNIST, the validation dataset can reach an accuracy range of 87% to 93%, as long as each client has enough data to train the machine learning model.

5) Advantages and Limitations: Our evaluation of Privacy-Oblivious FSL shows shorter training times and less memory footprint, although its accuracy is consistently lower (but only slightly) to that of other methods. However, when server node resources are not limited, e.g., in the cloud, FSL may have a marginal gain over PSL. Thus, it is important to consider resource availability when choosing a system design.

C. Evaluation Results for Privacy-Aware FSL

In this section, we present the evaluation results of our privacy-aware FSL architecture and show that it can provide certain privacy guarantees. FSL clients do not share the source data and model weights, so adversaries cannot directly access the source data or reconstruct them with the model weights using model inversion attacks [29], [30]. However, when compared with Federated Reconstruction (FRC) [26] which trains a complete model at the client, FSL still sends the intermediate data through a network to complete the forward and backward propagation between clients and (edge) servers. An adversary could use such data to reproduce the source data, e.g., through an Autoencoder [48] NN, trained with a specific dataset, in Section IV-A.

To assess how our approach mitigates such vulnerabilities, we first introduce the evaluation setup, the design and usage of the attacker Autoencoder NN, and our experiment methodology. Then, we discuss the results of different privacy approaches, i.e., NoPeek [31] and the Client Based Privacy Approach (CPA), and the privacy level of different ways of partitioning the NN. NoPeek solves a multi-objective optimization problem of two loss functions, i.e., one maximizes accuracy, and the other maximizes the differences between source images and intermediate data. For CPA, we evaluate CPA-DC and CPA-DP. CPA-DC (Section IV-C) optimize the two loss function in NoPeek alternatively. CPA-DP applies a DP-SGD [32] algorithm in the clients. Finally we evaluate the privacy guarantee of different partitioning of client NN and server NN motivated by Section IV-D. We conclude this section by presenting results that demonstrate the high resilience to privacy attacks of our proposed FSL and the advantages in training efficiency.

1) Evaluation Settings: Our Privacy-Aware FSL and PSL extend our Privacy-Oblivious version by adding the CPA. Partitioning the NN was made easy by considering only sequential

NNs (e.g., LeNET and VGG16). We tested both systems with the same image classification workloads (e.g., MNIST and CIFAR10) on the same hardware (i.e., NVIDIA RTX6000 and NVIDIA V100, respectively) as the Privacy-Oblivious setting.

2) Setup the Attacker's Auto-Encoder Neural Network: Following Section IV-A, the attacker trains an autoencoder NN, consisting of an Encoder and a Decoder, with an auxiliary dataset using the MSE loss function to minimize the difference between inputs and outputs, i.e. the input data can be faithfully reconstructed (decoded) from the output. The encoder part uses convolutional layers to extract latent variables from its input dataset, and the decoder uses the encoder's last activation function outputs and transposed convolutional layers to reproduce the input dataset of the encoder. Consequently, we implement the encoder NN with the client NN structure, and let the decoder strictly mirror the client NN structure, i.e., the i^{th} layer of an encoder (client NN) would be the i^{th} last layer of the decoder, with transposed convolutional layer.

3) Privacy Evaluation: Methodology: In this subset of our evaluation, we want to show both CPA and carefully designed ways of partitioning NN provide high attack resilience while preserving high accuracy, based on different datasets, NNs, for split learning-based systems.

Each experiment includes trials initializing the Autoencoder's weights and data loaders with different random seeds. In each trial, the attacker used a dataset containing similar features to the learner's source dataset. To reconstruct MNIST, we selected EMNIST [49], and for CIFAR10 we selected the CIFAR100 [9]. The EMNIST dataset contains hand-written characters instead of numbers in MNIST, so features like lines and curves are the same and the attacker can decode those activation function outputs. Similarly, the CIFAR100 dataset contains 100 classes of RGB images instead of the 10 classes in CIFAR10, so the common features, e.g. classifying cat or dog, can be used to reconstruct with the activation function outputs from the CIFAR10 dataset.

For each trial of the experiment, we first let the attacker learn to reproduce her datasets. Based on the MSE loss, *i.e.* a loss function that measures how different the original and reproduced images are, the attacker updates her weights in each epoch. After 20 epochs, we used the decoder to reproduce the learner's dataset from the intermediate data.

When an autoencoder NN is trained, we train a new classifier with the same NN structure and source data as the learner to classify the reproduced images for 20 epochs. The mean and standard deviation of this classifier's *attack resilience* τ (Section IV-B) are recorded.

We show the results comparing Privacy-Oblivious and Privacy-Aware FSL and PSL systems with different CPAs and Cut Indexes. Then, we evaluate the trade-off between accuracy and attack resilience for FSL and PSL.

4) Privacy Evaluation Using NoPeek: As we illustrated in Section II-B, NoPeek solves a multi-objective optimization problem that takes in the source data and intermediate data to maximize the difference with a Distance Correlation (DC) loss function, as well as the prediction and labels to maximize the accuracy. To solve such an optimization problem, NoPeek has to share the value of the loss over a network which may cause vulnerability or added complexity in maintaining the gradient graph. As shown in Table I, this approach has both high attack resilience (i.e., 97% for PSL and 98% for FSL) and high learner's accuracy (i.e., 97% for PSL and 96% for FSL) when trained for the same number of epochs and clients as the Privacy Oblivious experiment with the MNIST dataset and LENET NN.

TABLE I
NoPeek Stats With 20 Clients Training 20 Epochs

cases	PSL	FSL
attack_resilience(τ)	0.9733	0.9837
learner_accuracy	0.9702	0.9614

5) Evaluation Result Using Client-Based Privacy Approach Via Distance Correlation (CPA-DC): To mitigate the drawbacks of the loss value sharing, we consider a new approach that prevents transmitting data outside clients, improving upon NoPeek. We optimize for the similar two objectives in NoPeek alternatively in the Client-Based Privacy Approach via DC. As shown in (4), there are *DC Frequency (F)* and *Loss Multiplier (m)* to evaluate. *F* defines how many times the DC loss function is optimized given the Cross Entropy has been optimized once. *m* is applied to the loss function result. These two parameters control how different the intermediate data and source data will be, by changing the frequency of optimizing the DC loss and by changing the learning rate of gradients applied during that optimization, respectively.

We note multiple tradeoffs in CPA-DC. First, CPA-DC is not as training time-efficient as NoPeek. NoPeek can optimize its two objectives with one forward and backward steps while CPA-DC has to solve them sequentially. However, we consider that NoPeek transfers more information than necessary over a network. Second, there are tradeoffs for *DC Frequency (F)* and *Loss Multiplier (m)*. Increasing *F* adds more epochs to optimize for DC loss, so the attack resilience would be higher at the expense of a longer training time. Instead, by increasing *m*, we can keep a small *F*, which reduces training time while maintaining attack resilience. Intuitively, multiplying the DC loss by a larger *m* is similar to increasing the gradient descent step size. Thus, we need fewer DC epochs to maintain the attack resilience, while the gradients can be sub-optimal. Based on the discussion, we expect that a large *m* combined with *F* of 1 can balance between training time efficiency and DC loss gradients' accuracy. These two parameters should be carefully designed in a production environment.

We first experimented with MNIST classification with different DC Frequencies and a constant loss multiplier of 0.1 (equivalent to reducing the learning rate by 10%), and studied the tradeoff between accuracy and attack resilience, as shown in Fig. 8(a). The x-axis represents the DC Frequency, the left y-axis shows the learner accuracy and the right y-axis shows the corresponding attack resilience.

From the top plot of Fig. 8,[2] as DC Frequency is increasing, for both Privacy-Aware FSL (PAFSL) and Privacy-Aware PSL (PAPSL) systems, the attack resiliency increases and the learner accuracy decreases, as expected. Notice that PAFSL achieves better accuracy and good resilience for most DC Frequency values. For DC Frequency from 10 to 20, given that the attack

[2]A DC Frequency of zero corresponds to Privacy-Oblivious FSL and Privacy-Oblivious PSL, i.e. without privacy awareness.

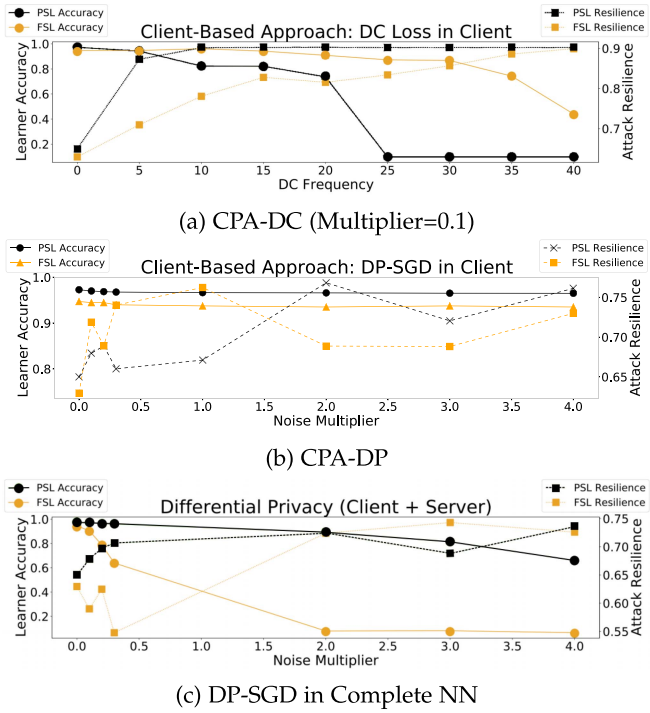


Fig. 8. LeNet+MNIST: Accuracy and attack resilience (τ) with 20 clients and a Cut Index of 3 for Client-Based Privacy Approaches (via DC (a) and via DP (b)) and DP-SGD on the global learner model (c). Our FSL with both client-based policies guarantees a high level of privacy and accuracy.

resilience of PAFSL and PAPSL are close within 10% difference, PAFSL achieves more than 90% accuracy. From 25 to 35, PAPSL does not learn any features while PAFSL still has about 80% accuracy. When DC Frequency is five, the PAPSL has an advantage over PAFSL, with close accuracy, and PAPSL has around 20% more attack resilience.

The Result Shows That Privacy-Aware FSL With CPA-DC is Easier to Tune for High Accuracy and Attack Resilience: Within the wider domain of DC Frequency, PAFSL has higher accuracy and good attack resilience compared to PAPSL. This is because of the learning rate in SerAVG and PSL. The server weight update rule of PSL is shown in (7).

$$W_g^{t+1} = W_g^t - \eta \sum_{i=1}^{N_C} \frac{\partial g(f(x_i))}{\partial W_g} \tag{7}$$

Similarly for FSL, we have (8).

$$W_g^{t+1} = \frac{\sum_{i=1}^{N_C} \left(W_g^t - \eta \frac{\partial g(f(x_i))}{\partial W_g} \right)}{N_C} = W_g^t - \frac{\eta}{N_C} \sum_{i=1}^{N_C} \frac{\partial g(f(x_i))}{\partial W_g}, \tag{8}$$

where W_g^t indicates the weights in the server at iteration t, g is the server NN, f is the client NN, x_i represent the i-th batch of data, N_C is the number of clients, and η is the step size. Intuitively, since PSL has a larger step size, its server NN can be confused quicker than FSL servers by the intermediate data. Moreover, the confused server NN can further confuse the client NN. It justifies our observation that PSL's accuracy and attack resilience become unstable quickly when increasing the *DC Frequency (F)*. Therefore, we conclude that *FSL is easier to tune compared to PSL.*

In Fig. 7(b) and (d), with a fixed *DC Frequency (F)*, we show the accuracy (left y-axis) and attack resilience (right y-axis) based on different *Loss Multiplier (m)* for different models and datasets. Furthermore, we compared the privacy oblivious cases (Fig. 7(a) and (c)), and the privacy-aware cases at different Cut Indexes (x-axis). As expected, increasing the *Loss Multiplier (m)* enhances attack resilience but reduces accuracy for different datasets prevalently, especially when the client NN is deep.

Overall our evaluation of CPA-DC shows good attack resilience and accuracy with a combination of small *DC Frequency (F)* and big *Loss Multiplier (m)*. And our FSL has better accuracy and similar attack resilience to PSL. We hence conclude that our CPA-DC can defend against our attacker model.

6) Privacy Evaluation With Differential Privacy Approach: The previous section has discussed the CPA-DC, but instead of DC there are other lightweight methods that can enhance the privacy guarantee which adds noise to the client's NN while preventing depletion of the client's battery quickly. In this section, we compare CPA via Differential Privacy (CPA-DP uses the popular DP-SGD [32] algorithm inside clients) and CPA-DC. Also, we show that naïvely using DP-SGD in an FSL system would lead to low accuracy. The implementation extends the Privacy-Oblivious FSL with a DP-SGD optimizer, provided by the Opacus [50] library. This method would add normally distributed random noise to the gradients during backward propagation based on $noise\_multiplier$ ϵ. This parameter controls the magnitude of the noise added. Notice that DC generates the gradients in a specific direction to reduce the correlation between intermediate data and source data in each *Distance Correlation round*. So we expect CPA-DP to have a worse level of privacy, given the same level of learner accuracy, compared to CPA-DC. Thus, the focus of this section is to show that CPA can be applied with other privacy methods like DP, despite DP's worse privacy compared to DC.

We summarize the results of CPA-DP in Fig. 8(b). This plot shows the result when Cut Index equals 3. The x-axis is the $noise\_multiplier$. The attack resiliency of FSL and PSL with $noise\_multiplier > 0$ is consistently better by nearly 5% than FSL and PSL with $noise\_multiplier = 0$. At the same time, the accuracy decreases by less than 1% in either FSL or PSL from $noise\_multiplier = 0$ to $noise\_multiplier = 4$.

CPA is a General Approach and Can Be Customized With Different Methods to Enhance Attack Resilience: The evaluation shows that CPA-DP can also improve attack resilience, while learner accuracy does not change much. Compared with CPA-DC, both methods provide similar learner accuracy, but CPA-DC's attack resilience is higher.

We now compare CPA-DP against applying DP-SGD in both client NN and server NN. Fig. 8(c) shows high attack resilience, but the learner accuracy of FSL drops below 60% when $noise\_multiplier \geq 0.3$. Meanwhile, PSL shows a similar behavior as using CPA-DP. So, CPA-DP is considered a better method for FSL to enhance its attack resilience than DP-SGD applied in both clients and servers. The reason for FSL's lower accuracy under DP-SGD and high noise can be attributed to its

SerAVG. After applying SerAVG, the distribution of the random noise in the server NN can be arbitrary, as shown in (9), while the noise in the client NN stays intact. Thus, the resulting complete NN in FSL may not converge.

$$W_g^{t+1} = W_g^t - \frac{\eta}{N_C} \sum_{i=1}^{N_C} \left(\frac{\partial g(f(x_i))}{\partial W_g} + e_{g_i}^t \right) \tag{9}$$

The variable $e_{g_i}^t$ indicates the gaussian noise added for the i-th server NN at iteration t according to $e_{g_i}^t \sim \mathcal{N}(0, (noise\_multiplier \times max\_gradient\_norm)^2)$ [50].

7) Evaluation Using Different Ways of Partitioning NN: In Fig. 7(a) and (c), the right y-axis shows the attack resilience, and the x-axis indicates the Cut Indexes. Overall, if we have a deeper client NN (i.e., moving from smaller to bigger Cut Indexes), the attack resilience increases and accuracy decreases (consistent with the SerAVG Evaluation in Section V-B4). After adding more layers, the intermediate data would have fewer features from the source data, but only keeps those that can improve the classification accuracy. Thus, fewer features are preserved and the attacker's ability to reconstruct the source's data is hindered.

We also want to emphasize that with the CIFAR10 workload, when there are 7 layers in the clients, the attack resilience reaches about 80%. We reach the same result with MNIST workload at the Cut Index of 4. No extra privacy-aware method was used, and as we discussed earlier, the transmission delay can also be reduced with a deeper NN in the client due to potentially smaller intermediate data.

Cut Index is an Important Hyper-Parameter for Training Delay, Accuracy, and Privacy: Different Cut Indexes bring the following tradeoff: the deeper client NN adds more resource demand at the edge, but reduces the transmission time and enhances attack resilience. On the other hand, a shallower server NN may lead to lower accuracy with SerAVG.

Furthermore, system architects can combine the approaches mentioned above, e.g., having a moderately deep NN in clients and using the CPA-DC, to find a balance between resource demand and performance. As in Fig. 7(d), with $cut\_index = 4$, $DC\_Frequency = 1$ and $loss\_multiplier = 0.3$, we still get about 80% attack resilience and more than 90% accuracy.

On the other hand, when comparing FSL and PSL, we note that FSL has more hyper-parameters to tune. But, in all experiments reported in Fig. 7, when the Cut Index is large, FSL has better accuracy than PSL. So we conclude that a carefully specified way of partitioning the NN can benefit the most when applied in hybrid federated-split learning systems.

8) Advantages and Limitations: The CPA-DC method introduces a separate *DC round* to minimize DC loss, which adds extra training overhead. We further introduce a *loss multiplier*, to minimize the overhead by changing the step size. Tuning it requires profiling and experience. We thus conclude that FSL and CPA cannot solve all the limitations in Federated Learning (FL) and Split Learning (SL) systems. But, their flexibility allows users to customize a better distributed learning system that meets their objectives in terms of latency, privacy, and accuracy.

VI. Conclusion

Systems like Federated Learning (FL), Split Learning (SL), and later works aim to fit specific scenarios such as distributed model training and inference. However, they are not flexible enough to fit some use cases with the recent development in edge and constrained devices.

In this work, we propose and extensively evaluate Federated Split Learning (FSL), a system for efficient training and inference with high-level privacy for clients' source data. We present a Client-based Privacy Approach (CPA) for split learning-based systems to provide high attack resilience by adding noise to the intermediate data. We also study the training time, accuracy, and privacy level of different ways to partition a NN in FSL. As further works, we aim to research prediction-based NN partitioning methods.

Moreover, comparisons between FSL and existing NN training and inference systems at the edge are carried out, along with explanations of their pros and cons against FSL. Among the main analyzed systems, the Parallel Split Learning (PSL)'s principal limitation is a slow and stateful server, and the Federated Reconstruction (FRC) system is inefficient in training time and inference time.

References

[1] D. H. Mahlool and M. H. Abed, "A comprehensive survey on federated learning: Concept and applications," 2022, *arXiv:2201.09384*.

[2] P. Moritz et al., "Ray: A distributed framework for emerging AI applications," in *Proc. USENIX Conf. Operating Syst. Des. Implementation*, 2018, pp. 561–577.

[3] C. Park, D. Hong, and C. Seo, "An attack-based evaluation method for differentially private learning against model inversion attack," *IEEE Access*, vol. 7, pp. 124988–124999, 2019.

[4] S. Ramaswamy, R. Mathews, K. Rao, and F. Beaufays, "Federated learning for emoji prediction in a mobile keyboard," 2019, *arXiv: 1906.04329*.

[5] A. Hard et al., "Federated learning for mobile keyboard prediction," 2018, *arXiv: 1811.03604*.

[6] D. Leroy, A. Coucke, T. Lavril, T. Gisselbrecht, and J. Dureau, "Federated learning for keyword spotting," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2019, pp. 6341–6345.

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

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.

[9] A. Krizhevsky, "Learning multiple layers of features from tiny images," Dept. Comput., Univ. Toronto, Toronto, Canada, Tech. Rep., 2009.

[10] H. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2017, pp. 1273–1282.

[11] S. Liu, J. Yu, X. Deng, and S. Wan, "FedCPF: An efficient-communication federated learning approach for vehicular edge computing in 6G communication networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 2, pp. 1616–1629, Feb. 2022.

[12] R. Pathak and M. J. Wainwright, "FedSplit: An algorithmic framework for fast federated optimization," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2020, pp. 7057–7066.

[13] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *J. Netw. Comput. Appl.*, vol. 116, pp. 1–8, 2018.

[14] A. Abedi and S. S. Khan, "FedSL: Federated split learning on distributed sequential data in recurrent neural networks," 2020, *arXiv:2011.03180*.

[15] C. Thapa, P. C. M. Arachchige, S. Camtepe, and L. Sun, "Splitfed: When federated learning meets split learning," in *Proc. AAAI Conf. Artif. Intell.*, 2022, pp. 8485–8493.

[16] J. Jeon and J. Kim, "Privacy-sensitive parallel split learning," in *Proc. Int. Conf. Inf. Netw.*, 2020, pp. 7–9.

[17] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of FedAvg on Non-IID data," 2019, *arXiv:1907.02189*.
[18] Y. Fraboni, R. Vidal, and M. Lorenzi, "Free-rider attacks on model aggregation in federated learning," in *Proc. Int. Conf. Artif. Intell. Statist.st.*, 2021, pp. 1846–1854.
[19] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *MLSys*, pp. 429–450, 2020.
[20] Y. Gao et al., "End-to-End evaluation of federated learning and split learning for Internet of Things," in *Proc. Int. Symp. Reliable Distrib. Syst.*, 2020, pp. 91–100.
[21] H. Zhou, W. Zhang, C. Wang, X. Ma, and H. Yu, "BBNet: A novel convolutional neural network structure in edge-cloud collaborative inference," *Sensors*, vol. 21, 2021, Art. no. 4494.
[22] A. E. Eshratifar, A. Esmaili, and M. Pedram, "Bottlenet: A deep learning architecture for intelligent mobile cloud computing services," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Des.*, 2019, pp. 1–6.
[23] Y. Matsubara, M. Levorato, and F. Restuccia, "Split computing and early exiting for deep learning applications: Survey and research challenges," *ACM Comput. Surv.*, vol. 55, no. 5, pp. 1–30, Dec. 2022.
[24] Y. Matsubara, S. Baidya, D. Callegaro, M. Levorato, and S. Singh, "Distilled split deep neural networks for edge-assisted real-time systems," in *Proc. Workshop Hot Top. Video Analytics Intell. Edges*, 2019, pp. 21–26.
[25] V. Turina, Z. Zhang, F. Esposito, and I. Matta, "Federated or split? A performance and privacy analysis of hybrid split and federated learning architectures," in *Proc. IEEE 14th Int. Conf. Cloud Comput.*, 2021, pp. 250–260.
[26] K. Singhal, H. Sidahmed, Z. Garrett, S. Wu, J. K. Rush, and S. Prakash, "Federated reconstruction: Partially local federated learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2021, pp. 11 220–11 232.
[27] J. Yan and J. Yuan, "A survey of traffic classification in software defined networks," in *Proc. 1st IEEE Int. Conf. Hot Inf.-Centric Netw.*, 2018, pp. 200–206.
[28] H. Yang, J. Zhao, Z. Xiong, K.-Y. Lam, S. Sun, and L. Xiao, "Privacy-preserving federated learning for UAV-enabled networks: Learning-based joint scheduling and resource management," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 10, pp. 3144–3159, Oct. 2021.
[29] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2015, pp. 1322–1333.
[30] S. Hidano, T. Murakami, S. Katsumata, S. Kiyomoto, and G. Hanaoka, "Model inversion attacks for prediction systems: Without knowledge of non-sensitive attributes," in *Proc. 15th Annu. Conf. Privacy Secur. Trust*, 2017, pp. 11501–11509.
[31] P. Vepakomma, A. Singh, O. Gupta, and R. Raskar, "NoPeek: Information leakage reduction to share activations in distributed deep learning," in *Proc. Int. Conf. Data Mining Workshops*, 2020, pp. 933–942.
[32] M. Abadi et al., "Deep learning with differential privacy," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 308–318.
[33] Y. Mao, S. Yi, Q. Li, J. Feng, F. Xu, and S. Zhong, "Learning from differentially private neural activations with edge computing," in *Proc. IEEE/ACM Symp. Edge Comput.*, 2018, pp. 90–102.
[34] J. Huang, C. Samplawski, D. Ganesan, B. Marlin, and H. Kwon, "CLIO: Enabling automatic compilation of deep learning pipelines across IoT and cloud," in *Proc. 26th Annu. Int. Conf. Mobile Comput. Netw.*, 2020, pp. 1–12.
[35] S. Qiu, D. Wang, G. Xu, and S. Kumari, "Practical and provably secure three-factor authentication protocol based on extended chaotic-maps for mobile lightweight devices," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 2, pp. 1338–1351, Mar./Apr. 2022.
[36] Q. Jiang, N. Zhang, J. Ni, J. Ma, X. Ma, and K.-K. R. Choo, "Unified biometric privacy preserving three-factor authentication and key agreement for cloud-assisted autonomous vehicles," *IEEE Trans. Veh. Technol.*, vol. 69, no. 9, pp. 9390–9401, Sep. 2020.
[37] J. Li, A. S. Rakin, X. Chen, Z. He, D. Fan, and C. Chakrabarti, "ResSFL: A resistance transfer framework for defending model inversion attack in split federated learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 10184–10192.
[38] Y. Kang et al., "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proc. 22nd Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2017, pp. 615–629.
[39] T. Unterthiner, D. Keysers, S. Gelly, O. Bousquet, and I. Tolstikhin, "Predicting neural network accuracy from weights," 2020, *arXiv:2002.11448*.
[40] K. K. et al., "Lessons learned from the chameleon testbed," in *Proc. USENIX Conf. Usenix Annu. Tech. Conf.*, 2020, pp. 219–233.
[41] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 8026–8037.
[42] T. Ryffel et al., "A generic framework for privacy preserving deep learning," 2018, *arXiv:1811.04017*.
[43] PyGrid, 2020. [Online]. Available: https://github.com/OpenMined/PyGrid
[44] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[45] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "End-to-end encrypted traffic classification with one-dimensional convolution neural networks," in *Proc. IEEE Int. Conf. Intell. Secur. Inform.*, 2017, pp. 43–48.
[46] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, 2012.
[47] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *J. Mach. Learn. Res.*, vol. 18, pp. 6765–6816, 2018.
[48] G. E. Hinton and R. S. Zemel, "Autoencoders, minimum description length and helmholtz free energy," in *Proc. 6th Int. Conf. Neural Inf. Process. Syst.*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993, pp. 3–10.
[49] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "EMNIST: Extending MNIST to handwritten letters," in *Proc. Int. Joint Conf. Neural Netw.*, 2017, pp. 2921–2926.
[50] A. Yousefpour et al., "Opacus: User-friendly differential privacy library in PyTorch," 2021, *arXiv:2109.12298*.

Zongshun Zhang received the BS degree in computer science from the University of Minnesota, Twin City, in 2019. He is currently working toward the PhD degree in computer science with Boston University, advised by Professor Abraham Matta. He is interested in researching cloud resource-orchestration methods, recently focusing on the efficient usage of serverless platforms for ML(Neural Network) algorithms.

Andrea Pinto received the BS and MS degrees in computer engineering from the University of Naples, Federico II, Italy, in 2020. He is currently working toward the PhD degree in computer science since Fall 2021. His research interests include computer networks, network management, and artificial intelligence for network softwarization through technologies such as Software-Defined Networks. Dr. Flavio Esposito is his primary adviser.

Valeria Turina received the MS degree in mathematical engineering from Politecnico di Torino, in 2020. She is a data scientist with Data Reply IT; she mainly deals with Machine Learning and Deep Learning projects applied to Natural Language Processing and Text Analysis. She was a visiting scholar with Computer Science Department, Saint Louis University. Her research interests include focused on efficient decentralized deep learning architectures and privacy-aware algorithms.

Flavio Esposito received the PhD degree in computer science from Boston University, USA, in 2013, and the MS degree in telecommunication engineering from the University of Florence, Italy. He is an associate professor with the Department of Computer Science, Saint Louis University. His research interests include network management, network virtualization, and distributed systems.

Ibrahim Matta (Senior Member, IEEE) received the PhD degree in computer science from the University of Maryland at College Park, in 1995. He is a professor and chair of computer science with Boston University. His research interests include involves network protocols, architectures, and performance evaluation. He has published more than 100 peer-reviewed articles. He received the National Science Foundation CAREER Award in 1997 for his research on QoS routing. He has served as the chair or co-chair of many technical committees, including IEEE 2011 CCW and 2005 ICNP. He is a senior member of ACM.