PCAT: Functionality and Data Stealing from Split Learning by Pseudo-Client Attack

Xinben Gao and Lan Zhang, *University of Science and Technology of China*

https://www.usenix.org/conference/usenixsecurity23/presentation/gao

This paper is included in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

PCAT: Functionality and Data Stealing from Split Learning by Pseudo-Client Attack

Xinben Gao
University of Science and Technology of China
gxb1320276347@mail.ustc.edu.cn

Lan Zhang*
University of Science and Technology of China
zhanglan@ustc.edu.cn

Abstract

Split learning (SL) is a popular framework to protect a client's training data by splitting up a model among the client and the server. Previous efforts have shown that a semi-honest server can conduct a model inversion attack to recover the client's inputs and model parameters to some extent, as well as to infer the labels. However, those attacks require the knowledge of the client network structure and the performance deteriorates dramatically as the client network gets deeper ($\geq$ 2 layers). In this work, we explore the attack on SL in a more general and challenging situation where the client model is unknown to the server and gets more complex and deeper. Different from the conventional model inversion, we investigate the inherent privacy leakage through the server model in SL and reveal that clients' functionality and private data can be easily stolen by the server model, and a series of intermediate server models during SL can even cause more leakage. Based on the insights, we propose a new attack on SL: **P**seudo-**C**lient **AT**tack (PCAT). To the best of our knowledge, this is the first attack for a semi-honest server to steal clients' functionality, reconstruct private inputs and infer private labels without any knowledge about the clients' model. The only requirement for the server is a tiny dataset (about 0.1% - 5% of the private training set) for the same learning task. What's more, the attack is transparent to clients, so a server can obtain clients' privacy without taking any risk of being detected by the client. We implement PCAT on various benchmark datasets and models. Extensive experiments testify that our attack significantly outperforms the state-of-the-art attack in various conditions, including more complex models and learning tasks, even in non-i.i.d. conditions. Moreover, our functionality stealing attack is resilient to the existing defensive mechanism.

1 Introduction

The last decade has seen the flourishing and widespread adoption of deep neural networks (DNNs), which achieve remarkable performance on rich tasks. Split learning (SL) is an emerging learning paradigm proposed to enable a data owner with constrained computing resources or sensitive data to train a large model with a powerful server cooperatively [3, 16, 25, 29, 34, 36, 39, 44, 45]. It splits a DNN into a client model and a server model, and the client only needs to perform lightweight computations and output intermediate layer activations (smashed data) instead of raw data. Since the server doesn't have access to the client-side model and inputs, SL is considered to be capable of protecting the functionality of the client model and the privacy of inputs from stealing.

In many application areas, like healthcare and finance, the data and labels from clients can be valuable and sensitive, and AI services built on these data are often very lucrative. Therefore, attackers and even the server have a strong incentive to steal the sensitive data and functionality of the client model. In scenarios where a compute provider (the server) and a data provider (the client) collaborate to train a SL model for profit, stealing the functionality of the client model allows the server to get rid of the client and perform the inference on its own without sharing the revenue earned by the SL model. Some recent works [9, 19, 33] have presented a series of attacks on SL. Feature-space hijacking attack (FSHA) [13, 33] points out that the server can hijack clients to leak important features about the private dataset for the server to uncover private inputs. But the FSHA server is malicious since it completely disrupts the process of SL, making this attack detectable by clients [8]. Besides, the server can't steal the functionality of the client model due to the damaged SL model. UnSplit [9] is the state-of-the-art attack designed for a semi-honest server, which requires the server to know the client model structure and the smashed data, and conducts a model inversion attack to reconstruct the client model and raw inputs. It searches over the space consisting of all possible input values and client network's parameters by a coordinate gradient descent approach. However, since both the model parameters and the inputs are unknown, the search space could be too large to converge. Once the learning task and the client model become slightly complex, its performance deteriorates dramatically and the

*Corresponding Author.

attack fails. For example, when training VGG16, the test accuracy of the stolen functionality by UnSplit drops to 22.1% on CIFAR-10 when the client model has two layers (see Tab. 5). As for label inference attack, the norm-based label uncovering method [28] uses the norm of gradients to infer labels, but it only suits the imbalanced binary classification. UnSplit [9] is also based on the gradients' information, but it only works under the strong assumption that the client top model has only a depth of one (see Tab.9). In short, for a semi-honest server, existing attacks on SL mainly based on the idea of model inversion, which requires the knowledge of the client model structure and could fail due to the large search space as the client model gets deeper and more complex. Therefore, how to effectively steal the functionality, inputs and labels of the client in an undetectable way, even if the client model's structure is unknown and complex, is still an open question.

To explore the answer to this question, in this work, we propose a novel and widely effective attack paradigm — Pseudo-Client ATtack (PCAT), to steal the functionality and data from the client. As aforementioned, conventional attacks first obtain smashed data and client model structure, and then invert the client model. Differently, the effectiveness of PCAT is based on our insights that a well-trained server model itself can provide sufficient information to construct a pseudo-client model to steal client's functionality by using very limited training samples, even without using the smashed data. Moreover, we discover that a series of intermediate server models during normal SL can provide extra knowledge to the pseudo-client model to gain closer functionality. With these insights, our main idea is that the server can take full use of the knowledge learned by evolving server models to train a pseudo-client model to gain a functionality as close to that of the real client as possible, whose structure can be completely different from that of the real client model. Hence, PCAT doesn't require any knowledge about the real client model. The only assumption is that the server can obtain a small dataset of few samples for the same learning task from any public sources, whose size is orders of magnitude smaller than the private training set. Once the pseudo-client model learns a mapping from inputs to the feature space of smashed data, the server can transform the given smashed data back to raw inputs by learning a reverse mapping. It is also able to replace the top model by a pseudo-top model to infer private labels.

The main contributions of this paper are summarized as follows:

•**New attack:** We propose a novel pseudo-client attack on SL, which, to the best of our knowledge, is the first attack enabling a semi-honest server to achieve three goals — functionality stealing, input data reconstruction and labels inference, without any preknowledge about the client model. Compared with previous attacks on SL, our attack has the following advantages: 1) It is widely effective since it suits all variants of SL and doesn't require any knowledge of the client model; 2) It is effective in cases with more complex tasks and client models; 3) It is hard to detect since it's transparent to clients; 4) Our functionality stealing attack doesn't require smashed data, thus it is robust even if clients use defensive mechanisms like nopeek [45].

•**New insights:** We provide the insights that a server model in SL contains rich private information, though it doesn't have access to any private data directly. A trained server model can be utilized to construct a pseudo-client model to gain the functionality of the real client, even without using any smashed data. Moreover, a series of intermediate server models can "guide" the pseudo-client model to reach a better performance. Thus, on the basis of the insights, we design PCAT that significantly outperforms state-of-the-art attacks in three attack goals. Our successful attacks also reveal the high risk of leaking privacy through the server model in SL, even when the client model structure and smashed data are protected.

•**New results:** We implement PCAT on various benchmark datasets and models to verify its effectiveness. The results show that the server can use a very small dataset to gain a pseudo-client model whose functionality very close to the real client. When the dataset size is only 0.1% - 1% of the private dataset and the client model has one layer, the accuracy gap between the PCAT pseudo-client model and victim model is only 0.20%, 2.10%, and 1.33% on MNIST, CIFAR-10, and Tiny ImageNet respectively; while UnSplit produces the client model with 4.25% and 26.31% accuracy gap on MNIST and CIFAR-10. When dataset size is only 1% - 5% of the private dataset and the client model has two layers, the accuracy gap of PCAT is only 1.09%, 4.78%, and 3.49% on three datasets respectively; while the accuracy gap of UnSplit significantly deteriorates to 34.70% and 48.88% on MNIST and CIFAR-10. For MobileNet on Tiny ImageNet, when the client model has four layers, the accuracy gap achieves 10%, indicating the attack still works with acceptable accuracy. Even the client uses a defence technique like Nopeek [45], it has little influence on the effectiveness of PCAT. The server can use this pseudo-client model to reconstruct raw inputs more precisely than any other previous attack for a semi-honest server. For the U-shape SL, the label inference accuracy of PCAT achieves 96.58% on MNIST with LeNet-5 layer2, while UnSplit has only 9.1% accuracy. Extensive evaluations show that our attack is also robust to non-i.i.d. situations when the server lacks training samples of some classes. The server can reconstruct the raw inputs of the missing classes though it doesn't have any preknowledge about these classes.

2 Preliminaries and Related Work

2.1 Split Learning

As a rising paradigm of distributed machine learning, split learning (SL) [16,34,44] is proposed for resource-constrained data owners by letting the powerful server undertake the majority of computation. In SL, the ML model (usually a neural network) is split into several parts. Without loss of gener-

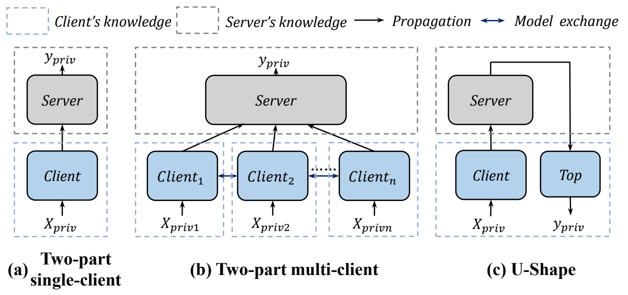


Figure 1: Three typical variants of SL [29, 44], as presented in Fig.1. Our attack mechanism is effective against all three variants.

ality, the model is partitioned to a server model (f_s) and a client model (f_c), and the function of the whole model is $f = f_s(f_c(\cdot))$. The client model is allocated to the data owners, and the server model is at the place concentrating sufficient computing power. During the overall training phase, the server should be oblivious to any private information of the client.

There are three typical settings for SL [29, 34, 44], as presented in Fig.1. In the single-client setting, the client transmits smashed data ($f_c(X)$) and labels to the server, and the server computes loss and gradients. Then the server performs backward propagation and sends gradients back to the client. Receiving the gradients, the client performs backward propagation, and both the server and client update their weights. It iterates in this way until the model converges. For the multi-client setting, the clients take part in training in a round-robin sequence. Each client updates its local model from the last client before training. As for the U-shape setting, a top model is split out and assigned to clients to protect private labels and clients need to calculate the loss.

To improve its practicality, a variety of extensions have been carried out. A combination of federated learning and SL called SplitFed [41] can combine these two frameworks' strengths and complement their weaknesses. SL is also applied to other kinds of neural networks such as GNN [17], RNN [2] and Transformers [32]. Due to its small computation resource consumption, SL is widely used in IoT scenarios [4, 12, 31, 42, 47].

2.2 Attacks on Split Learning

With many systems increasingly relying on machine learning, a variety of attacks on machine learning models are emerging rapidly [5, 35, 37]. Most attacks aim at stealing the functionality of the model and the privacy of the training data. When machine learning as a server (MLaaS) engines provide APIs for clients to query, based on the input-output pairs by querying the black-box model, the attacker can train a knockoff net with the functionality close to the victim model even if the attacker doesn't know the victim model's structure and hyperparameters [43, 46]. Orekondy et al. [30] propose to use a modification of knowledge distillation, by which the victim model acts as a "teacher" and provides hard labels to the knockoff net to perform a functionality stealing attack. Model inversion attacks [10, 11] and attribute inference attacks [14, 15, 23] reconstruct inputs or sensitive features given the outputs of the target model. Those attacks are designed to steal the functionality or privacy of a complete model, thus can not be directly adopted to attack the client model in SL.

In SL, recently, several attacks [9, 19, 28, 33] are designed for the server to reconstruct the clients' raw inputs, model parameters and labels. In FSHA [33], a malicious server can hijack clients to leak essential features about their raw data. However, due to that the server changes the learning task and disrupts the learning process, the malicious behaviors can be detected by clients [8] and the attacker can not steal clients' functionality. Thus, FSHA achieves fewer attack goals than PCAT. UnSplit [9] provides the state-of-the-art attack on SL based on the assumption that the server has the client's model structure, and a semi-honest server can reconstruct the raw inputs and parameters of the clients' model by utilizing the smashed data. Such a reconstructed model can work like a clone model of the real client model. Since both the model parameters and the inputs are unknown, the solution space could be too large to converge once the client's model or the learning task becomes more complex. Therefore, in practice this attack can easily fail when any of the following situations occur: 1) the structure of the client model is unknown; 2) the client model has more than one layer; 3) the learning task is complex; 4) the smashed data is protected by defense mechanism like nopeek [45]. When attacking the U-shape SL, both norm-based label-uncovering method [28] and UnSplit [9] use gradients information to reconstruct private labels. However, these two attacks are only effective for the imbalanced binary classification setting or the setting where the top model contains only one layer. Deep Leakage from Gradients (DLG) [50] is another attack for centralized learning, which requires the model structure and calculates loss of gradients to recover the inputs. Differently, PCAT doesn't need the client model structure and generates a similar client model to steal the client's functionality and further reconstruct its inputs and labels.

As a summary, existing attacks on SL mainly based on the idea of model inversion, which inherently suffer from the large search space of both possible inputs and client model parameters. Therefore, those attacks are only effective on simple tasks and client models. In this work we consider a more general and challenging problem: how to steal the functionality and data of the client model in an oblivious way when the structure of the client model is unknown, tasks and client models are more complex, and the client may even use some defensive methods to protect the smashed data.

3 Goal and Main Idea

3.1 Goals and Settings

In this work, we aim to design an attack mechanism on SL, which supports a semi-honest server to perform all three attacks: 1) steal functionality of the client model; 2) reconstruct the client's raw inputs; 3) infer labels of the client's inputs. To make the attack mechanism widely effective, we design our mechanism to meet the following requirements.

(1) Minimal knowledge about the client model: the server knows the learning task but it doesn't need to know the structure or hyper parameters of the client model. For the functionality stealing, the sever doesn't even need to use the smashed data, since there exist methods to protect the smashed data [45].

(2) Support more complex client models and tasks: existing methods [9,28] support only simple tasks, like imbalanced binary classification and handwritten digits classification, and simple client models with only one layer. Differently, our mechanism should support more complex tasks, as well as deeper and wider client models.

(3) Effective against three variants of SL: as shown in Fig.1, there are three typical variants of SL. Our attack mechanism should be effective against all of them.

(4) Transparent to the client: the server is semi-honest, and in the client's view, the training process with an attack is indistinguishable from a normal training without an attack.

The assumptions for PCAT are that: 1) Both server and client are semi-honest, which follow the SL protocol, but the server is curious about the client's model, inputs and labels. 2) The server doesn't known anything bout the client model structure. 3) The server can collect a limited number of training samples (X_{server}) for the same learning task from public sources. As the default setting in SL protocols, the server and the client use the same optimizer.

3.2 Insights and Main Ideas

To achieve aforementioned ambitious goals, the major challenge stems from the fact that the server does not know anything about the client model when conducting functionality stealing. This challenge makes all previous query based and model inversion based attacks inapplicable. Therefore, we turn to explore what the server model learns and whether it can be utilized to construct a pseudo-client model, whose functionality is very similar to the real client model. To find the answer, we start by investigating how to steal the functionality of a complete model, followed by the idea to steal the client model in SL.

(1) Steal a Complete Model

Considering a model with a sufficiently large dataset as the victim model, denoted by F, the functionality stealing attack tries to construct a knockoff model using a small dataset,

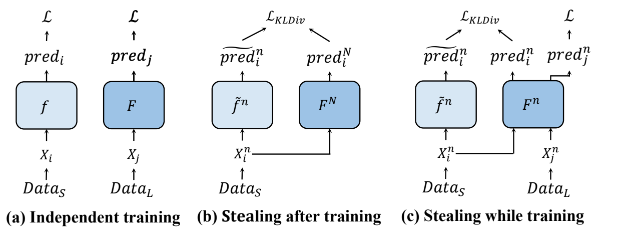


Figure 2: Two strategies to construct a pseudo model $\widetilde{f}$ which steals the functionality of the victim model F. F is trained on a large dataset $Data_L$ and $\widetilde{f}$ is trained on a small dataset $Data_S$ with the help of F. N is the total number of iterations during training and $n \in [0,N]$. That is, F^n denotes the model after n iterations and F^N denotes the final model after training. Before calculating $\mathcal{L}_{KLDiv}$, we need to perform $log(softmax(pred/\tau))$ and τ is temperature [20].

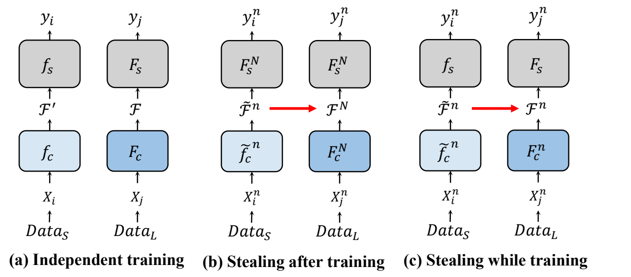


Figure 3: Two strategies to steal the client model in SL. (a) shows training two SL models on small($Data_S$) and large($Data_L$) datasets independently. (b) and (c) illustrate the strategies that train a pseudo-client model $\widetilde{f_c}$ to steal the functionality of the victim client model F_c after and while training the server model, separately. $n \in [0,N]$, F_s^N denotes the final server model after SL and F_s^n denotes the intermediate server model after n iterations during the SL. $\mathcal{F}$ is the feature space of the smashed data. The red arrows indicate that under the restriction of F_s^N or F_s^n, the feature space of the pseudo-client's outputs is learning to get closed to the feature space of the victim client's outputs.

denoted by $\widetilde{f}$, which behaves very much like F. The most critical challenge is how to let F effectively teach $\widetilde{f}$ when the attacker has very limited training data.

•Basic strategy: stealing after training. A basic solution is to train the victim model first, then let the victim model teach the knockoff model in the way of knowledge distillation [20]. As presented in Fig.2(a) and (b), after training the victim model for N iterations on a large dataset, we obtain a well-trained model F^N. Then the knockoff model $\widetilde{f}$ can be trained with the help of F^N in the following way: in the n-th iteration, the inputs (X_i^n) from a small dataset are fed to both models, and the learning targets of $\widetilde{f}$ are the output soft labels of F^N rather than the hard labels. The loss function is

$$\mathcal{L}_{KLDiv} = KLDivergenceLoss(soft(\widetilde{pred}_i^n), soft(pred_i^N)), \tag{1}$$

Algorithm 1: Stealing Client after Training Server (Vanilla-PCAT)

```
Data: Server's data: $(X_{server}, y_{server})$, total epochs: $N$
/* Initialize models */
$F_s^N$ is a well-trained server model;
$\widetilde{f_c}$ is randomly initialized;
while $n < N$ do
    Randomly select $(X_i, y_i)$ from $(X_{server}, y_{server})$;
    $\widetilde{smashed} \leftarrow \widetilde{f_c}(X_i)$;
    $\widetilde{z} \leftarrow F_s(\widetilde{smashed})$;
    $\widetilde{\mathcal{L}} \leftarrow \mathcal{L}(\widetilde{z}, y_i)$;
    $\widetilde{\nabla}_{smashed} \leftarrow$ compute_gradient$(\widetilde{smashed}, \widetilde{\mathcal{L}})$;
    $\widetilde{\nabla}_c \leftarrow$ compute_gradient$(\widetilde{f_c}, \widetilde{\nabla}_{smashed})$;
    $\widetilde{f_c}' \leftarrow$ update_weight$(\widetilde{f_c}, \widetilde{\nabla}_c)$;
    /* The weight of $F_s$ isn't updated. */
end
```

which is the KL divergence loss between the soft labels of two models.

By learning from the victim model, the performance of the knockoff model can often be obviously improved, e.g., from 73.52% to 82.06% in the handwritten digits classification task in Fig.4(a), when the attacker has only 100 training samples (10 samples per class). However, there is still a significant gap from the performance of the victim model, which is 98.82%.

•**Our improved strategy: stealing while training.** To further improve the performance of the knockoff model, we have the observation that a series of intermediate victim models during training can provide essential information, which can teach the knockoff model better than the final well-trained victim model does. Based on this observation, different from traditional knowledge distillation, we propose to steal the victim model while it is being trained. As illustrated in Fig.2(c), in the n-th iteration, taking the same inputs from the small dataset, the optimization objective of $\widetilde{f^n}$ is to minimize the KL divergence loss between the output soft labels of two current models. Note that, F^n also takes inputs from the large dataset, and its optimization objective remains the loss on the hard labels. In this way, both F^n and $\widetilde{f^n}$ are trained synchronously. F^n evolves by using the large dataset, while $\widetilde{f^n}$ uses a sequence of evolving targets to train itself. Although at the beginning, learning targets are not as accurate as the final target, actually the evolving learning targets can "guide" $\widetilde{f^n}$ to converge more precisely to the final target. As plotted in Fig.4(a) and Fig.4(b), our stealing while training strategy increases the accuracy of the knockoff model to 88.44%, which is significantly higher than the stealing after training strategy.

(2) Steal a Client Model in Split Learning

The aforementioned strategies cannot be directly applied to steal a client model in SL. Different from stealing a complete

Algorithm 2: Stealing Client while Training Server (PCAT)

```
Data: Server's data: $(X_{server}, y_{server})$, Client's data:
      $(X_{priv}, y_{priv})$, epochs: $N$
/* Initiate models */
$F_s, F_c, \widetilde{f_c}$ are all randomly initialized and $F_c \neq \widetilde{f_c}$;

For the client:
while $n < N$ do
    Randomly select $(X_j, y_j)$ from $(X_{priv}, y_{priv})$;
    $Smashed \leftarrow F_c(X_j)$;
    send_to_server$(Smashed, y_j)$;
    recv_from_server$(\nabla_{Smashed})$;
    $\nabla_c \leftarrow$ compute_gradient$(F_c, \nabla_{Smashed})$;
    $F_c' \leftarrow$ update_weight$(F_c, \nabla_c)$;
end

For the server:
while $n < N$ do
    recv_from_client$(Smashed, y_j)$;
    Select $X_i$ from $X_{server}$ so that $y_i = y_j$  // Align
        labels
    $\widetilde{Smashed} \leftarrow \widetilde{f_c}(X_i)$;
    $\widetilde{z} \leftarrow F_s(\widetilde{Smashed})$;
    $z \leftarrow F_s(Smashed)$;
    $\widetilde{\mathcal{L}} \leftarrow \mathcal{L}(\widetilde{z}, y_i)$;
    $\mathcal{L} \leftarrow \mathcal{L}(z, y_j)$;
    $\nabla_s \leftarrow$ compute_gradient$(F_s, \mathcal{L})$;
    $\widetilde{\nabla}_{Smashed} \leftarrow$ compute_gradient$(\widetilde{Smashed}, \widetilde{\mathcal{L}})$;
    $\nabla_{Smashed} \leftarrow$ compute_gradient$(Smashed, \mathcal{L})$;
    $\widetilde{\nabla}_c \leftarrow$ compute_gradient$(\widetilde{f_c}, \widetilde{\nabla}_{Smashed})$;
    $\widetilde{f_c}' \leftarrow$ update_weight$(\widetilde{f_c}, \widetilde{\nabla}_c)$;
    send_to_client$(\nabla_{Smashed})$;
    $F_s' \leftarrow$ update_weight$(F_s, \nabla_s)$;
    // $F_s$ updates weight using grad from
        SL.
end
```

model, to construct a pseudo-client model, we still need to address the following challenges: 1) the server cannot obtain the intermediate client models during training, but only knows all intermediate server models; 2) the server cannot obtain the inputs nor the output soft labels, and even the smashed data cannot be used as aforementioned in Section 3.1; 3) the server cannot even feed samples from its small dataset to the client model, otherwise the client will be aware of the attack.

Facing these challenges, w.l.o.g., we further analyze the single-client SL. As illustrated in Fig.3(a), the client model maps raw inputs X to a certain feature space $\mathcal{F}$. Then the server model maps intermediate activation from this feature

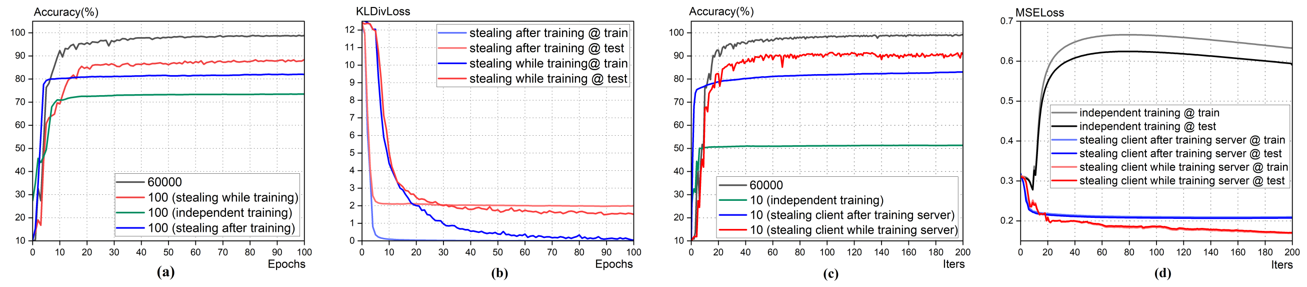


Figure 4: Performance of victim models and pseudo models obtained by different strategies. It uses LeNet-5 on MNIST [49] dataset. (a) and (b) show the performance when stealing a complete model after and while the training of the victim model. (c) and (d) show the performance when stealing a client model after and while the training of the server model in SL. The split layer is 2. MSELoss is calculated between the smashed data output by the client model and the pseudo-client model, which characterizes the difference between $\widetilde{\mathcal{F}}^N$ and $\mathcal{F}^N$. 60000, 100 are the amounts of training samples.

space to logits. The SL model trained on a large dataset, denoted by $F = F_s(F_c(\cdot))$, usually performs much better than the SL model trained on a small dataset, denoted by $f = f_s(f_c(\cdot))$. As shown in Fig.4(c), with 60,000 training samples, the SL model achieves 99.18%, while the accuracy is only 51.33% with 10 training samples (1 sample per class). Since the server aims to construct a pseudo model with good performance, it must make full use of the knowledge in the client's large dataset. Therefore, we propose to steal the functionality of the client model by using only the server model(s) trained on the client's dataset and a small dataset from the server itself. As we will present below, such strategies work surprisingly well, even though the server does not know the structure or input of the client model and does not query the client or use the smashed data at all.

•**Our basic strategy for SL: stealing client after training server.** As illustrated in Fig.3(b), the server conducts the normal SL first. After N iterations, the input X is mapped to the feature space $\mathcal{F}^N$ by the victim client model F_c^N. The server model F_s^N is well-trained and capable to convert the smashed data in $\mathcal{F}^N$ to logits. Now the server can connect a pseudo-client model ($\widetilde{f_c}$) to the trained server model F_s^N, and use its small dataset to train ($\widetilde{f_c}$) while fixing the parameters of F_s^N. The pseudo-client model actually maps inputs to another feature space $\widetilde{\mathcal{F}}^N$. To steal the functionality of the victim client model F_c^N, the training algorithm for the pseudo-client model should optimize its output smashed data as close to $\mathcal{F}^N$ as possible, so that the smashed data can be classified by the server model correctly. The detailed training algorithm is presented in Algorithm 1. Note that, it is not necessary for $\widetilde{f_c}$ to have the same structure as the victim client model (F_c), as long as it can learn the mapping from raw inputs the to feature space. Results in Fig.4(c) and (d) show that, this strategy can increase the accuracy of the server's pseudo model from 51.33% to 83.05% when the server has only 10 samples.

•**Our advanced strategy for SL: stealing client while training server.** Similar to the advanced strategy when stealing a complete model, we find that using a sequence of intermediate server models F_s^n, $n \in [0, N]$ during SL, the server can train the pseudo-client model gradually to achieve better performance. In another words, the pseudo-client model can leverage a series of learning targets { $\mathcal{F}^0$, ..., $\mathcal{F}^N$} to steal the functionality more accurately. This strategy is feasible since the server knows the intermediate server model for every iteration. Fig.4(c) testifies that using evolving server models can greatly improve the pseudo-client model's accuracy, from 83.05% to 90.7% in our experiments. Fig.4 (d) further illustrates that the adversarial feature space $\widetilde{\mathcal{F}}^N$ is very close to the victim feature space $\mathcal{F}^N$ by this strategy. Note that, without the constraints from the well-trained server models, the feature space of smashed data in an independently trained pseudo model is quite far away from the victim feature space.

Summaries: We have insights that a server model trained by standard SL can provide sufficient knowledge to train a well-performing pseudo-client model using very limited training samples; a series of intermediate server models during SL can significantly improve its accuracy. These insights not only inspire us to design an highly effective attack mechanism against SL, but also reveal the privacy threats posed by the server model in SL. Based on these insights, our main idea to steal the functionality of the client model is to train a pseudo-client model with any structure that can map inputs X to the target feature space $\mathcal{F}^N$. Once the pseudo feature space $\widetilde{\mathcal{F}}^N$ is sufficiently close to $\mathcal{F}^N$, the pseudo-client model can take place of the victim client model. Then, using the pseudo-client model and smashed data of client's inputs, the server can reconstruct the private inputs of the client by learning a reverse mapping, as well as infer their labels.

4 Pseudo-Client Attack

In this section, following our main idea, we present the detailed design of our attack mechanism, the Pseudo-Client ATtack (PCAT), on different variants of SL. The whole attack process is illustrated in Fig. 5. We also present some details that makes PCAT more effective.

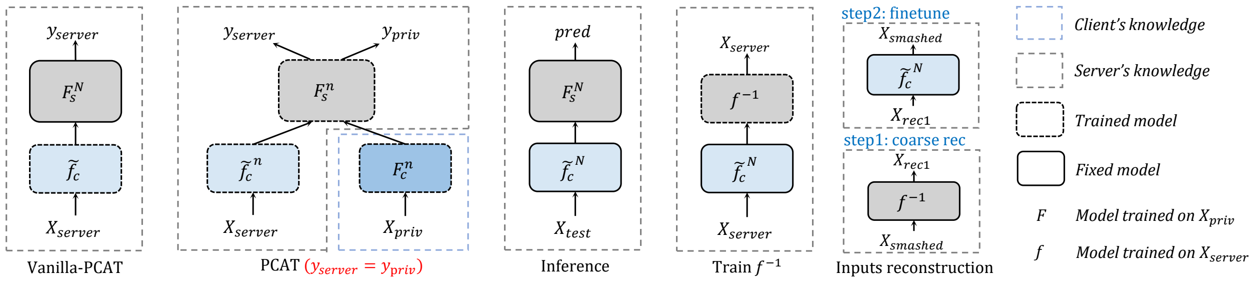


Figure 5: Pseudo-client attack in two-part SL. The server constructs a pseudo-client model $\widetilde{f_c}^N$ by the PCAT algorithm. With $\widetilde{f_c}^N$, the server can perform inference without the involvement of the victim client. During the normal SL, the server stores smashed data (denoted as $X_{smashed}$) of the victim client. Using $\widetilde{f_c}^N$, the server can train an reverse mapping f^{-1} and reconstruct the client's private inputs X_{priv} in two steps.

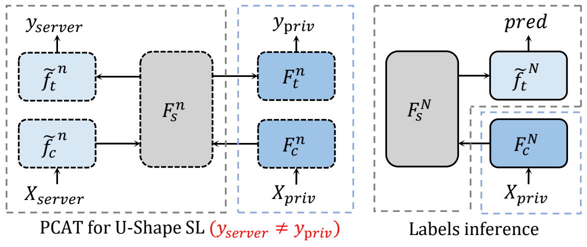


Figure 6: Pseudo-client attack in U-Shape SL. The legends are the same as that in Fig.5. Here, we omitted the methods for the server to perform inference alone and reconstruct input data, since which are the same as that in two-part SL.

4.1 Functionality Stealing

Algorithm 1 has already illustrated how to train a pseudo-client model after the server model has been trained in a normal SL. To further improve the performance of the pseudo-client model, our insights guide us to take full use of a series of intermediate server models. Specifically, the server initializes the pseudo-client model before the SL starts and then trains both pseudo and real client models simultaneously. For each iteration, the victim client performs its forward propagation and sends the smashed data and labels to the server. Then the server selects training samples from its own dataset with the same labels as those uploaded by the client, i.e., $y_{server} = y_{priv}$, and feeds these samples into the pseudo-client model to obtain the smashed data. For both pseudo and victim client models, the server model takes their respective smashed data as input, performs forward propagation and calculates the loss, separately. For SL, the server computes the gradients of smashed data and sends the gradients back to the victim client. The victim client calculates the local gradients and updates the client model. For pseudo-client training, based on the loss, the server calculates the gradients of its own smashed data and the pseudo-client model, with which the pseudo-client model is updated. At the end of each iteration, the server model is updated using only the gradients from normal SL with the victim client. Algorithm 2 presents the details of the attack process. Since both the victim client model and the server model are updated based on only the gradients of normal SL, the whole training process is the same as a normal SL in the client's view. Therefore, this attack is transparent to clients.

In multi-client SL, each client participates in a round-robin mode to train the client model, so all clients obtain the same client model after SL. From the server's perspective, the evolution of the server and client model is the same as that in single-client SL. Therefore, the server can apply PCAT to multi-client SL directly without any modification.

4.2 Inputs Reconstruction

After stealing the functionality, the server obtains a pseudo-client model that maps inputs to a feature space of smashed data, which is very close to the real feature space in SL. Given smashed data from the victim client ($X_{smashed}$), the server can reconstruct the private raw inputs (X_{priv}) by reversing the mapping. As presented in Fig.5, we propose to reconstruct the client's inputs using the following steps:

(1) Train a reverse mapping f^{-1}: since the server has a few training samples, it can train a reverse mapping f^{-1} to map smashed data from the feature space back to the input space. Specifically, the server maps raw training samples to smashed data by using the pseudo-client model, then trains f^{-1} to map them back to raw inputs. The reverse mapping is composed of transposed convolution and upsample layers to transfer smashed data from low resolution to high resolution. The specific reverse mapping we use for every model splitting is shown in Table 14 in Appendix 7.

(2) Coarse-grained reconstruction and fine-tuning: the server can feed $X_{smashed}$ into f^{-1} to obtain the reconstructed inputs(X_{rec}). However, due to the insufficient training samples in server, X_{rec} is usually coarse-grained. To achieve more precise reconstruction, we design a fine-tuning method which takes the coarse-grained X_{rec} from f^{-1} as inputs and the real smashed data $X_{smashed}$ as learning targets. Towards the targets,

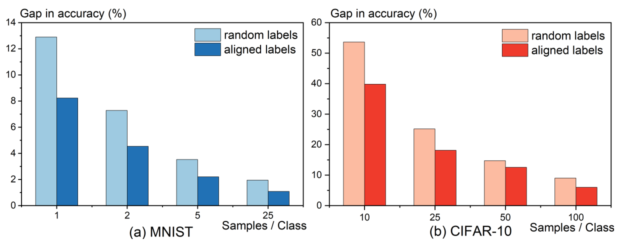


Figure 7: Functionality gap between the pseudo model and the victim model, which is measured by the inference accuracy decrease of $F_s(\widetilde{f_c}(\cdot))$ compared with $F_s(f_c(\cdot))$. The experiment is performed on MNIST [49] and CIFAR-10 [26] dataset.The inference accuracy of the baseline $F_s(f_c(\cdot))$ is 99%/93.2% for MNIST/CIFAR-10."Samples/Class" means the number of samples per class in the server's dataset.

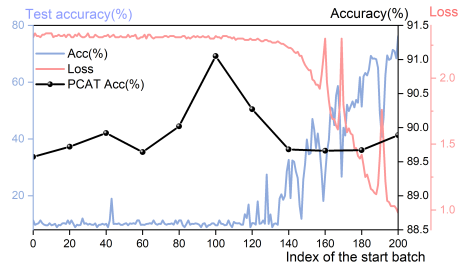


Figure 8: Inference accuracy of the pseudo model $F_s(\widetilde{f_c}(\cdot))$ by skipping batches in the early training. The experiment is performed on MNIST [49] dataset, where the server has one sample per class. The pink and light blue lines denote the test accuracy and loss of SL baseline in the first epoch . The black dots mean the final test accuracy of $F_s(\widetilde{f_c}(\cdot))$ if $\widetilde{f_c}$ starts to train from a certain batch idx.

the server optimizes and fine tunes X_{rec} with fixed pseudo-client model $\widetilde{f_c}^N$, until the output smashed data of $\widetilde{f_c}^N$ is close enough to $X_{smashed}$. In this way, we can obtain a more fine-grained reconstruction.

Note that, we steal the functionality of the client model in a totally different way from UnSplit, thus the input reconstruction is also different from UnSplit. Specifically, UnSplit adopts the client model structure and smashed data to search the client model parameters and inputs simultaneously. While PCAT first learns a pseudo-client model using only the server model and a small training dataset of the server, and then learns a reverse mapping with the server's training data. The smashed data are used for fine-tuning. Since PCAT reconstructs the inputs after the pseudo-client model is well constructed, the search space is significantly smaller than that of the UnSplit. Our experiments verify that PCAT reconstructs raw inputs more precisely than UnSplit.

4.3 Attack on U-Shape Split Learning

In U-shape SL, as shown in Fig. 6, there are a bottom model F_c and a top model F_t placed on the client side. The server needs to train a pseudo-client model $\widetilde{f_c}$ as well as a pseudo-top model $\widetilde{f_t}$. By using the same strategy in two-part SL (i.e., Algorithm 2), the server can obtain $\widetilde{f_c}^N$ and $\widetilde{f_t}^N$ and perform inference alone. With $\widetilde{f_c}^N$, the private inputs of the client can also be reconstructed by the methods in Section 4.2.

Unlike two-part SL where clients send data labels to the server, in U-shape SL labels are considered as privacy and protected from the server by the top model. Since PCAT can construct a pseudo-top model to replace the real top model, the server can feed smashed data of the victim client to the pseudo-top model to infer their private labels.

4.4 Other Details to Improve PCAT

(1) Aligning labels. When attacking two-part SL by Algorithm 2, the server aligns labels of the training samples for the pseudo and victim client models, i.e., to make $y_{server} = y_{priv}$. For the attack on U-shape SL, due to the invisibility of the client's labels, the server randomly selects X_{server} for each iteration. We measure the functionality of the pseudo-client model with and without label alignment. Fig.7 shows that the inference accuracy gap between the pseudo model $F_s(\widetilde{f_c}(\cdot))$ and the victim model $F_s(f_c(\cdot))$ is only 1.09%/1.95% with/without label alignment on MNIST and 6.01%/9.06% on CIFAR-10, when there are 25/100 samples per class in the server's dataset. The results testify that PCAT can steal the functionality of the victim model with high accuracy; and label alignment can increase the pseudo-client model's accuracy. We believe the reason is that the gradients of pseudo and victim client models depend in their respective inputs and labels. Aligned training samples can make the gradients of two client models closer, thus the optimization directions of F_s and $\widetilde{f_c}$ are more consistent.

(2) Late start. In the early stages of SL, since the victim model hasn't started to converge, the unstable server model could guide the pseudo-client model to a wrong direction. Thus, the pseudo-client model can skip some batches in the early stages to avoid being misled and gain a better performance. The experiment results in Fig.8 show that a proper late start can improve the performance of the pseudo model. For example, when the pseudo model training starts from the 100-th batches, the inference accuracy can be raised from 89.57% to 91.05%. But the training should not start too late, otherwise the victim model has already begun to converge and the pseudo model will miss a portion of guidance information. The server should starts to train the pseudo-client model at the time when the test accuracy of the victim model starts to rise and the loss starts to fall. Note that, even if the start time is not optimal, the pseudo-client model still performs better than that trained after the SL is over (i.e., the pseudo-client model trained by using vanilla-PCAT).

5 Implementations

To demonstrate the effectiveness and practicality of PCAT, we implemented PCAT for a variety of learning tasks and models, as well as different settings of split learning.

5.1 Datasets and Models

We implemented PCAT for the following cases with different popular models and benchmark datasets, which cover situations where the models and tasks are simple or complex.

	Dataset	Model	Param	Complexity
1	MNIST	LeNet-5	61.5K	Low
2	CIFAR-10	VGG16	14.6M	Medium
3	Tiny-Imagenet	MobileNet	28.5M	High

Table 1: The settings for all cases and their complexity comparison.

5.2 Data Processing

We assume that the server's and the client's datasets should be prepared for the same learning task. We consider two typical cases: 1) the server's dataset is a subset of the client's private dataset, i.e., $X_{server} \subset X_{priv}$; 2) the server's dataset has no intersection with the client's private dataset, i.e., $X_{server} \bigcap X_{priv} = \emptyset$.

For training, we randomly divide the training set of each benchmark dataset into a public dataset X_{pub} and a private dataset X_{priv}, and $X_{pub} : X_{priv} = 1 : 9$. X_{priv} is allocated to victim clients as the training set of the split learning. X_{pub} is a public dataset that the server can retrieve some samples from it to compose its training set X_{server}. If not specified, we make X_{server} and X_{priv} i.i.d. We will also analyze the effectiveness of PCAT in non-i.i.d. cases. To evaluate the generalizability of PCAT, we also let the server use datasets(X_{dif}) very different from X_{priv}. In our ablation study, X_{dif} is a subset of Imagenet [6] while X_{priv} is from CIFAR-10. The samples in X_{dif} have the same labels as the labels in CIFAR-10. For testing, we use the original testing sets to evaluate models.

5.3 Model Splitting

We adopt LeNet-5 [27], VGG16 [38], MobileNet [21] to validate the effectiveness of our attack. We consider various model splitting strategies, as shown in Figure 9. For two-part split learning, each model split into 2 parts. We split MobileNet from 1 to 4 layers to show that PCAT is robust to the cases that client's model is extreme complex. For U-shape split learning, the client has one or two bottom layers and one or two top layers, while the intermediate layers are allocated to the server.

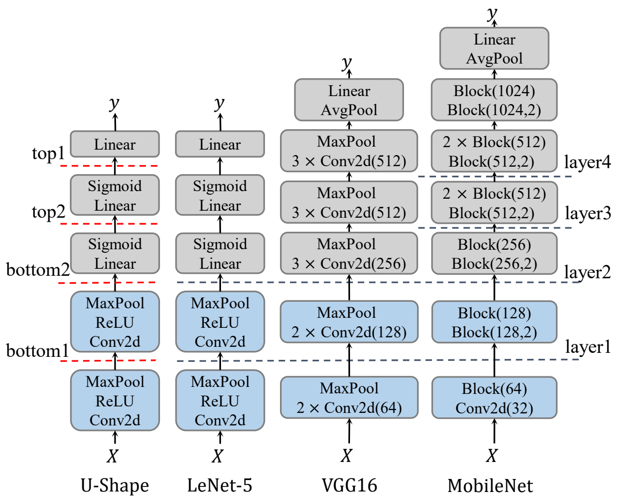


Figure 9: Model splitting strategies. We omit batchnorms and ReLU in VGG16 and MobileNet.

6 Experimental Results

Based on the implementation in Section5, we conduct comprehensive experiments to demonstrate the effectiveness of PCAT on three attack goals in Section3.1, including stealing the functionality of the client model, reconstructing the private inputs, and inferring the labels. The results of successful attacks on different models and datasets testify that the broad applicability of PCAT.

6.1 Functionality Stealing

(1) PCAT for Two-part SL

•**In i.i.d. settings.** We first launch our pseudo-client attack in the case that X_{server} and X_{priv} are i.i.d. and $X_{server} \subset X_{priv}$. We split each model from layer 2 to make the client model more complex. For MNIST, CIFAR-10 and Tiny ImageNet, the server has 5, 250, and 10 training samples per class. Fig.10 shows the performance of the pseudo model constructed by PCAT and Vanilla-PCAT on three datasets. Both vanilla-PCAT and PCAT effectively steal the functionality of the client model. When the server trains a complete model independently using only its own data X_{server}, its accuracy is only 63.38%, 74.16%, and 13.62% on three datasets, respectively. With the same dataset X_{server}, Vanilla-PCAT achieves 91.67%, 85.62%, and 69.8% accuracy, respectively. PCAT further increases the inference accuracy to 96.89%, 89.48%, and 73.46%, which are very close to the performance of the victim SL model. Vanilla-PCAT converges faster than PCAT, because its training is directly guided by the well-trained server model. Though converging slower, PCAT achieves much better accuracy because of the guidance of evolving server models, which drives the output feature space of the pseudo-client model closer to that of the victim client model.

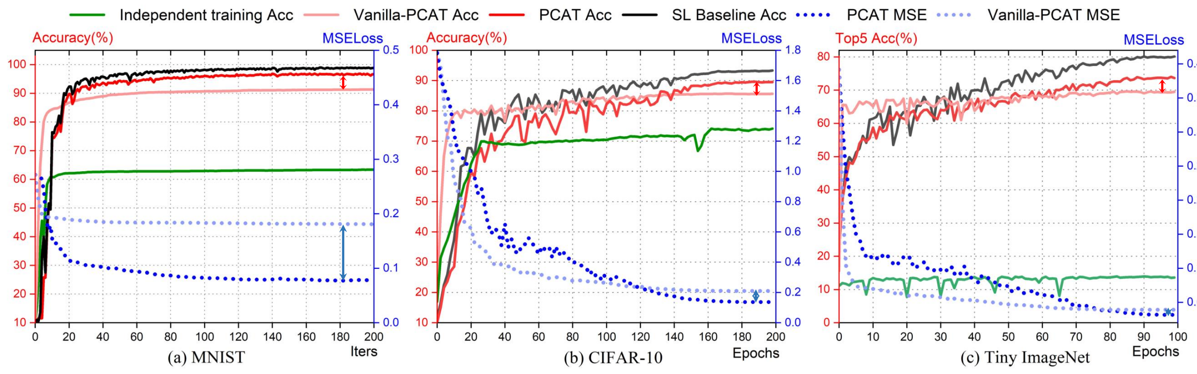


Figure 10: Performance of the pseudo model constructed by PCAT on three datasets. For MNIST, CIFAR-10 and Tiny ImageNet, the server has 5, 250 and 10 training samples per class, respectively. $X_{server} \subset X_{priv}$ and all models split from layer2. Independent training means the server training a complete model using only its own data X_{server}. SL baseline is the victim model trained on X_{priv}. The MSELoss represents the distance between the real feature space and the pseudo feature space.

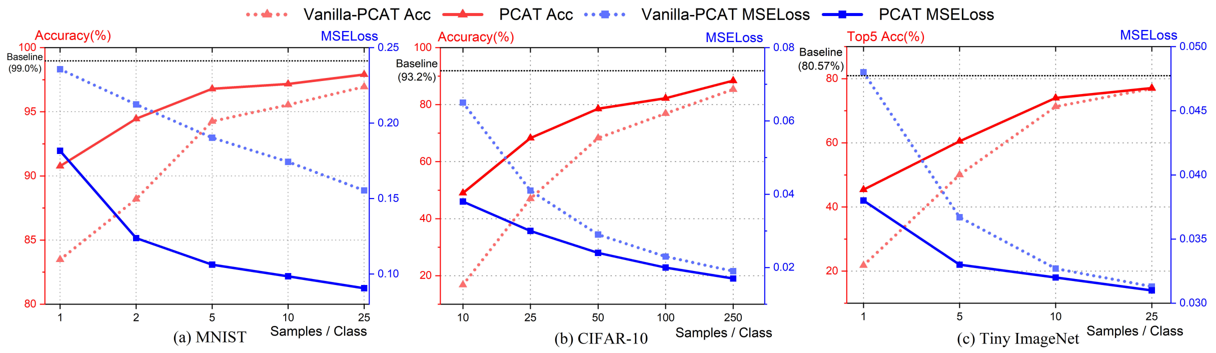


Figure 11: Performance of PCAT changes with the number of training samples owned by the server. All models split from layer2 and $X_{server} \subset X_{priv}$.

Intuitively, the more samples the server has, the better performance PCAT can achieve. Fig.11 illustrates the intuition. On MNIST, when the sample number for each class increases from 1 to 25, the accuracy achieved by PCAT grows from 90.77% to 97.91%. On CIFAR-10, when the sample number for each class increases from 10 to 250, the accuracy achieved by PCAT grows from 49.03% to 89.42%. On Tiny ImageNet, when the sample number for each class increases from 1 to 25, the accuracy achieved by PCAT grows from 45.40% to 77.11%. Compared with the size of the private dataset, which are 54000, 45000, and 90000, (5400, 4500, 450 samples per class) respectively, PCAT requires only a small number of training samples to effectively steal the functionality. And surprisingly, PCAT can achieve a fairly good attack on LeNet-5 even if there is only 1 sample per class.

•**In non-i.i.d. settings.** Now we consider a more common situation that the dataset of the server lacks samples of some classes. In Fig.12, (a) and (b) illustrate the cases that the server lacks one class of samples, e.g., the digit “3” in MNIST and class “deer” (labeled “4”) in CIFAR-10. When the server independently trains a complete model, the model cannot recognize the missing class at all, and the accuracy of related classes also drops significantly. Vanilla-PCAT can recognize most samples in the missing class and achieve 78.91%/51.5% accuracy on MNIST/CIFAR-10, but there is still an obvious gap with the 99.11%/95.2% (on MNIST/CIFAR-10) baseline. PCAT significantly mitigates the effect of the missing class and achieves 94.95%/70.2% accuracy on MNIST/CIFAR-10. Moreover, the overall performance of PCAT is very close to the SL baseline.

(c) and (d) further analyze how the overall accuracy affected by the number of missing classes in the server’s dataset. Compared with independent training, PCAT is much more robust to non-i.i.d. datasets. On MNIST, The accuracy of PCAT is still over 90% even if there are 4 missing classes. On CIFAR-10, PCAT can use only 3 classes to achieve the same performance as Vanilla-PCAT with 9 classes. We think that the reason behind the surprising result is that a subset of classes can represent the mapping functionality of the client model.

•**Ablation study for the influence of X_{server}.** As we mentioned in Section 5.2, we analyze how the difference between the distribution of X_{priv} and X_{server} affects the performance of PCAT. As shown in Fig. 13, we consider four cases: X_{server} is randomly selected from X_{priv}, X_{pub}, half from X_{pub} and half from X_{dif} and all from X_{dif}. The correlations between

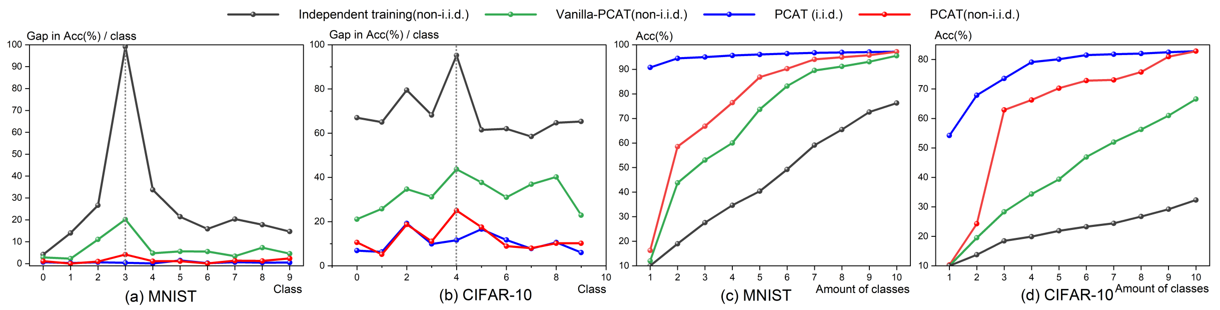


Figure 12: Performance of PCAT in non-i.i.d. settings on MNIST/CIFAR-10. (a) and (b) show the accuracy gap when X_{server} lacks one class ("3"/"4" for MNIST/CIFAR-10). (c) and (d) present accuracy changing with the amounts of classes in X_{server}.

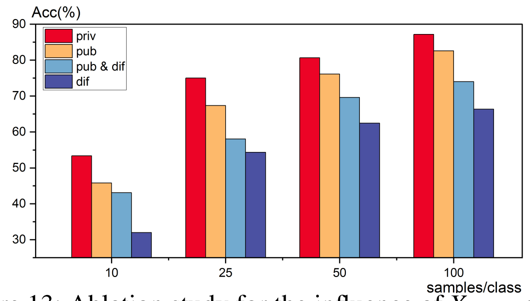


Figure 13: Ablation study for the influence of X_{server} on the performance. X_{priv} is the victim client's private dataset. Both X_{priv} and X_{pub} are from CIFAR-10. X_{dif} is from Imagenet. "priv", "pub", "dif" means that X_{server} is randomly selected from X_{priv}, X_{pub}, X_{dif}. "pub & dif" means that half is from X_{pub} and half is from X_{dif} in X_{server}.

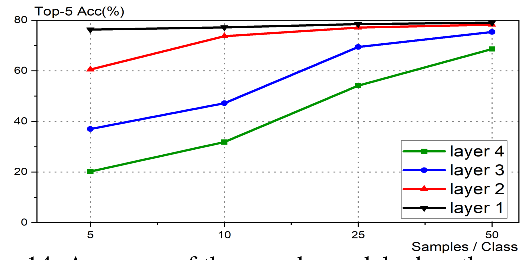


Figure 14: Accuracy of the pseudo model when the model is split from different layers. The model is MobileNet on Tiny ImageNet and $X_{server} \subset X_{priv}$

X_{server} and X_{priv} decreases progressively. Fig. 13 illustrates that higher correlation can lead to better performance of the functionality stealing attack.

•**Performance of different pseudo-client model structures.** In PCAT, the server does not know the structure of the victim client model, and only knows learning task and its input and output formats. It can adopt any structure that works for the task as the pseudo-client model. We evaluate PCAT in cases that the structure of pseudo-client model is simpler or more complex than the victim client model. Table 2 presents the performance with different variants of pseudo-client model on MNIST. We also consider the situation that convolution layers are replaced by ResBlock [18] on CIFAR-10 [26] and present the results in Table 3. Both tables show that when the pseudo-client model has the same structure as the victim model, it achieves the best performance. A more complex pseudo-client model can achieve almost the same performance as the same structure does, while two simpler structures suffer performance degradation to different degrees. We think the reason is that the simpler the model, the less qualified to learn the behaviors of the victim client model. Therefore, a more complex pseudo model has a higher chance to successfully steal the functionality of a SL model.

	Pseudo client			Victim client
	Simple	Same	Complex	
Model	MaxPool ReLU Conv2d	MaxPool ReLU Conv2d MaxPool ReLU Conv2d	MaxPool ReLU Conv2d ReLU Conv2d MaxPool ReLU Conv2d	MaxPool ReLU Conv2d MaxPool ReLU Conv2d
Acc(%)	73.60	97.17	97.13	99.06
MSE	0.387	0.133	0.141	0

Table 2: Performance of PCAT on MNIST, when the pseudo-client model has simpler, the same, or more complex structures than the client. The server has 5 samples per class.

	Pseudo client				Victim client
	Simple	Same	Complex	Other	
Model	MaxPool Conv2d MaxPool Conv2d	MaxPool Conv2d Conv2d MaxPool Conv2d Conv2d	MaxPool Conv2d Conv2d Conv2d MaxPool Conv2d Conv2d Conv2d	ResBlock ResBlock	MaxPool Conv2d Conv2d MaxPool Conv2d Conv2d
Acc(%)	87.54	88.90	88.35	84.96	93.20
MSE	0.0279	0.0134	0.0166	0.0511	0

Table 3: Performance of PCAT on CIFAR-10, when the structure of the pseudo-client model is simpler, the same, more complex than the victim client model, or even use other elementary structure. The server has 250 samples per class.

•**Performance with different splitting depths.** As presented in Section 5.3 and Fig.9, we implemented PCAT with different model splitting strategies and evaluate how the complexity/depth of the victim client model affects the attack. Fig.14 shows the results of MobileNet on Tiny ImageNet. When the client model is split from Layer1 to Layer4, the accuracy of the pseudo model reaches 79.01%, 78.23%, 75.33%, and 68.63%, respectively. Though the increasing complexity of the client model affects the accuracy of functionality stealing, the attack is still effective.

•**Comparison with previous work.** We compare PCAT with the most related work UnSplit [9], which is the SOTA

MNIST					
Samples / Class	1	2	5	10	25
Acc(%)	72.30	84.27	91.83	93.27	96.98

CIFAR-10					
Samples / Class	10	25	50	100	250
Acc(%)	36.31	65.42	76.16	82.56	93.28

Table 4: Functionality stealing results of U-Shape PCAT. It is on MNIST/CIFAR-10 with bottom layer2, top layer2/layer1 and $X_{server} \subset X_{priv}$.

Datasets	**MNIST**		**CIFAR-10**	
Methods	UnSplit [9]	PCAT	UnSplit [9]	PCAT
SL Baseline	98.00	99.00	71.00	93.20
split layer = 1	93.75	**98.75**	43.69	**91.10**
split layer = 2	63.3	**96.79**	22.12	**78.57**

Table 5: Comparison on functionality stealing performance (Acc: %) with UnSplit [9]. In PCAT, the attacker has 5 samples per class from X_{priv} in MNIST and 50 samples per class from X_{priv} in CIFAR-10

method for a semi-honest server to steal the functionality of the client model and reconstruct the raw inputs and labels. The main difference is that UnSplit requires the server to know the structure of the victim client model while PCAT treats the victim client as a black box. Table 5 compares their ability to steal functionality. PCAT significantly outperforms UnSplit in all cases with different models, datasets and splitting strategies. For example, on MNIST, when the client has two layers, the accuracy of UnSplit is only 63.3%, while PCAT achieves 96.79%. On CIFAR-10, where the model is more complex, the accuracy of UnSplit is only 22.12%, while PCAT achieves 78.57%.

(2) PCAT for U-Shape SL As illustrated in Fig.9, we also implement a U-Shape LeNet-5/VGG16 split learning on MNIST/CIFAR-10, which has two bottom layers and two/one top layers on the client side. We use PCAT to successfully steal the functionality of the U-Shape split learning model. Results in Table 4 show that PCAT achieves 96.98%/93.28% accuracy when there are 25/250 samples per class in the server's dataset. Hence, PCAT can attack a wide range of split learning, including two-part SL and U-shape SL.

6.2 Input Data Reconstruction

After stealing the functionality of the victim client model, following the method in Section 4.2, the server can reconstruct the private inputs of the client. We present our reconstruction results for two-part split learning on three datasets and compare them with UnSplit in Table 6. More reconstruction results are presented in Appendix Fig. 17, Fig.18, Fig.19. Note that, UnSplit didn't implement data reconstruction on Tiny-ImageNet. Obviously, the images reconstructed by our method are much more informative and clearer than those

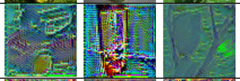
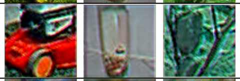
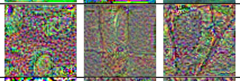
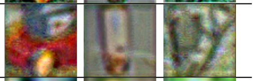
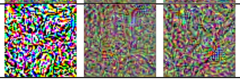
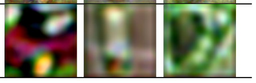
	UnSplit	PCAT
truth		
layer1		
layer2		
layer3		

Table 6: Comparison of data reconstruction on Tiny-ImageNet between UnSplit and PCAT.

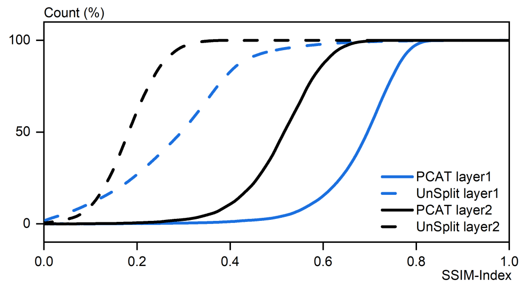


Figure 15: The cdf curves of ssim [48] index between reconstruction results and ground truth on CIFAR-10 testsets, which shows that PCAT can recover private inputs with much more structural similarity.

reconstructed by UnSplit.

We also make quantitative analysis of reconstruction result of PCAT and UnSplit by calculating the SSIM [48] index between reconstructed images and the groundtruth. The comparison is shown in Fig. 15. Obviously under the same condition, PCAT can reconstruct raw inputs with much higher similarity. Especially, when the client model and the learning task are complex, e.g., the client model has two layers from VGG16 on CIFAR-10, UnSplit almost fails to reconstruct the input images, while our method can still reconstruct the input images with fairly good clearness.

We also evaluate our input reconstruction method for U-shape split learning and non-i.i.d. setting. The results are shown in Fig.16. For U-shape learning, our method can also clearly reconstruct the private inputs. More importantly, in the non-i.i.d. setting, though the server lacks samples of some classes, it can still reconstruct the inputs of the unknown classes. This is a serious privacy breach since the server steals the data it has never seen before from the client.

Input reconstruction requires a reverse mapping to map smashed data back to the input space. If the client model is more complex and harder to invert, it may impede the data reconstruction attack. Thus, we analyze how the client structure can affect the attack performance from two aspects: 1) more complex client; 2) adding noise to smashed data to reduce the correlations between smashed data and raw inputs. In [22], model structures (e.g. layer depth) and the amount of param-

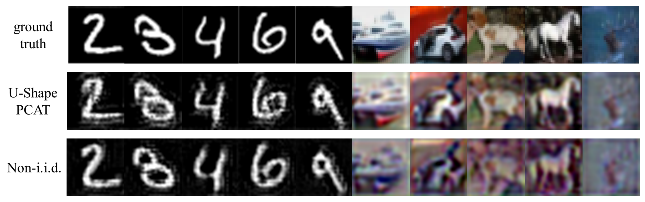
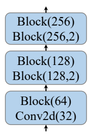
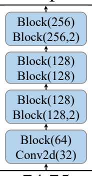
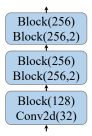
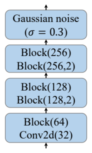


Figure 16: Data reconstruction results on MNIST and CIFAR-10 in non-i.i.d. and U-shape cases. For U-Shape split learning, the server has 5/20 (MNIST/CIFAR-10) samples per class from X_{priv}, with bottom layer = 1 and top layer = 2. For the non-i.i.d. case, split layer = 1; the server lacks samples from class "3"/ "deer" (MNIST/CIFAR-10) and has 5/20 samples from X_{priv} for each other class, but it can still reconstruct the missing classes accurately.

Client	Base	Deeper	More param	Noise

Structure	 Block(256)
	Block(256,2)
	Block(128)
	Block(128,2)
	Block(64)
	Conv2d(32)
		 Block(256)
		Block(256,2)
		Block(128)
		Block(128)
		Block(128)
		Block(128,2)
		Block(64)
		Conv2d(32)
			 Block(256)
			Block(256,2)
			Block(256)
			Block(256,2)
			Block(128)
			Conv2d(32)
				 Gaussian noise
				(σ = 0.3)
				Block(256)
				Block(256,2)
				Block(128)
				Block(128,2)
				Block(64)
				Conv2d(32)

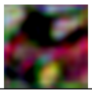
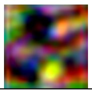
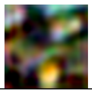
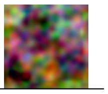
Acc(%)	74.52	74.75	73.70	79.00
MSE	0.0362	0.0339	0.0398	0.2108
				

Table 7: Data reconstruction results when the client structure is harder to invert. It is performed on Tiny-Imagenet. In all the cases, X_{server} has 50 samples per class.

eters are two factors to influence model complexity. Thus, we test data reconstruction in three cases: 1) the client model has more layers; 2) the client model has more parameters; 3) the client adds noise to the smashed data before sending it to the server. The results are presented in Table 7. Through the results can we conclude that model complexity will affect data reconstruction to a certain extent, but it only triggers very limited impacts on functionality stealing attacks. Surprisingly, we find that adding Gaussian noise to the smashed data can improve pseudo-client's accuracy. The reason behind this phenomenon is that noise will slow down the baseline's convergence, and the pseudo-client can better catch up with the baseline. More experiments about this phenomenon is presented in Table 12.

6.3 Label Inference

Based on the pseudo-client model, we conduct label inference attack on U-shape split learning on MNIST/CIFAR-10 Datasets. As shown in Table 8, PCAT achieves high accuracy in label inference, which is 98.23%/93.23% when the server has 25/250 samples per class. Also, we compare PCAT with UnSplit [9] in label inference in Table 9. When the top model has one layer, the inference accuracy of PCAT is 98.82%/93.42%, which is slightly lower than Unsplit. But,

MNIST					
Samples / Cls	1	2	5	10	25
Acc(%)	82.65	94.42	96.58	96.89	98.23
CIFAR-10					
Samples / Cls	10	25	50	100	250
Acc(%)	19.29	89.10	92.83	93.08	93.23

Table 8: Label inference accuracy of PCAT on U-shape split learning. It is implemented on MNIST/CIFAR-10 with bottom layer2 and top layer2/layer1. $X_{server} \subset X_{priv}$.

Datasets	**MNIST**		**CIFAR-10**	
Methods	UnSplit	PCAT	UnSplit	PCAT
top layer = 1	100.0	**98.82**	100.0	**93.42**
top layer = 2	9.1	**96.58**	8.1	**92.57**

Table 9: Comparison on label inference accuracy(%) with UnSplit. In PCAT, the attacker has 5/100 samples per class from X_{priv} in MNIST/CIFAR-10.

when the top model becomes more complex, e.g. having two layers, the inference accuracy of UnSplit severely drops to 9.1%/8.1%, while PCAT still achieves a high accuracy (96.58%/92.57%). Therefore, PCAT is more robust to various split learning models.

6.4 Against defensive mechanisms

So far, there have been some defenses proposed for SL privacy attacks: some of them are formal privacy protection mechanisms (e.g. differential privacy [1]) and others are designed for SL (e.g. Nopeek [45]). Our experiments shows that PCAT can still work well against those defenses.

•**Nopeek:** Distance correlation minimization [40] is a widely adopted defensive mechanism in split learning. It is designed to let clients calculate the distance correlation loss of each sample and its smashed data to reduce unnecessary information leakage for attackers to reconstruct raw inputs. The loss function is as the following:

$$\mathcal{L} = \alpha \cdot DCOR(x, f_c(x)) + (1-\alpha) \cdot TASK(y, f_s(f_c(x))) \quad (2)$$

Nopeek [45] is a modification of distance correlation minimization to prevent leakage at the beginning of split learning. Though it reduces the risk of data leakage, the server still knows the output feature space of the client model. Thus, our functionality stealing is still effective. We perform attacks against distance correlation minimization with Nopeek. Table 10 shows that the defense technique like Nopeek has little influence on the effectiveness of PCAT, and the pseudo-client model still achieves a very small accuracy gap with the real client model, which varies from 0.99% to 3.57%.

•**Differential privacy:** Differential privacy can provide a rigorous mathematical privacy guarantee [7], which can be applied to deep learning in [1]. We adopt this defense mechanism for SL. To be more specific, when the victim client

MNIST					
α	0	0.2	0.4	0.6	0.8
Baseline Acc(%)	99.00	98.52	98.10	96.98	94.33
PCAT Acc(%)	98.01	97.27	96.89	93.41	92.55
Acc(%) Gap	0.99	1.25	1.21	3.57	1.78
CIFAR-10					
α	0	0.1	0.2	0.4	0.6
Baseline Acc(%)	93.2	87.56	78.64	68.04	62.61
PCAT Acc(%)	82.77	75.29	64.42	60.05	55.13
Acc(%) Gap	10.43	12.27	14.22	7.99	7.47

Table 10: The functionality stealing performance of PCAT against the SL with defense mechanism Nopeek [45]. It is performed on MNIST/CIFAR-10, with split layer2/layer1. The server obtains 10/20 samples per class and $X_{server} \subset X_{pub}$ α is the parameter in the loss function of Nopeek.

MNIST				
σ	+∞	70	60	50
Baseline Acc(%)	99.00	94.10	90.79	84.71
PCAT Acc(%)	97.31	91.12	88.66	80.84
Acc(%) Gap	1.69	2.98	2.13	3.87
CIFAR-10				
σ	+∞	200	100	50
Baseline Acc(%)	93.20	85.18	80.17	73.17
PCAT Acc(%)	86.50	77.45	71.14	68.34
Acc(%) Gap	6.70	7.73	9.03	4.83

Table 11: The functionality performance of PCAT against the SL with differential privacy [1] defense. It is performed on MNIST/CIFAR-10, with split layer2/layer1. The server obtains 5/100 samples per class and $X_{server} \subset X_{priv}$

receives gradients from the server, it will add Laplacian noise under DP's guarantee. Thus, the client can protect its model and the next smashed data sent to the server. The results are shown in Table. 11: though the accuracy of baseline decreases as the noise becomes larger, the accuracy gap between the baseline and pseudo-client is steady at a certain level.

•**Add noise to smashed data**Another intuitive defense is adding Guassian noise to smashed data directly. The results are presented in Table. 12. We are surprised to discover that adding noise to smashed data can not impede PCAT's attack but improve pseudo-client to achieve better performance. We think the reason behind this phenomenon is that noise will impede the baseline's convergence which helps the pseudo-client catches up with the baseline.

6.5 Computation Cost

We compare computation cost of PCAT and UnSplit for three attack goals: functionality stealing, data reconstruction, and label inference. UnSplit uses disjoint gradient descent which means in every iteration it performs forward and backward propagation twice: one for optimizing model and the other for inputs. But UnSplit only needs to compute the client model, while PCAT needs to compute the complete model. For data reconstruction, PCAT only executes once at the last epoch but UnSplit executes once for every epoch. For label inference, UnSplit performs backward propagation for every class while PCAT only needs to infer labels once. We summarize the computation cost of PCAT and UnSplit in Table 13. We also present the practical terms for each factor in Table 13. Take PCAT on MNIST with LeNet-5 as an example: If the server has 50 training samples, then $K = 54000, k = 50, N = 2000, n = 20$ and $F_c = \widetilde{F_c} = 376.51KFLOPs$, $F_s = 58.92KFLOPs$, $F_t = 11.26KFLOPs$. Therefore, PCAT consumes much less computation than UnSplit.

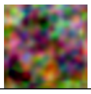
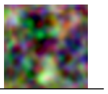
σ	0	0.1	0.3	0.5
Baseline Acc(%)	80.28	79.80	79.90	80.07
PCAT Acc(%)	74.52	77.79	79.00	79.45
MSE	0.0362	0.0864	0.2108	0.3690
				

Table 12: Performance of functionality stealing and input reconstruction when noises are added to smashed data. In all cases we add Gaussian noises. We test them on Tiny-Imagenet and the server has 50 samples per class.

Attacks	UnSplit [9]	PCAT
Functionality stealing	nKF_c	$Nk(\widetilde{F_c} + F_s)$
Data reconstruction	nKF_c	$K\widetilde{F_c}$
Labels inference	$N_{cls}KF_t$	KF_t

Table 13: Computation cost of PCAT and UnSplit. n and N are the total epochs for UnSplit and PCAT, respectively. K and k denote the number of samples in X_{priv} and X_{server}. $\widetilde{F_c}$, F_s, F_c, F_t corresponds to FLOPs of pseudo-client, server, victim client, and top model. N_{cls} means the number of classes.

7 Conclusions

In this work, we propose a novel pseudo-client attack (PCAT) mechanism on various SL models. PCAT enables an semi-honest server to conduct functionality stealing, input data reconstruction and label inference without knowing the structure of the victim client model. We implemented PCAT for rich models, tasks and settings. Comprehensive experiments demonstrate that PCAT works effectively for rich models, tasks and settings. The whole attack process is transparent to the client, which thus reveals a serious privacy risk of SL.

There remains some open problems for the future work. For example, in this work the data reconstruction attack relies on the assumption that the client model is invertible. We will further explore which determines the invertibility of a model and how to reconstruct the input if the model is not invertible.

Acknowledgements

Lan Zhang is the corresponding author. The authors wish to thank the anonymous reviewers for their insightful and constructive comments on this paper. This research was supported by the National Key RD Program of China 2021YFB2900103, China National Natural Science Foundation with No. 61932016. This work was partially supported by "the Fundamental Research Funds for the Central Universities"WK2150110024.

References

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 308–318, New York, NY, USA, 2016. Association for Computing Machinery.

[2] Ali Abedi and Shehroz S. Khan. Fedsl: Federated split learning on distributed sequential data in recurrent neural networks. *CoRR*, abs/2011.03180, 2020.

[3] Sharif Abuadbba, Kyuyeon Kim, Minki Kim, Chandra Thapa, Seyit Ahmet Çamtepe, Yansong Gao, Hyoungshick Kim, and Surya Nepal. Can we use split learning on 1d CNN models for privacy preserving training? In Hung-Min Sun, Shiuh-Pyng Shieh, Guofei Gu, and Giuseppe Ateniese, editors, *ASIA CCS '20: The 15th ACM Asia Conference on Computer and Communications Security, Taipei, Taiwan, October 5-9, 2020*, pages 305–318. ACM, 2020.

[4] Ahmad Ayad, Melvin Renner, and Anke Schmeink. Improving the communication and computation efficiency of split learning for iot applications. In *IEEE Global Communications Conference, GLOBECOM 2021, Madrid, Spain, December 7-11, 2021*, pages 1–6. IEEE, 2021.

[5] Emiliano De Cristofaro. An overview of privacy in machine learning. *CoRR*, abs/2005.08679, 2020.

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 248–255. IEEE Computer Society, 2009.

[7] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014.

[8] Ege Erdogan, Alptekin Küpçü, and A. Ercüment Çiçek. Splitguard: Detecting and mitigating training-hijacking attacks in split learning. *IACR Cryptol. ePrint Arch.*, page 1080, 2021.

[9] Ege Erdogan, Alptekin Küpçü, and A. Ercüment Çiçek. Unsplit: Data-oblivious model inversion, model stealing, and label inference attacks against split learning. *IACR Cryptol. ePrint Arch.*, page 1074, 2021.

[10] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 1322–1333. ACM, 2015.

[11] Matthew Fredrikson, Eric Lantz, Somesh Jha, Simon M. Lin, David Page, and Thomas Ristenpart. Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing. In Kevin Fu and Jaeyeon Jung, editors, *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*, pages 17–32. USENIX Association, 2014.

[12] Yansong Gao, Minki Kim, Sharif Abuadbba, Yeonjae Kim, Chandra Thapa, Kyuyeon Kim, Seyit Ahmet Çamtepe, Hyoungshick Kim, and Surya Nepal. End-to-end evaluation of federated learning and split learning for internet of things. In *International Symposium on Reliable Distributed Systems, SRDS 2020, Shanghai, China, September 21-24, 2020*, pages 91–100. IEEE, 2020.

[13] Grzegorz Gawron and Philip Stubbings. Feature space hijacking attacks against differentially private split learning. *CoRR*, abs/2201.04018, 2022.

[14] Neil Zhenqiang Gong and Bin Liu. You are who you know and how you behave: Attribute inference attacks via users' social friends and behaviors. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 979–995, 2016.

[15] Neil Zhenqiang Gong and Bin Liu. Attribute inference attacks in online social networks. *ACM Transactions on Privacy and Security (TOPS)*, 21(1):1–30, 2018.

[16] Otkrist Gupta and Ramesh Raskar. Distributed learning of deep neural network over multiple agents. *J. Netw. Comput. Appl.*, 116:1–8, 2018.

[17] Chaoyang He, Keshav Balasubramanian, Emir Ceyani, Yu Rong, Peilin Zhao, Junzhou Huang, Murali Annavaram, and Salman Avestimehr. Fedgraphnn: A federated learning system and benchmark for graph neural networks. *CoRR*, abs/2104.07145, 2021.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016.

[19] Zecheng He, Tianwei Zhang, and Ruby B. Lee. Model inversion attacks against collaborative inference. In David Balenson, editor, *Proceedings of the 35th Annual Computer Security Applications Conference, ACSAC 2019, San Juan, PR, USA, December 09-13, 2019*, pages 148–162. ACM, 2019.

[20] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.

[21] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[22] Xia Hu, Weiqing Liu, Jiang Bian, and Jian Pei. Measuring model complexity of neural networks with curve activation functions. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1521–1531, 2020.

[23] Jinyuan Jia and Neil Zhenqiang Gong. {AttriGuard}: A practical defense against attribute inference attacks via adversarial machine learning. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 513–529, 2018.

[24] Guoxi Xu Jiayu Wu, Qixiang Zhang. Tiny imagenet challenge. http://cs231n.stanford.edu/reports/2017/pdfs/930.pdf, 2021.

[25] Jongwon Kim, Sungho Shin, Yeonguk Yu, Junseok Lee, and Kyoobin Lee. Multiple classification with split learning. In *SMA 2020: The 9th International Conference on Smart Media and Applications, Jeju, Republic of Korea, September 17 - 19, 2020*, pages 358–363. ACM, 2020.

[26] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[27] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.

[28] Oscar Li, Jiankai Sun, Xin Yang, Weihao Gao, Hongyi Zhang, Junyuan Xie, Virginia Smith, and Chong Wang. Label leakage and protection in two-party split learning. *CoRR*, abs/2102.08504, 2021.

[29] Wei Yang Bryan Lim, Jer Shyuan Ng, Zehui Xiong, Dusit Niyato, Cyril Leung, Chunyan Miao, and Qiang Yang. Incentive mechanism design for resource sharing in collaborative edge learning. *CoRR*, abs/2006.00511, 2020.

[30] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Knockoff nets: Stealing functionality of black-box models. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 4954–4963. Computer Vision Foundation / IEEE, 2019.

[31] Safa Otoum, Nadra Guizani, and Hussein Mouftah. On the feasibility of split learning, transfer learning and federated learning for preserving security in its systems. *IEEE Transactions on Intelligent Transportation Systems*, 2022.

[32] Sangjoon Park, Gwanghyun Kim, Jeongsol Kim, Boah Kim, and Jong Chul Ye. Federated split task-agnostic vision transformer for covid-19 cxr diagnosis. *Advances in Neural Information Processing Systems*, 34, 2021.

[33] Dario Pasquini, Giuseppe Ateniese, and Massimo Bernaschi. Unleashing the tiger: Inference attacks on split learning. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 2113–2129. ACM, 2021.

[34] Maarten G. Poirot, Praneeth Vepakomma, Ken Chang, Jayashree Kalpathy-Cramer, Rajiv Gupta, and Ramesh Raskar. Split learning for collaborative deep learning in healthcare. *CoRR*, abs/1912.12115, 2019.

[35] Maria Rigaki and Sebastian Garcia. A survey of privacy attacks in machine learning. *CoRR*, abs/2007.07646, 2020.

[36] Jihyeon Ryu, Dongho Won, and Youngsook Lee. A study of split learning model. In *16th International Conference on Ubiquitous Information Management and Communication, IMCOM 2022, Seoul, Korea, Republic of, January 3-5, 2022*, pages 1–4. IEEE, 2022.

[37] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 1310–1321. ACM, 2015.

[38] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR*

2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015.

[39] Abhishek Singh, Praneeth Vepakomma, Otkrist Gupta, and Ramesh Raskar. Detailed comparison of communication efficiency of split learning and federated learning. *CoRR*, abs/1909.09145, 2019.

[40] Gábor J. Székely, Maria L. Rizzo, and Nail K. Bakirov. Measuring and testing dependence by correlation of distances. *The Annals of Statistics*, 35(6):2769 – 2794, 2007.

[41] Chandra Thapa, Mahawaga Arachchige Pathum Chamikara, and Seyit Camtepe. Splitfed: When federated learning meets split learning. *CoRR*, abs/2004.12088, 2020.

[42] Chandra Thapa, Mahawaga Arachchige Pathum Chamikara, and Seyit Ahmet Çamtepe. Advancements of federated learning towards privacy preservation: from federated learning to split learning. *CoRR*, abs/2011.14818, 2020.

[43] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 601–618. USENIX Association, 2016.

[44] Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. Split learning for health: Distributed deep learning without sharing raw patient data. *CoRR*, abs/1812.00564, 2018.

[45] Praneeth Vepakomma, Abhishek Singh, Otkrist Gupta, and Ramesh Raskar. Nopeek: Information leakage reduction to share activations in distributed deep learning. In Giuseppe Di Fatta, Victor S. Sheng, Alfredo Cuzzocrea, Carlo Zaniolo, and Xindong Wu, editors, *20th International Conference on Data Mining Workshops, ICDM Workshops 2020, Sorrento, Italy, November 17-20, 2020*, pages 933–942. IEEE, 2020.

[46] Binghui Wang and Neil Zhenqiang Gong. Stealing hyperparameters in machine learning. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 36–52. IEEE Computer Society, 2018.

[47] Song Wang, Xinyu Zhang, Hiromasa Uchiyama, and Hiroki Matsuda. Hivemind: Towards cellular native machine learning model splitting. *IEEE J. Sel. Areas Commun.*, 40(2):626–640, 2022.

[48] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[49] Christopher J.C. Burges Yann LeCun, Corinna Cortes. Mnist handwritten digital database. http://yann.lecun.com/exdb/mnist/, 2021.

[50] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 14747–14756, 2019.

Appendix

Reverse mapping structure. In Table 14, we illustrate the clients' and pseudo-clients' model structures we used when we perform input reconstruction. The notation $\widetilde{f_c}$ indicates the pseudo-client's structures while f^{-1} indicates the reverse mapping's structures.

Layer	Model	MNIST	CIFAR-10	Tiny ImageNet

1	$\widetilde{f_c}$	MaxPool
		ReLU
		Conv2d
			MaxPool
			Conv2d
			Conv2d
				Block(64)
				Conv2d(32)

1	f^{-1}	Convt2d Tanh Upsample	Convt2d Tanh Upsample	Convt2d Tanh

2	$\widetilde{f_c}$	MaxPool
		ReLU
		Conv2d
		MaxPool
		ReLU
		Conv2d
			MaxPool
			Conv2d
			Conv2d
			MaxPool
			Conv2d
			Conv2d
				Block(128)
				Block(128,2)
				Block(64)
				Conv2d(32)

2	f^{-1}	Upsample Convt2d Tanh Upsample	Upsample Convt2d Tanh Upsample	Convt2d Tanh Upsample

3	$\widetilde{f_c}$	/	/	Block(256)
				Block(256,2)
				Block(128)
				Block(128,2)
				Block(64)
				Conv2d(32)

3	f^{-1}	/	/	Upsample Convt2d Tanh Upsample

Table 14: Illustrations of reverse mapping for every case in data reconstruction. Convt2d denotes transposed convolution layer. Upsample denotes bilinear upsampling layer.

Gradient of distance correlation. In the privacy defense Nopeek [45], the distance correlation loss between smashed data and raw inputs is calculated. In this section we present the formulation of distance correlation loss as well as its gradients calculation. As mentioned in [45], Distance correlation between centered data:

$$\frac{(\mathbf{X^TXZ^TZ})}{\sqrt{(\mathbf{X^TX})^2(\mathbf{Z^TZ})^2}} \tag{3}$$

Distance covariance in the numerator:

$$(\mathbf{X^TZX}) = \sum_{ij} \langle z_i, z_j \rangle (\|x_i - x_j\|)^2 \tag{4}$$

This can be written in matrix form using basis vectors e_i, e_j as

$$\sum_{ij} [(\mathbf{Z^T e_i e_j^T Z}) \mathbf{Tr}(\mathbf{X^T}(\mathbf{e_i} - \mathbf{e_j})(\mathbf{e_i} - \mathbf{e_j})^\mathbf{T}\mathbf{X})] \tag{5}$$

Simplifying the notation with $M_{ij} = e_i e_j^T$ and $A_{ij} = (e_i - e_j)(e_i - e_j)^T$ we have

$$\frac{\partial Tr(\mathbf{Z^T L_Z Z})}{\partial \mathbf{Z}} = \sum_{ij} (\mathbf{2M_{ij}Z}) \mathbf{Tr}(\mathbf{X^T A_{ij} X}) \tag{6}$$

On the lines of 5, we have

$$Tr(\mathbf{Z^T L_Z Z}) = \sum_{ij} [Tr(\mathbf{Z^T M_{ij} Z}) \mathbf{Tr}(\mathbf{Z^T A_{ij} Z})] \tag{7}$$

Therefore utilizing these identities, the derivative of squared distance correlation w.r.t **Z** can be written as

$$\frac{c_x Tr(\mathbf{Z^T L_Z Z}) \frac{\partial Tr(\mathbf{X^T L_Z X})}{\partial \mathbf{Z}} - [Tr(\mathbf{X^T L_Z X})]^2 c_x \frac{\partial Tr(\mathbf{Z^T L_Z Z})}{\partial \mathbf{Z}}}{[Tr(\mathbf{Z^T L_Z Z})]^2} \tag{8}$$

Extra data reconstruction results. To show effectiveness of PCAT to reconstruct input data, we perform extra experiments and the reconstruction results are presented in Fig. 17-19, as well as the comparison with UnSplit [9] in the same cases.

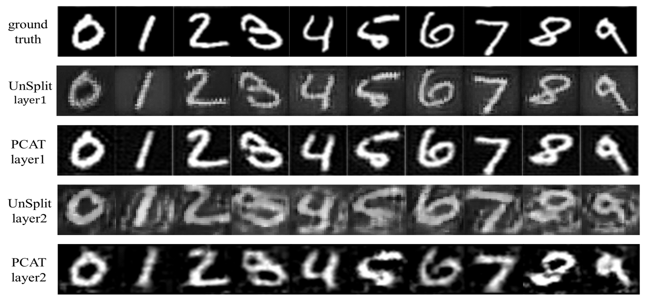


Figure 17: Input data reconstruction results of PCAT and UnSplit [9] on MNIST. In PCAT, the server attacks using 5 samples per class from X_{priv} for layer1 and 10 samples per class for layer2.

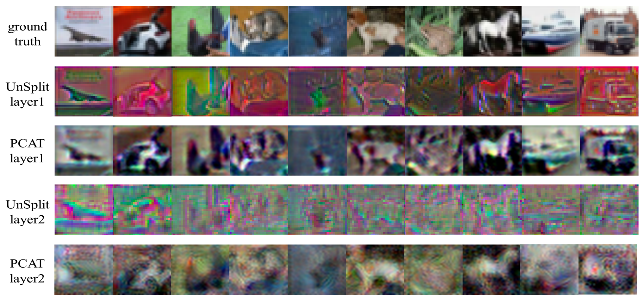


Figure 18: Input data reconstruction results of PCAT and UnSplit [9] on CIFAR-10. In PCAT, the server attacks using 50 samples per class from X_{priv} for layer1 and 250 samples per class for layer2.

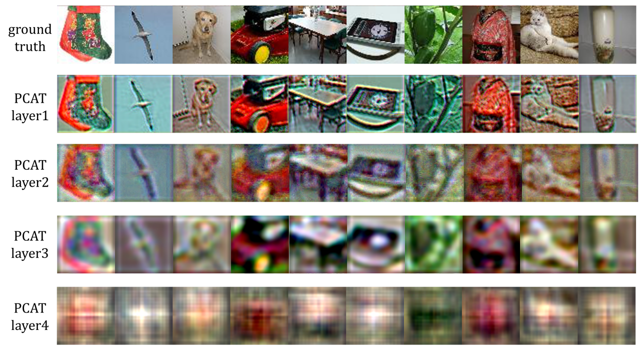


Figure 19: Input data reconstruction results of PCAT on Tiny-ImageNet [24]. In PCAT, the server attacks using 5, 10, 25 samples per class from X_{priv} for layer1, 2, 3 individually.