Unleashing the Tiger: Inference Attacks on Split Learning

Dario Pasquini
Sapienza University of Rome
Institute of Applied Computing, CNR
pasquini@di.uniroma1.it

Giuseppe Ateniese
Stevens Institute of Technology
gatenies@stevens.edu

Massimo Bernaschi
Institute of Applied Computing, CNR
massimo.bernaschi@cnr.it

ABSTRACT

We investigate the security of *split learning*—a novel collaborative machine learning framework that enables peak performance by requiring minimal resources consumption. In the present paper, we expose vulnerabilities of the protocol and demonstrate its inherent insecurity by introducing general attack strategies targeting the reconstruction of clients' private training sets. More prominently, we show that a malicious server can actively hijack the learning process of the distributed model and bring it into an insecure state that enables inference attacks on clients' data. We implement different adaptations of the attack and test them on various datasets as well as within realistic threat scenarios. We demonstrate that our attack is able to overcome recently proposed defensive techniques aimed at enhancing the security of the split learning protocol. Finally, we also illustrate the protocol's insecurity against malicious clients by extending previously devised attacks for Federated Learning.

To make our results reproducible, we made our code available.[1]

1 INTRODUCTION

Once the cattle have been split up, then the tiger strikes.

A Myanma proverb

Deep learning requires massive data sets and computational power. State-of-the-art neural networks may contain millions or billions [13] of free parameters and necessitate representative training sets. Unfortunately, collecting suitable data sets is difficult or sometimes impossible. Entities and organizations may not be willing to share their internal data for fear of releasing sensitive information. For instance, telecommunication companies would benefit extraordinarily from deep learning techniques but do not wish to release customer data to their competitors. Similarly, medical institutions cannot share information because privacy laws and regulations shelter patient data.

Secure data sharing and learning can only be achieved via cryptographic techniques, such as homomorphic encryption or secure multi-party computation. However, the combination of cryptography and deep learning algorithms yields expensive protocols. An alternative approach, with mixed results, is distributed/decentralized machine learning, where different parties cooperate to learn a shared model. In this paradigm, training sets are never shared directly. In federated learning [11, 30, 31], for example, users train a shared neural network on their respective local training sets and provide only model parameters to others. The expectation is that by sharing certain model parameters, possibly "*scrambled*" [3], the actual training instances remain hidden and inscrutable. Unfortunately, in [25], it was shown that an adversary could infer meaningful information on training instances by observing how shared model parameters evolve over time.

Split learning is another emerging solution that is gaining substantial interest in academia and industry. In the last few years, a growing body of empirical studies [5, 19, 28, 29, 34, 37, 42, 48, 49], model extensions [4, 14, 26, 36, 40, 44, 46, 47], and other resources [12, 45] attested to the effectiveness, efficiency and relevance of the split learning framework. At the same time, split-learning-based solutions have been implemented and adopted in commercial as well as free open-source applications [1, 2, 6].

The success of split learning is primarily due to its practical properties. Indeed, compared with other approaches such as federated learning [11, 30, 31], split learning requires consistently fewer resources from the participating clients, enabling lightweight and scalable distributed training solutions. However, while the practical properties of split learning have been exhaustively validated [42, 49], little effort has been spent investigating the security of this machine learning framework. In this paper, **we carry out the first, in-depth, security analysis of split learning and draw attention to its inherent insecurity.** We demonstrate that the assumptions on which the security of split learning is based are fundamentally flawed, and a motivated adversary can easily subvert the defenses of the training framework. In particular, we implement a general attack strategy that allows a malicious server to recover private training instances during the distributed training. In the attack, the server hijacks the model's learning processes and drive them to an insecure state that can be exploited for inference attacks. In the process, the attacker does not need to know the client's private training sets or the client's architecture. The attack is domain-independent and can be seamlessly applied to various split learning variants [44, 46]. We call this general attack: the **feature-space hijacking attack** (FSHA) and introduce several adaptations of it. We test the proposed attack on different datasets and demonstrate their applicability under realistic threat scenarios such as data-bounded adversaries.

Furthermore, we show that client-side attacks that have been previously devised on federated learning settings remain effective within the split learning framework. In particular, we adapt and extend the inference attack proposed in [25] to make it work in Split Learning. Our attack demonstrates how a malicious client can recover suitable approximations of private training instances of other honest clients participating in the distributed training. Eventually, this result confirms the insecurity of Split Learning also against client-side attacks.

The contributions of the present paper can be then summarized as follows:

- We demonstrate the insecurity of Split Learning against a **malicious server** by devising a novel and general attack framework. Such a framework permits an attacker to

[1] https://github.com/pasquini-dario/SplitNN_FSHA

(1) recover precise reconstructions of individual clients' training instances as well as **(2)** perform property inference attacks [8] for arbitrary attributes. Additionally, we show that the proposed attacks can circumvent defensive techniques devised for split learning [47, 50].
- We demonstrate the insecurity of split learning against a **malicious client** by adapting and extending previously proposed techniques targeting federated learning [25]. The attack permits a malicious client to recover prototypical examples of honest clients' private instances.

Overview. The paper starts by surveying distributed machine learning frameworks in Section 2. Section 3 follows by introducing and validating our main contribution—the feature-space hijacking attack framework. Then, Section 4 covers the applicability of existing defensive mechanisms within the split learning framework. In Section 5, we analyze the security of split learning against malicious clients. Section 6 concludes the paper, although Appendices contain additional material. In the paper, background and analysis of previous works are provided, when necessary, within the respective sections.

2 DISTRIBUTED MACHINE LEARNING

Distributed (also collaborative [41]) machine learning allows a set of remote clients $\mathbf{Cs} = \{c_1, \ldots, c_n\}$ to train a shared model F. Each client c_i participates in the training protocol by providing a set of training instances X_{priv_i}. This set is private and must not be directly shared among the parties running the protocol. For instance, hospitals cannot share patients' data with external entities due to regulations such as HIPAA [7].

In this section, we focus on distributed machine learning solutions for deep learning models. In particular, we describe: **(1)** *Federated learning* [11, 30, 31] which is a well-established learning protocol and **(2)** *split learning* [22, 37, 48] a recently proposed approach that is gaining momentum due to its attractive practical properties.

2.1 Federated Learning

Federated learning [11, 30, 31] allows for distributed training of a deep neural model by aggregating and synchronizing local parameter adjustments among groups of remote clients. In the most straightforward setup, the protocol is orchestrated by a central server that manages clients' training rounds and maintains a master copy of the trained model.

In the initial setup phase, the parties choose a training task and define a machine learning model. The latter is initialized and hosted by the server that makes it available to all remote clients. At each training step, each client downloads the model from the server and locally applies one or more iterations of the standard SGD using its private training set. After the local training is done, clients send the accumulated gradients to the server.[2] The server aggregates these changes into a single training signal applied to the hosted model parameters, completing a global training iteration. Once the server's network is updated, the clients download the new state of the model and repeat the protocol till a stop condition is reached.

[2]This process may differ in practice as there are several implementations of federated learning.

At each iteration in federated learning, clients exchange an amount of data with the server that is linear in the number of parameters of the network. For large models, this becomes unsustainable and may limit the applicability of the approach. Several improvements to the framework have been proposed to address this problem [39, 51].

2.1.1 On the security of Federated Learning. Clients share only gradients/weights induced by the local training steps. The intuition behind federated learning is that local data is safe because it is never directly shared with the server or other clients. Additionally, gradients collected by the server can be further protected through a secure aggregation protocol. The aim is to hinder inference attacks by the server that cannot distinguish clients' individual gradients.

In federated learning, all the parties have equal access to the trained network. The server and the clients know the architecture of the network as well as its weights during the various training steps.

Under suitable assumptions, different attacks on federate learning were shown feasible. The first and most prominent is an active attack [25] that allows a malicious client to infer relevant information on training sets of other honest clients by manipulating the learning process. Other attacks include backdoor injection and poisoning [9, 10, 15]. Accordingly, variants of federated learning have been proposed to reduce the effectiveness of those attacks [16, 17, 23, 27], but they achieved only limited results.

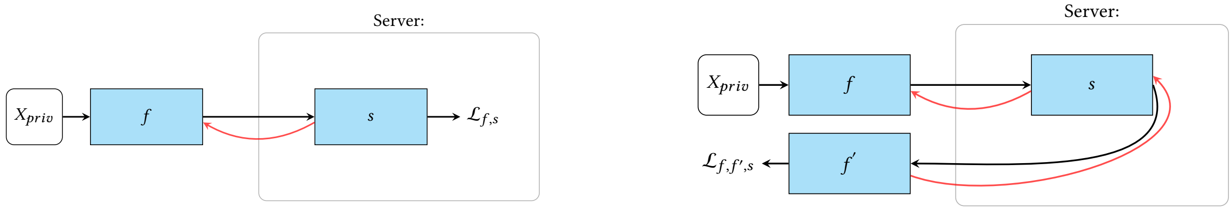
2.2 Split Learning

Split learning [22, 37, 48] enables distributed learning by partitioning a neural network in consecutive chunks of layers among various parties; typically, a set of clients and a server. In the protocol, the clients aim at learning a shared deep neural network by securely combining their private training sets. The server manages this process and guides the network's training, bearing most of the required computational cost.

In split learning, training is performed through a vertically distributed back-propagation [33] that requires to share *only* intermediate network's outputs (referred to as *smashed data*); rather than the raw, private training instances. This mechanism is sketched in Figure 1. In the minimal setup (i.e., Figure 1a), a client owns the first n layers f of the model, whereas the server maintains the remaining neural network s i.e., $F = s(f(\cdot))$. Here, the model's architecture and hyper-parameters are decided by the set of clients before the training phase. In particular, they agree on a suitable partition of the deep learning model and send the necessary information to the server. The server has no decisional power and ignores the initial split f.

At each training iteration, a client sends the output of the initial layers for a batch of private data X_{priv} (i.e., $f(X_{priv})$) to the server. The server propagates this remote activation through the layers s and computes the loss. Then, a gradient-descent-based optimization is locally applied to s. To complete the round, the server sends the gradient up to the input layer of s to the client that continues the back-propagation locally on f.

In the case of supervised loss functions, the protocol requires the client to share the labels with the server. To avoid that, split learning can be reformulated to support loss function computation



(a) Split learning.

(b) Split learning with labels protection.

Figure 1: Two variations of split learning. Black arrows depict the activation propagation of the participating neural networks, whereas red arrows depict the gradient that follows after the forward pass.

on the client-side (Figure 1b). Here, the activation of the last layer of s is sent to the client that computes the loss function[3], sending the gradient back to the server that continues the back-propagation as in the previous protocol.

Split learning supports the training of multiple clients by implementing a sequential turn-based training protocol. Here, clients are placed in a circular list and interact with the server in a round-robin fashion. On each turn, a client performs one or more iterations of the distributed back-propagation (i.e., Figure 1) by locally modifying the weights of f. Then, the client sends the new f to the next client that repeats the procedure. As stated in [22], the training process, for suitable assumptions, is functionally equivalent to the one induced by the standard, centralized training procedure. That is, clients converge to the same network that they would have achieved by training a model on the aggregated training sets.

To overcome the sequential nature of the training process, extensions of split learning have been proposed [26, 44, 46]. More prominently, in [44], split learning is combined with federated learning (i.e., *splitfed learning*) to yield a more scalable training protocol. Here, the server handles the forward signal of the clients' network in parallel and updates the weights of s. The clients receive the gradient signals and update their local models in parallel. Then, they perform federated learning to converge to a global f before the next iteration of split learning. This process requires an additional server that is different from the one hosting s.[4]

Split learning gained particular interest due to its efficiency and simplicity. Namely, it reduces the required bandwidth significantly when compared with other approaches such as federated learning [42, 49]. Certainly, for large neural networks, intermediate activation for a layer is consistently more compact than the network's gradients or weights for the full network. Furthermore, the computational burden for the clients is smaller than the one caused by federated learning. Indeed, clients perform forward/backward propagation on a small portion of the network rather than on the whole. This allows split learning to be successfully applied to the Internet of Things (IoT) and edge-device machine learning settings [19, 29, 34].

2.2.1 On the security of Split learning. Split learning has been proposed as a privacy-preserving implementation of distributed/collaborative learning [5, 22, 37, 47, 48]. In split learning, users' data privacy relies on the fact that raw training instances are never shared; only "*smashed data*" induced from those instances are sent to the server.

The main advantage of split learning in terms of security is that it can hide information about the model's architecture and hyperparameters. Namely, the server performs its task ignoring the architecture of f or its weights. As assumed in previous works [5, 22, 37, 48], this split is designed to protect the intellectual property of the shared model and reduces the risk of inference attacks perpetrated by a malicious server.

We will show that these assumptions are false, and the split learning framework presents several vulnerabilities that allow an attacker to subvert the training protocol and recover clients' training instances.

The most pervasive vulnerability of the framework is the server's entrusted ability to control the learning process of the clients' network. A malicious server can guide f towards functional states that can be easily exploited to recover X_{priv} data from $f(X_{priv})$. The main issue is that a neural network is a differentiable, smooth function that is naturally predisposed to be functionally inverted. There is no much that can be achieved by splitting it other than a form of *security through obscurity*, which is notoriously inadequate since it gives a false sense of security that fundamentally threatens user privacy.

In the next section, we empirically demonstrate how the split learning framework's inherent shortcomings can be exploited by a malicious server to completely disclose clients' private training sets. Furthermore, in Section 5, we demonstrate that split learning does not protect honest clients from malicious ones, even when the server is honest.

3 FEATURE-SPACE HIJACKING ATTACK

Here, we introduce our main attack against the split learning training protocol—the Feature-space hijacking attack (FSHA). We start in Section 3.1 by detailing the threat model. Then, Section 3.2 introduces the core intuition behind the attack, as well as its formalization. Section 3.3 covers the pragmatic aspects of the attack, demonstrating its effectiveness. Section 3.4 extends the FSHA framework to property inference attacks.

[3] The client can also apply additional layers before the loss computation.
[4] Otherwise, a single server would access both f and s, violating the security of the protocol.

3.1 Threat model

We model a malicious server that aims at inferring private training sets of targets. We assume that the attacker does not have information on the clients participating in the distributed training, except those required to run the split learning protocol. The attacker has no information on the architecture of f and its weights. This also ignores the task on which the distributed model is trained. However, the adversary knows a dataset X_{pub} that follows a distribution similar to that of the clients' training sets. For instance, if the model is trained on histological images, X_{pub} is composed of histological images as well. Nevertheless, no intersection between private training sets and X_{pub} is required. This assumption makes our threat model more realistic and less restrictive than the ones adopted in other works [47, 50], where the adversary is assumed to have direct access to leaked pairs of *smashed* data and private data.

It is crucial to understand that the premise of split learning, and all other distributed learning frameworks, is that the server cannot be trusted. If the server were trusted or honest, we would adopt a centralized approach, where clients send their private training sets to the server that carries out the training process on the combined sets.[5]

3.2 Attack foundations

As discussed in Section 2.2.1, the main vulnerability of split learning resides in the fact that the server has control over the learning process of the clients' network. Indeed, even ignoring the architecture of f and its weights, an adversary can forge a suitable gradient and force f to converge to an arbitrary target function chosen by the attacker. In doing so, the attacker can induce certain properties in the *smashed data* generated by the clients, enabling inference or reconstruction attacks on the underlying private data.

Here, we present a general framework that implements this attack procedure. In such a framework, the malicious server substitutes the original learning task chosen by the clients with a new objective that shapes, on purpose, the codomain/feature-space of f.[6] During the attack, the server exploits its control on the training process to hijack f and steer it towards a specific, target feature-space $\tilde{\mathbf{Z}}$ that is appositely crafted. Once f maps into $\tilde{\mathbf{Z}}$, the attacker can recover the private training instances by locally inverting the known feature-space.

Such an attack encompasses two phases: (1) a **setup phase** where the server hijacks the learning process of f, and (2) a subsequent **inference phase** where the server can freely recover the *smashed* data sent from the clients. Hereafter, we refer to this procedure as **Feature-space Hijacking Attack**, FSHA for short.

Setup phase. The setup phase occurs over multiple training iterations of split learning and is logically divided in two concurrent steps which are depicted in Figures 2a and 2b. In this phase of the attack, the server trains three different networks; namely, $\tilde{f}$, $\tilde{f}^{-1}$ and D. These serve very distinct roles; more precisely:

- $\tilde{f}$: is a pilot network that dynamically defines the target feature-space $\tilde{\mathbf{Z}}$ for the client's network f. Likewise f, $\tilde{f}$ is a mapping between the data-space and a target feature-space $\tilde{\mathbf{Z}}$, where $|\tilde{f}(x)| = |f(x)| = k$.
- $\tilde{f}^{-1}$: is an approximation of the inverse function of $\tilde{f}$. During the training, we use it to guarantee the invertibility of $\tilde{f}$ and recover the private data from *smashed* data during the inference phase.
- D: is a discriminator [20] that indirectly guides f to learn a mapping between the private data and the feature-space defined from $\tilde{f}$. Ultimately, this is the network that substitutes s in the protocol (e.g., Figure 1), and that defines the gradient which is sent to the client during the distributed training process.

The setup procedure also requires an unlabeled dataset X_{pub} that is used to train the three attacker's networks. Observe that this is the only knowledge of the clients' setup that the attacker requires. The effect of X_{pub} on the attack performance will be analyzed in the next section.

As mentioned before, at every training iteration of split learning (i.e., when a client sends *smashed* data to the server), the malicious server trains the three networks in two concurrent steps which are depicted in Figures 2a and 2b. The server starts by sampling a batch from X_{pub} that employs to jointly train $\tilde{f}$ and $\tilde{f}^{-1}$. Here, the server optimizes the weights of $\tilde{f}$ and $\tilde{f}^{-1}$ to make the networks converge towards an auto-encoding function i.e., $\tilde{f}^{-1}(\tilde{f}(x)) = x$. This is achieved by minimizing the loss function:

$$\mathcal{L}_{\tilde{f},\tilde{f}^{-1}} = d(\tilde{f}^{-1}(\tilde{f}(X_{pub})),\ X_{pub}), \tag{1}$$

where d is a suitable distance function, e.g., the Mean Squared Error (MSE).

Concurrently, also the network D is trained. **This is a discriminator [20] that is trained to distinguish between the feature-space induced from $\tilde{f}$ and the one induced from the client's network f.** The network D takes as input $\tilde{f}(X_{pub})$ or $f(X_{priv})$ (i.e., the *smashed* data) and is trained to assign high probability to the former and low probability to the latter. More formally, at each training iteration, the weights of D are tuned to minimize the following loss function:

$$\mathcal{L}_D = \log(1 - D(\tilde{f}(X_{pub}))) + \log(D(f(X_{priv}))). \tag{2}$$

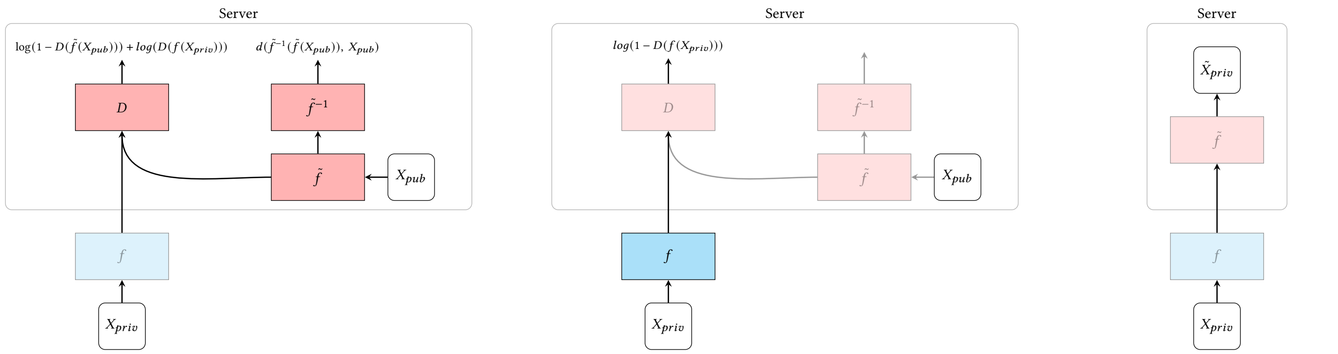
After each local training step for D, the malicious server can then train the network f by sending a suitable gradient signal to the remote client who is performing the training iteration. In particular, this gradient is forged by using D to construct an adversarial loss function for f; namely:

$$\mathcal{L}_f = \log(1 - D(f(X_{priv}))). \tag{3}$$

That is, f is trained to maximize the probability of being missclassified from the discriminator D. In other words, **we require the client's network to learn a mapping to a feature-space that is indistinguishable from the one of $\tilde{f}$.** Ideally, this loss serves as a proxy for the more direct and optimal loss function: $d(f(x),\ \tilde{f}(x))$. However, the attacker has no control over the input of f and must

[5] The centralized learning approach is more efficient and more secure. Clients do not have to communicate among them or compute on data. Furthermore, the server can better handle malicious clients since it runs the learning process in its entirety.

[6] The client's network f can be seen as a mapping between a data-space $\mathbf{X}$ (i.e., where training instances are defined) and a feature-space $\mathbf{Z}$ (i.e., where smashed data are defined).



(a) Attacker's training procedure. **(b) Client's training procedure.** **(c) Inference procedure.**

Figure 2: Schematic representation of the setup and inference process of the feature-space hijacking attack. In the scheme, opaque rectangles depict the neural networks actively taking part to the training. Instead, more transparent rectangles are networks that may participate to the forward propagation but do not modify their weights.

overcome the lack of knowledge about x by relying upon an adversarial training procedure that promotes a topological matching between feature-spaces rather than a functional equivalence between networks.

Attack inference phase. After a suitable number of setup iterations, the network f reaches a state that allows the attacker to recover the private training instances from the *smashed* data. Here, thanks to the adversarial training, the codomain of f **overlaps** with the one of $\tilde{f}$. The latter feature-space is known to the attacker who can trivially recover X_{priv} from the *smashed* data by applying the inverse network $\tilde{f}^{-1}$. Indeed, as the network f is now mapping the data-space into the feature-space $\tilde{\mathbf{Z}}$, the network $\tilde{f}^{-1}$ can be used to map the feature-space $\tilde{\mathbf{Z}}$ back to the data-space, that is:

$$\tilde{X}_{priv} = \tilde{f}^{-1}(f(X_{priv})),$$

where $\tilde{X}_{priv}$ is a suitable approximation of the private training instances X_{priv}. This procedure is depicted in Figure 2c. The quality of the obtained reconstruction will be quantified later in the paper.

We emphasize that the feature-space hijacking attack performs identically on the private-label version of the protocol, e.g., Figure 1b. In this case, at each training step, the server sends arbitrary forged inputs to the clients' final layers and ignores the gradient produced as a response, hijacking the learning process of f as in the previous instance. More generally, in the case of multiple vertical splits, a malicious party can always perform the attack despite its position in the stack. Basically, the attacker can just ignore the received gradient and replace it with the crafted one, leaving the underlying splits to propagate the injected adversarial task. Additionally, the effectiveness of the attack does not depend on the number of participating clients.

In the same way, the feature-space hijacking attack equally applies to extensions of split learning such as *Splitfed* learning [44]. Indeed, in this protocol, the server still maintains control of the learning process of f. The only difference is in how the latter is updated and synchronized among the clients. Interestingly, the attack can be potentially more effective as the server receives bigger batches of *smashed* data that can be used to smooth the learning process of the discriminator.

In the next section, we implement the feature-space hijacking attack, and we empirically demonstrate its effectiveness on various setups.

3.3 Attack implementations

We focus on demonstrating the effectiveness of the attack on the image domain as this is predominant in split learning studies [22, 22, 37, 44, 46–50]. In our experiments, we rely on different image datasets to validate the attack; namely, *MNIST*, Fashion-MNIST [52], *Omniglot* [32] and CelebA [35]. During the attacks, we simulate the clients' training set (i.e., X_{priv}) using the training partition of the datasets, whereas we use their validation sets as X_{pub} owned by the attacker. Note that these sets are always disjointed.

Attack setup. We implement the various networks participating in the attack as deep convolution neural networks. For the client's network f, we rely on a residual network [24] with a funnel structure—a pervasive architecture widely employed for tasks defined on the image domain. In our experiments, we test the proposed attack's effectiveness on increasingly deep splits of f. These are depicted in Figure 3.

The network $\tilde{f}$ (the attacker's pilot network) is constructed by leveraging a different architecture from the one used for f. In particular, the network is chosen to be as simple as possible (i.e., shallow and with a limited number of parameters). Intuitively, this permits to define a very smooth target latent-space $\tilde{\mathbf{Z}}$ and simplify the learning process of f during the attack. The inverse mapping $\tilde{f}^{-1}$ is also a shallow network composed of transposed convolutional layers. The discriminator D is a residual network and is chosen to be deeper than the other employed networks as to force the feature-spaces of f and $\tilde{f}$ to be as similar as possible as they become indistinguishable. During the setup phase, we regularize D with a gradient penalty and use the Wasserstein loss [21] for the adversarial training. This greatly improves the stability of the attack and speeds up the convergence of f. We rely on slightly different architectures for the attacker's networks (i.e., $\tilde{f}$, $\tilde{f}^{-1}$ and D) based

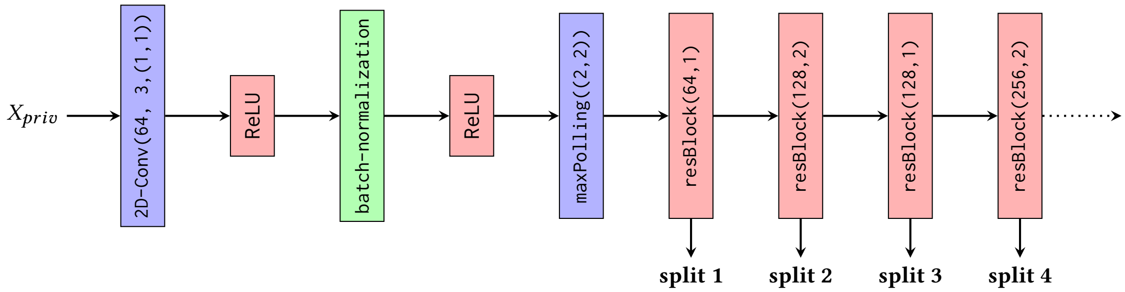
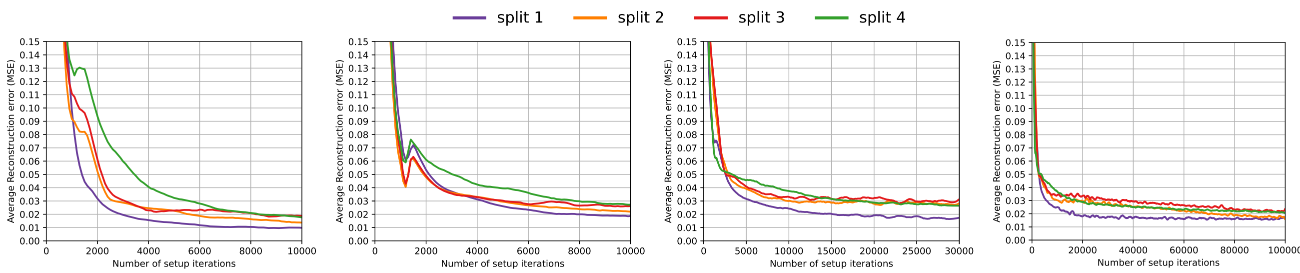


Figure 3: Architecture of the client's network f divided in 4 different depth levels. The internal setup of the adopted residual blocks is described in Algorithm 1.



(a) ***MNIST.*** **(b)** ***Fashion-MNIST.*** **(c)** ***Omniglot.*** **(d)** ***CelebA.***

Figure 4: Reconstruction error of private training instances during the setup phase for four different splits and four different datasets. This is measured as the average MSE between the images normalized in the $[-1, 1]$ interval.

on the depth of the split of f. More detailed information about these, other hyper-parameters, and datasets pre-processing operations are given in Appendix A.

Attack results. During the attack, we use the MSE as the distance function d (see Eq. 1). In the process, we track the attacker's reconstruction error measured as:

$$MSE(\tilde{f}^{-1}(f(X_{priv})), X_{priv}).$$

This is reported in Figure 4 for the four datasets and four different splits of f. In the experiments, different datasets required different numbers of setup iterations to reach adequate reconstructions. Low-entropy distributions like those in *MNIST* and *Fashion-MNIST* can be accurately reconstructed within the first 10^3 setup iterations. Natural images like those in *CelebA*, instead, required up to 10^4 iterations to be properly recovered.

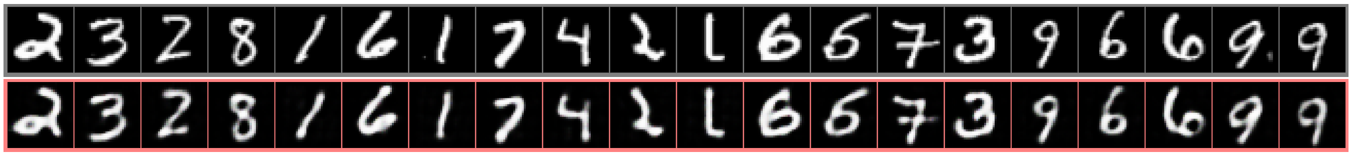
Algorithm 1: Residual Block: `resBlock`:

Data: number of filters: nf, stride s

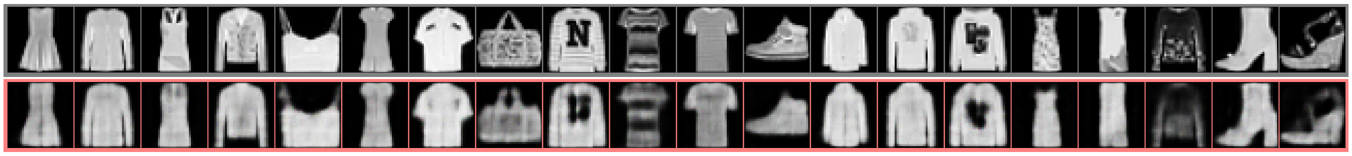
x = `ReLU`(x);
x = `2D-Conv(x, nf, 3, (s,s))`;
x = `ReLU`(x);
x = `2D-Conv(x, nf, 3, (1,1))`;
if $s > 1$ **then**
  x_{in} =`2D-Conv(`x_{in}`, nf, 3, (s,s))`;
return $x_{in} + x$

As the plots in Figure 4 show, there is only a negligible difference in the reconstruction error achieved from attacks performed on the four different splits of f. In the experiments, the depth of the client's network seems to affect the convergence speed of the setup phase; a deep f causes a higher reconstruction error within the same number of iterations compared to a shallower split. This is apparent when we compare the results of the shallowest split (i.e., split 1) with the others. The difference almost disappears when we compare split 3 to split 4. Ideally, we should not observe such performance variations as we expect the attack to improve when f has more parameters and can better approximate the target feature-space $\tilde{\mathbf{Z}}$. However, other factors affect the success of the attack. The performance drop can be easily attributed to instability brought from the deeper architecture of f into the adversarial training procedure, which is known to be ill-conditioned. An additional/concurrent cause could be the over-parameterization of the network f that shortly leads f to overfit X_{priv} in the setup phase.

Nevertheless, even the split 4 allows us to achieve precise reconstructions. These can be observed in Figure 5, where it is clear that the attack provides very accurate reconstructions of the original private data for simple datasets such as *MNIST*. Similarly, the results on *Fashion-MNIST* and *CelebA* prove that the attack can quickly scale to more complex data distributions. More interestingly, the *Omniglot* dataset highlights the generalization capability of the feature-space hijacking attack. The *Omniglot* dataset is often used as a benchmark for one-shot learning and contains 1623 different

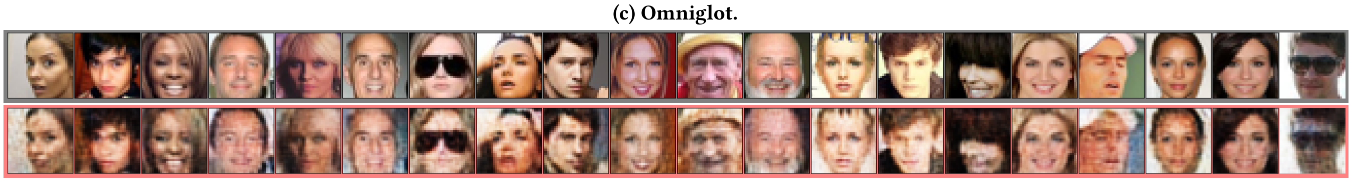


(a) MNIST.



(b) Fashion-MNIST.

(c) Omniglot.



(d) CelebA.

Figure 5: Examples of inference of private training instances from *smashed data* for four datasets for the split 4 of f. Within each panel, the first row (i.e., gray frame) reports the original data, whereas the second row (i.e., red frame) depicts the attacker's reconstruction. The reported examples are chosen randomly from X_{priv}.

classes with a limited number of examples each. The attack's performance on this dataset suggests that the proposed technique can reach a good generalization level over private data distributions. We will investigate this property more rigorously later in the section.

Hereafter, we will report results only for the split 4 as this represents the worst-case scenario for our attack. Moreover, it also better captures the best practices [5, 50] of split learning.[7]

Feature-space generalization. The training set X_{pub} employed by the server can critically impact the attack's effectiveness. This is used to train the attacker's models and indirectly defines the target feature-space imposed on f. Intuitively, to reach high-quality reconstruction, this should be distributed as similar as possible to the private training sets owned by the clients. However, under strict assumptions, the attacker may collect data instances that are not sufficiently representative. Next, we test the resilience of the Feature-space Hijacking Attack against unsuitable choices of X_{pub}.

To simulate this scenario, we create artificially mangled training sets X_{pub} for the *MNIST* dataset and test the attack's effectiveness accordingly. **In the experiment, the mangling operation consists of removing all the instances of a specific class from X_{pub} while leaving X_{priv} (the training set used by the clients) unchanged.** For instance, in the case of the *MNIST* dataset, we remove from X_{pub} all the images representing a specific digit. Then,

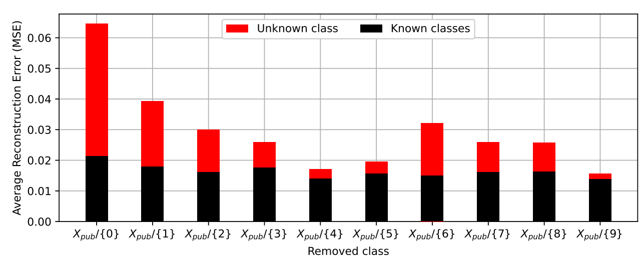
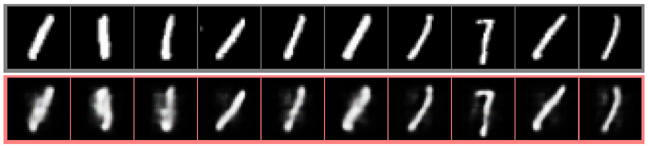


Figure 6: Average reconstruction error for FSHA with mangled X_{pub} for the *MNIST* dataset. Each bar represents the final reconstruction error of private data obtained with an FSHA based on a X_{pub} mangled of a specific class. Black bars report the average reconstruction error of private data instances of classes known to the attacker. Instead, red bars report the average reconstruction error of private data instances for the removed class. In the attacks, we used 15000 setup iterations.

[7] Deeper architectures for f are assumed to make it harder for an attacker to recover information from the *smashed* data as this has been produced using more complex transformations.

(a) Reconstruction 0 with $X_{pub}/\{0\}$.



(b) Reconstruction 1 with $X_{pub}/\{1\}$.

Figure 7: Two examples of inference of private training instances from *smashed data* given mangled X_{pub}. In the panel (a), the adversary carried out the attack without ever directly observing training instances representing the digit "0". Panel (b) reproduces the same result for the digit "1". Only the reconstruction of instances of the class unknown to the attacker are reported. Those have been sampled from X_{priv}.

we test the attack's capability to reconstruct instances of the removed class i.e., data instances that the attacker has never observed during the setup phase.

Interestingly, the attack seems quite resilient to an incomplete X_{pub}. The results are depicted in Figure 7 for 10 different attacks carried out with X_{pub} stripped of a specific class. For each attack, the average reconstitution error for the unknown classes (i.e., red bars) is only slightly larger than the one for the classes represented in X_{pub}. Here, the attacker can successfully recover a suitable approximation of instances of the unobserved class by interpolating over the representations of observed instances. The only outlier is the case $X_{pub}/\{0\}$. Our explanation is that the digit zero is peculiar, so it is harder to describe it with a representation learned from the other digits. Nevertheless, as depicted in Figure 7, the FSHA provides an accurate reconstruction also in the cases of 0 and 1.

3.4 Property inference attacks

In the previous setup, we demonstrated that it is possible to recover the entire input from the *smashed* data. However, this type of inference may be sub-optimal for an attacker who may be interested in inferring only a few specific attributes/properties of the private training instances (e.g., the gender of the patients in medical records); rather than reconstructing X_{priv} entirely. This form of inference was introduced in [8] and extended to neural networks in [18]. Property inference is simpler to perform and more robust to possible defensive mechanisms (see Section 4). Next, we briefly show how the Feature-space Hijacking Attack can be extended to perform property inference attacks.

As discussed in Section 3.2, we can force arbitrary properties on the *smashed* data produced by the clients by forging a tailored feature-space $\tilde{\text{Z}}$ and forcing the clients' network f to map into it.

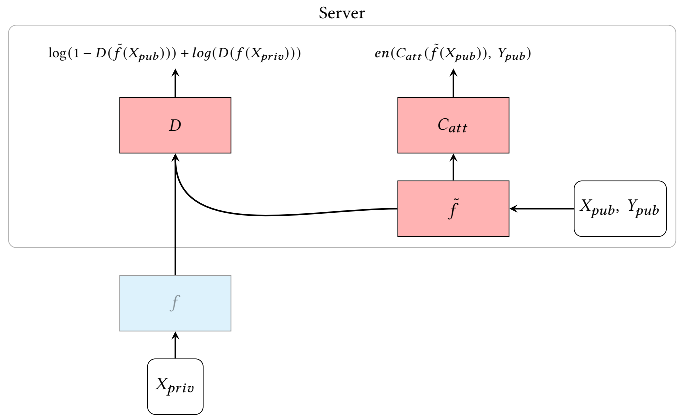


Figure 8: Schematic representation of the training process of the server's networks for the attribute inference attack. In the figure, the network C_{att} substitutes $\tilde{f}^{-1}$ and en refers to a suitable entropy measure for the classification task.

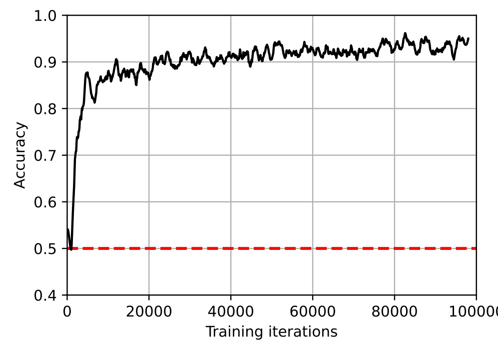


Figure 9: Example of attribute inference attack over the CelebA dataset. The plot reports the accuracy in inferring the attribute *"gender"* from instances of X_{priv} during the setup phase of the attack.

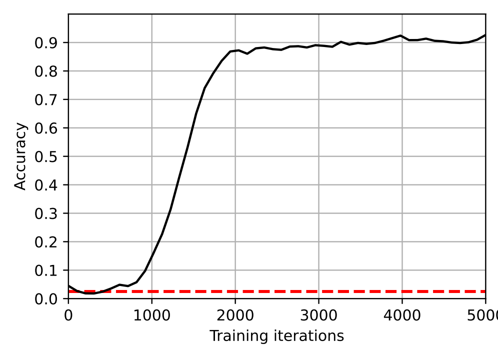


Figure 10: Classification accuracy during the setup phase of the FSHA performed on split 3 on the AT&T dataset. The red, dashed line marks random guessing.

The feature-space $\tilde{\mathbf{Z}}$ is dynamically created by training a pilot network $\tilde{f}$ in a task that encodes the target property. In the attack of Figure 2, we requested the invertibility of $\tilde{\mathbf{Z}}$ by training $\tilde{f}$ in an auto-encoding task with the support of a second network $\tilde{f}^{-1}$. Conversely, we can force the *smashed* data to leak information about a specific attribute by conditioning the feature-space $\tilde{\mathbf{Z}}$ with a classification task.

It is enough to substitute the network $\tilde{f}^{-1}$ with a classifier C_{att} that is trained to detect a particular attribute in the data points of $\tilde{\mathbf{Z}}$. However, unlike the previous formulation of the attack, the attacker has to resort to a supervised training set (X_{pub}, Y_{pub}) to define the target attribute. Namely, each instance of the attacker's dataset X_{pub} must be associated with a label that expresses the attribute/property *att* that the attacker wants to infer from the *smashed* data. The setup procedure for this attack is depicted in Figure 8, whereas the training procedure for the clients' network f remains the same.

3.4.1 Inferring binary attributes. In case of a binary attribute, the attacker has to train C_{att} in a binary classification using a binary cross-entropy loss function:

$$\mathcal{L}_{\tilde{f},C_{att}} = \begin{cases} \log(C_{att}(\tilde{f}(X_{pub}))) & \text{if } Y_{pub} = 1 \\ -\log(1 - C_{att}(\tilde{f}(X_{pub}))) & \text{if } Y_{pub} = 0 \end{cases}.$$

Here, we implement the network C_{att} to be as simple as possible to maximize the separability of the classes directly on $\tilde{\mathbf{Z}}$. In particular, we model C_{att} as a linear model by using a single dense layer. In this way, we force the representations of the classes to be linearly separable, simplifying the inference attack once the adversarial loss has forced the topological equivalence between the codomains of f and $\tilde{f}$. We leave the other models and hyper-parameters unchanged.

We validate this inference procedure on the CelebA dataset. This mainly includes frontal shoots (e.g., Figure 5d) which have been annotated with 40 different binary attributes such as *"blonde"*, *"wearing hat"*, *"smiling"*, etc. In the experiment, we try to infer the binary attribute *"gender"* (i.e., 0 ="*woman*"; 1 ="*man*") from the private training instances used by the clients. During the attack, we track the accuracy of the inference performed by the network C_{att}. This is reported in Figure 9, where the inference attack reaches an accuracy of $\sim 95\%$.

It is important to note that the property inference attack can be extended to any feature or task. For instance, the attacker can infer multiple attributes simultaneously by training C_{att} in a multi-label classification, rather than a binary one. The same applies to multi-class classification and regression tasks. In this direction, the only limitation is the attacker's capability to collect suitable labeled data to set up the attack.

3.4.2 Inferring categorical attributes. The attacker can infer categorical attributes rather than binary ones by training the network C_{att} in a multi-class classification and providing suitable labels to X_{pub}. To implement this scenario, we use the AT&T dataset which is composed of frontal shots of 40 different individuals: 10 images each. This dataset has been previously used in [25]. Here, the server wants to identify the individuals represented on each of the images used during the distributed training. That is, the attacker wants to correctly assign one of the 40 possible identities (i.e., classes) to each received *smashed* data.

Given the small cardinality of the *AT&T* dataset, we use the split 3 of f to implement the attack as the split 4 quickly overfits within initial iterations of the setup phase. As for the previous attack, we use a single fully-connected layer to implement C_{att} (with 40 output units), but we train the model with a categorical cross-entropy loss function. Figure 10 reports the evolution of the classification accuracy during the setup phase of the attack on X_{priv}. Within a few initial iterations, the attacker reaches an accuracy higher than 90% in classifying the images of the 40 different individuals composing the set.

3.5 Attack Implications

The implemented attacks demonstrated how a malicious server could subvert the split learning protocol and infer information over the clients' private data. Unlike previous attacks in collaborative learning [25], here, the adversary can recover the single training instances from the clients, rather than only prototypical examples. This allows the attacker to fully expose the distribution of the private data of each client. Indeed, the server could determine which client owns a training instance upon receiving the clients' disjointed *smashed* data.

In the next section, we discuss the shortcomings of defense strategies proposed to prevent inference attacks.

4 ON DEFENSIVE TECHNIQUES

As demonstrated by our attacks, simply applying a set of neural layers over raw data cannot yield a suitable security level, especially when the adversary controls the learning process. As a matter of fact, as long as the attacker exerts influence on the target function of the clients' network, the latter can always be lead to insecure states. Unfortunately, there does not seem to be any way to prevent the server from controlling the learning process without rethinking the entire protocol from scratch. Next, we reason about the effectiveness of possible defense strategies.

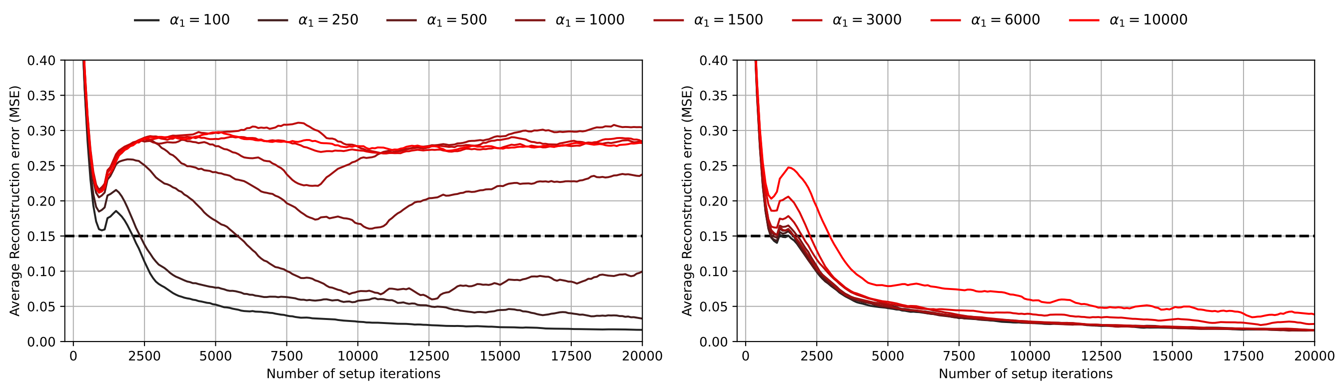
4.1 Distance correlation minimization

In [47, 50], the authors propose to artificially reduce correlation between raw input and *smashed* data by adding a regularization during the training of the distributed model in split learning. In particular, they resort to *distance correlation* [43]—a well-established measure of dependence between random vectors. Here, the clients optimize f to produce outputs that minimize both the target task loss (e.g., a classification loss) and the distance correlation. This regularization aims at preventing the propagation of information that is not necessary to the final learning task of the model from the private data to the *smashed* one. Intuitively, this is supposed to hamper the reconstruction of X_{priv} from an adversary that has access to the *smashed* data.

More formally, during the split learning protocol, the distributed model is trained to jointly minimize the following loss function:

$$\alpha_1 \cdot DCOR(X_{priv}, f(X_{priv})) + \alpha_2 \cdot TASK(y, s(f(X_{priv}))), \quad (4)$$

where $DCOR$ is the distance correlation metrics, $TASK$ is the task loss of the distributed model (e.g., cross-entropy for a classification



(a) Default: task loss $\times 1$

(b) Re-weighted: task loss $\times 25$.

Figure 11: Effect of the distance correlation minimization defense on FSHA for the *MNIST* dataset. Each curve in the figures depicts the reconstruction error of private data during the setup phase for a different value of α_1 imposed by the client. The two panels report the effect of scaling the task loss (e.g., α_2) server-side.

task), and y is a suitable label for the target task (if any). In the equation, the hyper-parameters α_1 and α_2 define the relevance of distance correlation in the final loss function, creating and managing a tradeoff between data privacy (i.e., how much information an attacker can recover from the smashed data) and model's utility on the target task (e.g., the accuracy of the model in a classification task). Note that the distance correlation loss depends on just the client's network f and the private data X_{priv}. Thus, it can be computed and applied locally on the client-side without any influence from the server.

While the approach proposed in [47, 50] seems to offer reasonable security in the case of a passive adversary, unfortunately, it is ineffective against the feature-space hijacking attack that influences the learning process of f. As a matter of fact, the learning objective injected by the attacker will naturally negate the distance correlation minimization, circumventing its effect. Moreover, this defensive technique does not prevent the property inference attack detailed in Section 3.4.

Figure 11a reports on the impact of the distance correlation minimization on the FSHA on the *MNIST* dataset for different values of α_1. In the plot, we start from $\alpha_1 = 100$, which is the smallest assignment of α_1 that does not affect the attack's performance, and we increase it until we reach impractical high values e.g., $\alpha_1 = 10000$. As shown in the plot, the defense becomes effective when α_1 reaches very high values. In these cases, the privacy loss completely eclipses the task loss of the distributed model (i.e., Eq. 4). As a result, any improvement of f in reducing the task loss becomes either impossible or extremely slow. Intuitively, this value of α_1 prevents the distributed model from achieving any utility on the imposed task. This is so regardless of whether the model is trained on the task originally selected by the clients or the adversarial task enforced by the malicious server.

Nevertheless, even if the clients set the parameter α_1 to a large value, they have no effective method to control α_2 if the server is malicious. Indeed, even in the label-private setting of split learning (i.e., Figure 1b), the server can arbitrarily determine the training objective for the model and adjust the task loss $TASK$. Trivially, this allows the attacker to indirectly control the ratio between the privacy-loss (which is performed locally at the client) and the target loss (i.e., the adversarial loss imposed by the attacker), nullifying the effect of a heavy regularization performed at the client-side. Figure 11b explicates how the malicious server circumvents the client-side defense by just scaling the adversarial loss function by a factor of 25. In this case, even impractically large values of α_1 are ineffective .

To improve the defense mechanism above, one could apply gradient clipping on the gradient sent by the server during the training. However, gradient clipping further reduces the utility of the model as it weakens the contribution of the target loss function in case of an honest server.

Additionally, it is possible to devise a more general strategy and allow a malicious server to adopt advanced approaches to evade the defenses implemented in [47, 50]. Indeed, distance correlation can be easily circumvented by forging a suitable target feature-space. The key idea is that the attacker can create an "*adversarial*" feature-space that minimizes the distance correlation but allows the attacker to obtain a precise reconstruction of the input. We detail this possibility in the Appendix B. Once the adversarial feature-space is obtained, the attacker can hijack f, minimize the distance correlation loss of f, and recover the original data precisely.

4.2 Detecting the attack

Alternatively, clients could detect the feature-space hijacking attack during the training phase and then halt the protocol. Unfortunately, detecting the setup phase of the attack seems to be a complex task. Here, clients could continuously test the effectiveness of the network on the original training task and figure out if the training objective has been hijacked. However, clients have no access to the full network during training and cannot query it to detect possible anomalies. This is also true for the private-label scenario, i.e., Figure 1b of split learning, where clients compute the loss function on their devices. Indeed, in this case, the attacker can simply provide fake inputs to f' (see Figure 1b) that has been forged to minimize the clients' loss. For instance, the attacker can simply train a second dummy network $\tilde{s}$ during the setup phase and send its output to the client. Here, the network $\tilde{s}$ receives the *smashed* data as input and it is directly trained with the gradient received from f' to minimize the loss function chosen by the client. To note that, during the attack, the network f does not receive the gradient from $\tilde{s}$ but only from D.

5 THE SECURITY OF SPLIT LEARNING AGAINST MALICIOUS CLIENTS

In recent works [49], the authors claim that the splitting methodology could prevent client-side attacks that were previously devised against federated learning, such as the GAN-based attack [25]. Actually, we show that the attacks in [25] (albeit with some minimal adaptations) remain applicable even within the split learning framework.

Client-side attack on Federated Learning. The attack [25] works against the collaborative learning of a classifier C trained to classify n classes, say $y_1, \ldots, y_n$. Here, a malicious client intends to reveal prototypical examples of a target class y_t, held by one or more honest clients. During the attack, the malicious client exerts control over a class $y_{\tilde{t}}$ that is used to actively poison the trained model and improve the reconstruction of instances y_t.

To perform the inference attack, the malicious client trains a local generative model G to generate instances of the target class y_t. During each iteration, the attacker samples images from G, assigns the label $y_{\tilde{t}}$ to these instances, and uses them to train the model C according to the learning protocol. Once the clients have contributed their training parameters, the attacker downloads the updated model C from the server and uses it as the discriminator [20] to train the generative model G. The confidence of C on the class y_t is used as the discriminator's output and maximized in the loss function of G. Once the generator has been trained, the attacker can use it to reproduce suitable target class instances y_t.

5.1 Client-side Attack on Split Learning

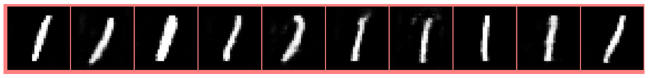
The attack [25] can be performed on split learning under the same threat model. To note how, in this setup, the split learning server is honest, whereas the malicious client does not know the data distribution of the other clients' training sets.

Considering the private-label case (i.e., Figure 1b), a malicious client exerts a strong influence over the learning process of the shared model $C = f'(s(f(\cdot))$ and can set up an attack similar to the one performed on federated learning. Here, the attacker trains a generator G by using the distributed model $C = f'(s(f(\cdot)))$ as the discriminator by just providing suitable pairs (input, label) during the split learning protocol. This attack procedure is summarized in Algorithm 2. During the attack, the only impediment is the limited control of the attacker on the weights update procedure of the network s hosted by the server. **Indeed, to soundly train the generator using the adversarial loss based on the distributed model C, the attacker must prevent the update of s while training the generator G.** However, the weights update operation

Algorithm 2: Client-side attack [25] in split learning.

Data: Number of training iterations: N, Target class: y_t, Dummy class for poisoning $y_{\tilde{t}}$, Scaling factor gradient: ϵ

```
/* Initialize the local generative model */
G = initGenerator();
for i in [1, N] do
    /* Download updated network splits */
    f, f' = get_models();
    /* Alterning poisoning attack and adversarial training */
    /* (more sophisticated scheduler may be used) */
    if i%2 == 0 then
        poisoning = True;
    else
        poisoning = False
    /* -- Start distributed forward-propagation */
    /* Sample data instances from the generator G */
    x ~ G;
    z = f(x);
    /* Send smashed data to the server and get s(f(x)) back */
    z' = send_get_forward(z);
    /* Apply final layers and compute the probability for each class */
    p = f'(z');
    if poisoning then
        /* Dummy label */
        y = y_t~;
    else
        /* Target label */
        y = y_t;
    /* Compute loss */
    L = cross-entropy(y, p);
    /* -- Start distributed back-propagation */
    /* Compute local gradient till s */
    ∇_f' = compute_gradient(f', L);
    if not poisoning then
        /* Scale down gradient */
        ∇_f' = ε · ∇_f';
    else
        /* Apply gradient on f' */
        f' = apply(f', ∇_f')
    /* Send gradient to the server and receive gradient till f */
    ∇_s = send_get_gradient(∇_f');
    if not poisoning then
        /* Scale back gradient */
        ∇_s = (1/ε) · ∇_s;
    /* Compute local gradient till G */
    ∇_f = compute_gradient(f, ∇_s);
    if poisoning then
        /* Apply gradient on f */
        f = apply(f, ∇_f)
    else
        /* Compute local gradient till G's input */
        ∇_G = compute_gradient(G, ∇_f);
        /* Apply gradient on the generator */
        G = apply(G, ∇_G)
```



(a) ***MNIST*** $y_t = 1$

(b) ***MNIST*** $y_t = 2$

(c) ***MNIST*** $y_t = 3$

(d) ***AT&T*** $y_t = 1$

(e) ***AT&T*** $y_t = 2$

(f) ***AT&T*** $y_t = 3$

Figure 12: Results from the client-side attack performed on split learning. The images are random samples from the generator trained via Algorithm 2 on three attacks with different target classes. For the results on the dataset *AT&T,* we report also an instance of the target class in the leftmost corner of the panel in a gray frame.

of s is performed by the server and cannot be directly prevented by the malicious client.[8]

The gradient-scaling trick. Nevertheless, this limitation can be easily circumvented by manipulating the gradient sent and received by the server during the split learning protocol. In particular, the malicious client can resort to gradient-scaling to make negligible the training operation's impact on s. Here, before sending the gradient $\nabla_{f'}$ produced from f' to s, the client can multiply $\nabla_{f'}$ by a very small constant ϵ; that is:

$$\nabla_{f'} = \epsilon \cdot \nabla_{f'}. \tag{5}$$

This operation makes the magnitude of $\nabla_{f'}$, and so the magnitude of the weights update derived from it on s, negligible, **thus preventing any functional change in the weights of** s. Ideally, this is equivalent to force the server to train s with a learning rate close to zero.

Then, once s has performed its back-propagation step and sent the gradient ∇_s to f, the malicious client scales back ∇_s to its original magnitude by multiplying it by the inverse of ϵ; that is:

$$\nabla_s = \frac{1}{\epsilon} \cdot \nabla_s. \tag{6}$$

This allows the attacker to recover a suitable training signal for the generator G that follows the back-propagation chain. To note, the malicious client does not update either the weights of f or those of f' in the process. Eventually, the gradient-scaling operation allows the malicious client to train the generator using the distribute model C as a discriminator. We demonstrate the soundness of this procedure later in this section.

Although the gradient-scaling trick may provide a cognizant server an easy way to detect the attackers, a malicious client can always find a trade-off between attack secrecy and attack performance by choosing suitable assignments of ϵ. As a matter of fact, it is hard for the server to distinguish the scaled gradient from the one achieved by a batch of *easy examples* (that is, data instances that are correctly classified by the model with high confidence.)

The poisoning step of the attack [25] can be performed without any modification. The malicious client has to assign the label $y_{\tilde{t}}$ to instances sampled from the generator G and run the standard split learning training procedure. In this process, the attacker updates the weights of all the participating networks but G. However, during the attack, the malicious client must alternate between a poisoning step and a genuine training iteration for the generator as these cannot be performed simultaneously due to the gradient-scaling trick required to train the generator. Alternatively, the attacker can impersonate an additional client in the protocol and perform the poisoning iterations separately.

Attack validation. To implement the attack, we rely on architectures and hyper-parameters compatible with those originally used in [25] and perform the attack on the *MNIST* and *AT&T* datasets. More details are given in Appendix A.1. We use $\epsilon{=}1^{-5}$ in the "*gradient-scaling trick*". In our setup, we model 10 honest clients and a single malicious client who performs the attack described in Algorithm 2. In the process, we use the standard sequential training procedure of split learning [22]. However, the attack equally applies to parallel extensions such as *Splitfed* learning [44]. We run the attack for 10000 global training iterations. The results are reported in Figure 12 for three attacks targeting different y_t, and prove the generator is successfully reproducing instances of the target class.

6 FINAL REMARKS

In the present work, we described various structural vulnerabilities of split learning and showed how to exploit them and violate the protocol's privacy-preserving property. Here, a malicious server can accurately reconstruct, or infer properties on, training instances. Additionally, we have shown that defensive techniques devised to protect split learning can be easily evaded.

While federated learning exhibits similar vulnerabilities, split learning appears worse since it consistently leaks more information. Furthermore, it makes it even harder to detect ongoing inference attacks. Indeed, in standard federated learning, all participants store the neural network in its entirety, enabling simple detection mechanisms that, if nothing else, can thwart unsophisticated attacks.

[8] In these cases of f and f', the back-propagation is performed client-side, and the malicious client can explicitly avoid the weights update operations.

REFERENCES

[1] 2020. Acuratio. https://www.acuratio.com. (2020).

[2] 2020. OpenMined: SplitNN. https://blog.openmined.org/tag/splitnn/. (2020).

[3] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep Learning with Differential Privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. Association for Computing Machinery, New York, NY, USA, 308–318. https://doi.org/10.1145/2976749.2978318

[4] Ali Abedi and Shehroz S. Khan. 2020. FedSL: Federated Split Learning on Distributed Sequential Data in Recurrent Neural Networks. (2020). arXiv:cs.LG/2011.03180

[5] Sharif Abuadbba, Kyuyeon Kim, Minki Kim, Chandra Thapa, Seyit A. Camtepe, Yansong Gao, Hyoungshick Kim, and Surya Nepal. 2020. Can We Use Split Learning on 1D CNN Models for Privacy Preserving Training? (2020). arXiv:cs.CR/2003.12365

[6] Adam James Hall. 2020. Split Neural Networks on PySyft. https://medium.com/analytics-vidhya/split-neural-networks-on-pysyft-ed2abf6385c0. (2020).

[7] George J Annas. 2003. HIPAA regulations - a new era of medical-record privacy? *The New England journal of medicine* 348, 15 (April 2003), 1486—1490. https://doi.org/10.1056/nejmlim035027

[8] Giuseppe Ateniese, Luigi V. Mancini, Angelo Spognardi, Antonio Villani, Domenico Vitali, and Giovanni Felici. 2015. Hacking Smart Machines with Smarter Ones: How to Extract Meaningful Data from Machine Learning Classifiers. *Int. J. Secur. Netw.* 10, 3 (Sept. 2015), 137–150. https://doi.org/10.1504/IJSN.2015.071829

[9] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. 2020. How To Backdoor Federated Learning. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research)*, Silvia Chiappa and Roberto Calandra (Eds.), Vol. 108. PMLR, Online, 2938–2948. http://proceedings.mlr.press/v108/bagdasaryan20a.html

[10] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. 2019. Analyzing Federated Learning through an Adversarial Lens *(Proceedings of Machine Learning Research)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.), Vol. 97. PMLR, Long Beach, California, USA, 634–643. http://proceedings.mlr.press/v97/bhagoji19a.html

[11] K. A. Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé M Kiddon, Jakub Konečný, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. 2019. Towards Federated Learning at Scale: System Design. In *SysML 2019*. https://arxiv.org/abs/1902.01046 To appear.

[12] Brendan McMahan, Ramesh Raskar, Otkrist Gupta, Praneeth Vepakomma, Hassan Takabi, Jakub Konečný. 2019. CVPR Tutorial On Distributed Private Machine Learning for Computer Vision: Federated Learning, Split Learning and Beyond. https://nopeekcvpr.github.io. (2019).

[13] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. (2020). arXiv:cs.CL/2005.14165

[14] Iker Ceballos, Vivek Sharma, Eduardo Mugica, Abhishek Singh, Alberto Roman, Praneeth Vepakomma, and Ramesh Raskar. 2020. SplitNN-driven Vertical Partitioning. (2020). arXiv:cs.LG/2008.04137

[15] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. 2020. Local Model Poisoning Attacks to Byzantine-Robust Federated Learning. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 1605–1622. https://www.usenix.org/conference/usenixsecurity20/presentation/fang

[16] David Froelicher, Juan R. Troncoso-Pastoriza, Apostolos Pyrgelis, Sinem Sav, Joao Sa Sousa, Jean-Philippe Bossuat, and Jean-Pierre Hubaux. 2020. Scalable Privacy-Preserving Distributed Learning. (2020). arXiv:cs.CR/2005.09532

[17] Clement Fung, Chris J. M. Yoon, and Ivan Beschastnikh. 2020. The Limitations of Federated Learning in Sybil Settings. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*. USENIX Association, San Sebastian, 301–316. https://www.usenix.org/conference/raid2020/presentation/fung

[18] Karan Ganju, Qi Wang, Wei Yang, Carl A. Gunter, and Nikita Borisov. 2018. Property Inference Attacks on Fully Connected Neural Networks Using Permutation Invariant Representations. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*. Association for Computing Machinery, New York, NY, USA, 619–633. https://doi.org/10.1145/3243734.3243834

[19] Y. Gao, M. Kim, S. Abuadbba, Y. Kim, C. Thapa, K. Kim, S. A. Camtep, H. Kim, and S. Nepal. 2020. End-to-End Evaluation of Federated Learning and Split Learning for Internet of Things. In *2020 International Symposium on Reliable Distributed Systems (SRDS)*. 91–100. https://doi.org/10.1109/SRDS51746.2020.00017

[20] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger (Eds.), Vol. 27. Curran Associates, Inc., 2672–2680.

[21] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. 2017. Improved Training of Wasserstein GANs. In *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc., 5767–5777.

[22] Otkrist Gupta and Ramesh Raskar. 2018. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications* 116 (2018), 1 – 8. https://doi.org/10.1016/j.jnca.2018.05.003

[23] M. Hao, H. Li, X. Luo, G. Xu, H. Yang, and S. Liu. 2020. Efficient and Privacy-Enhanced Federated Learning for Industrial Artificial Intelligence. *IEEE Transactions on Industrial Informatics* 16, 10 (2020), 6532–6542. https://doi.org/10.1109/TII.2019.2945367

[24] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778. https://doi.org/10.1109/CVPR.2016.90

[25] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. 2017. Deep Models Under the GAN: Information Leakage from Collaborative Deep Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. Association for Computing Machinery, New York, NY, USA, 603–618. https://doi.org/10.1145/3133956.3134012

[26] J. Jeon and J. Kim. 2020. Privacy-Sensitive Parallel Split Learning. In *2020 International Conference on Information Networking (ICOIN)*. 7–9. https://doi.org/10.1109/ICOIN48656.2020.9016486

[27] J. Kang, Z. Xiong, D. Niyato, S. Xie, and J. Zhang. 2019. Incentive Mechanism for Reliable Federated Learning: A Joint Optimization Approach to Combining Reputation and Contract Theory. *IEEE Internet of Things Journal* 6, 6 (2019), 10700–10714. https://doi.org/10.1109/JIOT.2019.2940820

[28] J. Kim, Sungho Shin, Yeonguk Yu, Junseok Lee, and Kyoobin Lee. 2020. Multiple Classification with Split Learning. *ArXiv* abs/2008.09874 (2020).

[29] Yusuke Koda, Jihong Park, Mehdi Bennis, Koji Yamamoto, Takayuki Nishio, and Masahiro Morikura. 2019. One Pixel Image and RF Signal Based Split Learning for MmWave Received Power Prediction *(CoNEXT '19 Companion)*. Association for Computing Machinery, New York, NY, USA, 54–56. https://doi.org/10.1145/3360468.3368176

[30] Jakub Konečný, H. Brendan McMahan, Daniel Ramage, and Peter Richtárik. 2016. Federated Optimization: Distributed Machine Learning for On-Device Intelligence. (2016). arXiv:cs.LG/1610.02527

[31] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2017. Federated Learning: Strategies for Improving Communication Efficiency. (2017). arXiv:cs.LG/1610.05492

[32] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. 2015. Human-level concept learning through probabilistic program induction. *Science* 350, 6266 (2015), 1332–1338. https://doi.org/10.1126/science.aab3050 arXiv:https://science.sciencemag.org/content/350/6266/1332.full.pdf

[33] M. Langer, Z. He, W. Rahayu, and Y. Xue. 2020. Distributed Training of Deep Learning Models: A Taxonomic Perspective. *IEEE Transactions on Parallel and Distributed Systems* 31, 12 (2020), 2802–2818. https://doi.org/10.1109/TPDS.2020.3003307

[34] Wei Yang Bryan Lim, Jer Shyuan Ng, Zehui Xiong, Dusit Niyato, Cyril Leung, Chunyan Miao, and Qiang Yang. 2020. Incentive Mechanism Design for Resource Sharing in Collaborative Edge Learning. (2020). arXiv:cs.NI/2006.00511

[35] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. 2015. Deep Learning Face Attributes in the Wild. In *Proceedings of International Conference on Computer Vision (ICCV)*.

[36] Kamalesh Palanisamy, Vivek Khimani, Moin Hussain Moti, and D. Chatzopoulos. 2020. SplitEasy: A Practical Approach for Training ML models on Mobile Devices in a split second. *ArXiv* abs/2011.04232 (2020).

[37] Maarten G. Poirot, Praneeth Vepakomma, Ken Chang, Jayashree Kalpathy-Cramer, Rajiv Gupta, and Ramesh Raskar. 2019. Split Learning for collaborative deep learning in healthcare. (2019). arXiv:cs.LG/1912.12115

[38] Alec Radford, Luke Metz, and Soumith Chintala. 2016. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1511.06434

[39] F. Sattler, S. Wiedemann, K. R. Müller, and W. Samek. 2020. Robust and Communication-Efficient Federated Learning From Non-i.i.d. Data. *IEEE Transactions on Neural Networks and Learning Systems* 31, 9 (2020), 3400–3413. https://doi.org/10.1109/TNNLS.2019.2944481

[40] Vivek Sharma, Praneeth Vepakomma, Tristan Swedish, Ken Chang, Jayashree Kalpathy-Cramer, and Ramesh Raskar. 2019. ExpertMatcher: Automating ML Model Selection for Clients using Hidden Representations. (2019).

arXiv:cs.CV/1910.03731
[41] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-Preserving Deep Learning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*. Association for Computing Machinery, New York, NY, USA, 1310–1321. https://doi.org/10.1145/2810103.2813687
[42] Abhishek Singh, Praneeth Vepakomma, Otkrist Gupta, and Ramesh Raskar. 2019. Detailed comparison of communication efficiency of split learning and federated learning. (2019). arXiv:cs.LG/1909.09145
[43] Gabor Szekely, Maria Rizzo, and Nail Bakirov. 2008. Measuring and Testing Dependence by Correlation of Distances. *The Annals of Statistics* 35 (04 2008). https://doi.org/10.1214/009053607000000505
[44] Chandra Thapa, M. A. P. Chamikara, and Seyit Camtepe. 2020. SplitFed: When Federated Learning Meets Split Learning. (2020). arXiv:cs.LG/2004.12088
[45] Chandra Thapa, M. A. P. Chamikara, and Seyit A. Camtepe. 2020. Advancements of federated learning towards privacy preservation: from federated learning to split learning. (2020). arXiv:cs.LG/2011.14818
[46] Valeria Turina, Zongshun Zhang, Flavio Esposito, and Ibrahim Matta. 2020. Combining Split and Federated Architectures for Efficiency and Privacy in Deep Learning. In *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT '20)*. Association for Computing Machinery, New York, NY, USA, 562–563. https://doi.org/10.1145/3386367.3431678
[47] Praneeth Vepakomma, Otkrist Gupta, Abhimanyu Dubey, and Ramesh Raskar. 2019. Reducing leakage in distributed deep learning for sensitive health data. (05 2019).
[48] Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. 2018. Split learning for health: Distributed deep learning without sharing raw patient data. (2018). arXiv:cs.LG/1812.00564
[49] Praneeth Vepakomma, Tristan Swedish, Ramesh Raskar, Otkrist Gupta, and Abhimanyu Dubey. 2018. No Peek: A Survey of private distributed deep learning. (2018). arXiv:cs.LG/1812.03288
[50] Praneeth Vepakomma, Tristan Swedish, Ramesh Raskar, Otkrist Gupta, and Abhimanyu Dubey. 2018. No Peek: A Survey of private distributed deep learning. (2018). arXiv:cs.LG/1812.03288
[51] C. Wang, X. Wei, and P. Zhou. 2020. Optimize Scheduling of Federated Learning on Battery-powered Mobile Devices. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 212–221. https://doi.org/10.1109/IPDPS47924.2020.00031
[52] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. (2017). arXiv:cs.LG/1708.07747

APPENDICES

A ARCHITECTURES AND EXPERIMENTAL SETUPS

The employed architectures are reported in Table .1. For the definition of convolutional layers we use the notation:

"*(number of filters, kernel size, stride, activation function)*",

whereas for dense layers:

"*(number of nodes, activation function)*".

The residual block used to build the discriminator D is described in Algorithm 1.

To construct the clients' network f, we use a standard convolutional neural network (CNN) composed of convolutional layers and pooling layers. The attacker's network $\tilde{f}$ outputs a tensor with the same shape of f but diverges in every other parameter. Besides being a CNN as well, f builds on different kernel sizes, kernel numbers, and activation functions; $\tilde{f}$ does not include pooling layers, but it reduces the kernel's width by a larger stride in the convolutional layers.

In our experiments, we have intentionally chosen the architectures of f and $\tilde{f}$ to be different. Our aim is to be compliant with the defined threat model. However, we observed that choosing $\tilde{f}$ to be similar to f speeds up the attack procedure significantly.

Table A.2 reports additional hyper-parameters adopted during the attack.

Datasets preparation. All experiments reported in the paper have been carried out on RGB images with a resolution of 32×32. Grayscale images such as the ones in *MNIST*, *Fashion-MNIST*, *Omniglot* and *AT&T* are mapped into $32 \times 32 \times 3$ tensors by replicating the image three times alongside the channel dimension. For each dataset, color intensities are scaled in the real interval $[-1, 1]$.

A.1 Client-side attack

To implement the client-side attack, we rely on a DCGAN-like [38] architecture as in [25]. Specifically, the architecture for the splits f, s and f' as well as for the generator G are detailed in Table A.1. As in [25], we use a latent space of cardinality 100 with standard, Gaussian prior.

Table A.2: Other hyper-parameters used during the Feature-space hijacking attack.

Optimizer f	Adam with $lr = 0.00001$
Optimizer $\tilde{f}$ and $\tilde{f}^{-1}$	Adam with $lr = 0.00001$
Optimizer D	Adam with $lr = 0.0001$
Weight gradient penalty D	500.0

B EVADING THE DISTANCE CORRELATION METRIC VIA ADVERSARIAL FEATURE-SPACES

Despite the proven capability of the distance correlation metrics of capturing linear as well as non-linear dependence on high-dimensional data, this can be easily evaded by highly complex mappings like those defined by deep neural networks. More formally, given an input space X, it is quite simple to define a function f such that:

$$\mathbb{E}_{x \sim X}[DCOR(x, f(x))] = \epsilon_1 \text{ , but } \mathbb{E}_{x \sim X}[d(x, \tilde{f}^{-1}(f(x)))] = \epsilon_2, \tag{7}$$

where $\tilde{f}^{-1}$ is a decoder function, d is a distance function defined on X and ϵ_1 and ϵ_2 are two constant values close to 0. That is, the function $f(x)$ produces an output z that has minimal distance correlation with the input but that allows a decoder network $\tilde{f}^{-1}$ to accurately recover x from z. Intuitively, this is achieved by hiding information about x in z (smashed data) by allocating it in the blind spots of distance correlation metrics.

In practice, such function f can be learned by tuning a neural network to minimize the following loss function:

$$\mathcal{L}_{f,\tilde{f}^{-1}} = DCOR(x, f(x)) + \alpha_2 \cdot d(x, \tilde{f}^{-1}(f(x))) \tag{8}$$

that is, training the network to simultaneously produce outputs that minimize their distance correlation with the input and enable reconstruction of the input from the decoder $\tilde{f}^{-1}$. Next, we validate this idea empirically.

We report the result for *CelebA* and use f and $\tilde{f}^{-1}$ from the setup 4. We use *MSE* as d and $\alpha_2 = 50$. We train the model for 10^4 iterations. Figure B.1 reports the average distance correlation (Figure B.1a) and average reconstruction error (Figure B.1b) for the same model trained with three different losses; namely:

Table .1: Architectures used for running the Feature-space hijacking attack.

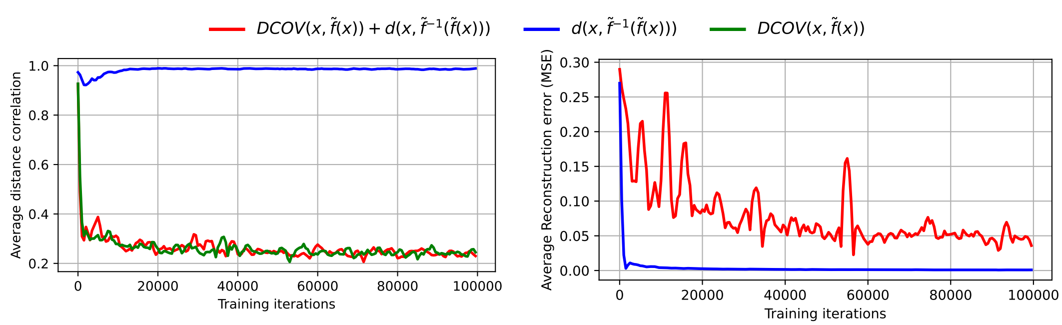
Split	f	$\tilde{f}$	$\tilde{f}^{-1}$	D

1	`2D-Conv(64, 3, (1,1), ReLU)`
	`batch-normalization`
	`ReLU`
	`maxPolling((2,2))`
	`resBlock(64, 1)`
		`2D-Conv(64, 3, (2,2), swish)`
		`2D-Conv(64, 3, (1,1), swish)`
			`2D-ConvTrans(256, 3, (2,2), ReLU)`
			`2D-Conv(3, 3, (1,1), tanh)`
				`2D-Conv(128, 3, (2,2), ReLU)`
				`2D-Conv(128, 3, (2,2))`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`2D-Conv(256, 3, (2,2), ReLU)`
				`dense(1)`

2	`2D-Conv(64, 3, (1,1), ReLU)`
	`batch-normalization`
	`ReLU`
	`maxPolling((2,2))`
	`resBlock(64, 1)`
	`resBlock(128, 2)`
		`2D-Conv(64, 3, (2,2), swish)`
		`2D-Conv(128, 3, (2,2), swish)`
		`2D-Conv(128, 3, (1,1)`
			`2D-ConvTrans(256, 3, (2,2), ReLU)`
			`2D-ConvTrans(128, 3, (2,2), ReLU)`
			`2D-Conv(3, 3, (1,1), tanh)`
				`2D-Conv(128, 3, (2,2))`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`2D-Conv(256, 3, (2,2), ReLU)`
				`dense(1)`

3	`2D-Conv(64, 3, (1,1), ReLU)`
	`batch-normalization`
	`ReLU`
	`maxPolling((2,2))`
	`resBlock(64, (1,1))`
	`resBlock(128, 2)`
	`resBlock(128, 1)`
		`2D-Conv(64, 3, (2,2), swish)`
		`2D-Conv(128, 3, (2,2), swish)`
		`2D-Conv(128, 3, (1,1)`
			`2D-ConvTrans(256, 3, (2,2), ReLU)`
			`2D-ConvTrans(128, 3, (2,2), ReLU)`
			`2D-Conv(3, 3, (1,1), tanh)`
				`2D-Conv(128, 3, (2,2))`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`2D-Conv(256, 3, (2,2), ReLU)`
				`dense(1)`

4	`2D-Conv(64, 3, (1,1), ReLU)`
	`batch-normalization`
	`ReLU`
	`maxPolling((2,2))`
	`resBlock(64, 1)`
	`resBlock(128, 2)`
	`resBlock(128, 1)`
	`resBlock(256, 2)`
		`2D-Conv(64, 3, (2,2), swish)`
		`2D-Conv(128, 3, (2,2), swish)`
		`2D-Conv(256, 3, (2,2), swish)`
		`2D-Conv(256, 3, (1,1))`
			`2D-ConvTrans(256, 3, (2,2), ReLU)`
			`2D-ConvTrans(128, 3, (2,2), ReLU)`
			`2D-ConvTrans(256, 3, (2,2), tanh)`
				`2D-Conv(128, 3, (1,1))`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`2D-Conv(256, 3, (2,2), ReLU)`
				`dense(1)`



(a) Distance correlation.

(b) Reconstruction error.

Figure B.1: The average distance correlation (panel (a)) and average reconstruction error (panel (b)) for the same model trained with three different losses on *CelebA*.

(1) In red, the model is trained on the adversarial loss reported in Eq. 8.
(2) In green, the model is trained only to minimize distance correlation.
(3) In blue, the model is trained only to minimize the reconstruction error (i.e., auto-encoder).

As can be noticed, the adversarial training procedure permits to learn a pair of f and $\tilde{f}^{-1}$ such that the distance correlation is minimized (the same as we train the model only to minimize distance correlation), whereas it enables the reconstruction of the input data.

Table A.1: Architectures for the client-side attacks.

f
2D-Conv(64, 5, (2,2)) LeakyReLU dropout(p=0.3)
s
2D-Conv(126, 5, (2,2) LeakyReLU dropout(p=0.3)
f'
dense(#classes) sigmoid
G
dense(7·7·256) batch-normalization LeakyReLU 2D-ConvTrans(128, 5, (1,1)) batch-normalization 2D-ConvTrans(128, 5, (1,1)) batch-normalization LeakyReLU 2D-ConvTrans(64, 5, (2,2)) batch-normalization LeakyReLU 2D-ConvTrans(1, 5, (2,2), tanh)