Passive Inference Attacks on Split Learning via Adversarial Regularization

Xiaochen Zhu*†, Xinjian Luo*‡, Yuncheng Wu§◇1, Yangfan Jiang*, Xiaokui Xiao*, and Beng Chin Ooi*
*National University of Singapore, †Massachusetts Institute of Technology,
‡Mohamed bin Zayed University of Artificial Intelligence, §Renmin University of China
xczhu@mit.edu, xinjian.luo@mbzuai.ac.ae, wuyuncheng@ruc.edu.cn,
yangfan.jiang@comp.nus.edu.sg, xkxiao@nus.edu.sg, ooibc@comp.nus.edu.sg

***Abstract*—Split Learning (SL) has emerged as a practical and efficient alternative to traditional federated learning. While previous attempts to attack SL have often relied on overly strong assumptions or targeted easily exploitable models, we seek to develop more capable attacks. We introduce SDAR, a novel attack framework against SL with an honest-but-curious server. SDAR leverages auxiliary data and adversarial regularization to learn a decodable simulator of the client's private model, which can effectively infer the client's private features under the vanilla SL, and both features and labels under the U-shaped SL. We perform extensive experiments in both configurations to validate the effectiveness of our proposed attacks. Notably, in challenging scenarios where existing passive attacks struggle to reconstruct the client's private data effectively, SDAR consistently achieves significantly superior attack performance, even comparable to active attacks. On CIFAR-10, at the deep split level of 7, SDAR achieves private feature reconstruction with less than 0.025 mean squared error in both the vanilla and the U-shaped SL, and attains a label inference accuracy of over 98% in the U-shaped setting, while existing attacks fail to produce non-trivial results.**

I. Introduction

To bridge the isolated data repositories across different data owners, federated learning (FL) [3], [26], [32], [55], [56] has been proposed as a solution to privacy-preserving collaborative learning. However, participants engaged in FL often suffer from low communication efficiency and heavy computational overhead. This is often imposed by the iterative process of local model training and the frequent exchange of model parameters, especially for deep neural network (NN) models. Naturally, as a simple adaption with enhanced communication and computation efficiency of FL, split learning (SL) [19], [39], [43], [48], [50], [51], [60] has drawn increasing attention in various applications, such as healthcare [20], [39], [50] and open source packages [21], [37]. The general idea of SL is to *split* an NN model into smaller partial models, with simpler models being allocated to clients, and more intricate ones hosted on a computationally capable server. During training, clients first send the intermediate representations produced by their partial models to the server. Subsequently, the server performs a forward pass on its own partial model and back-propagates the gradients back to the clients for model updates. In this way, SL enables privacy-preserving collaborative learning by sharing only the intermediate representations without revealing the original private data from the clients. Compared to FL, SL adopts a more computationally efficient approach, yielding improved communication efficiency and scalability without compromising model utility [14], [43], [60]. Nonetheless, privacy concerns still exist within the SL framework. Given that clients share model intermediate representations with the server, one may naturally wonder *if it is possible for the server to infer private data of clients from the shared intermediate representations.*

Related work. To address this question, several inference attacks [9], [13], [23], [38], [40], [61] have been devised to investigate the privacy risks of SL. However, *these attacks typically consider overly strong threat models and target non-standard settings favorable for the adversary.* For example, Pasquini et al. [38] assume a malicious server that *actively* tampers the back-propagated gradients. This deviates from the original training protocol of SL and can be easily detected as an agreement violation [8], [12]. In addition, Erdoğan et al. [9], Qiu et al. [40] and Gao and Zhang [13] propose passive inference attacks, in which the server honestly follows the SL training protocol but attempts to infer clients' private data by analyzing the shared intermediate outputs. Although these attacks are more stealthy, they often target non-standard and easily exploitable models, as did in [38]. Specifically, the targeted models on the client side are typically non-standard models that are overly wide for their input dimensions. This results in higher dimensional intermediate representations that encode more information about the private input, thus making the attacks more effective. Additionally, existing attacks are often evaluated on split configurations where the clients' models are shallow, which are commonly believed to be more vulnerable to attacks [13], [38], [52]. Correspondingly, as we later show in the paper, these attacks can be mitigated by considering models with adequate width and allocating more layers to the clients' partial models. Furthermore, the visual features of reconstructed images produced by existing passive attacks are hardly comparable to that of the active attack proposed by Pasquini et al. [38]. Therefore, it is still underexplored whether passive inference attacks can achieve reasonable effectiveness comparable to their active counterparts with less exploitable model structures.

In this paper, we address this gap by developing passive

◇This work was done at National University of Singapore.
1Corresponding author.

Network and Distributed System Security (NDSS) Symposium 2025
23 February - 28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2025.23030
www.ndss-symposium.org

feature and label inference attacks on more challenging SL settings where existing attacks [9], [13], [38] fail to work, i.e., we target standard models with adequate width at deeper split levels. By considering passive attacks, we can ensure the stealthiness of our attacks, making them hardly susceptible to detection by active defense mechanisms [8], [12]. By considering the more challenging settings, we can reveal the privacy vulnerabilities of SL that may exist in real-world applications but have not been explored. Nonetheless, before devising the proposed attacks, several challenges must be addressed.

Challenges. First, the main challenge in server-side attacks on SL is the absence of access (including the black-box access) to the client's partial model. Namely, the server cannot feed specific input to the client's model and observe the corresponding output. Such lack of access renders the traditional approach of model inversion [10], [11], [49], [58], [63] inapplicable. Second, in a passive setting where the server cannot manipulate the client's model, the only information available to the server is the shared intermediate representations during training. As the complexity of the client's partial model increases, e.g., with deeper split levels, the information encoded in these representations diminishes. This would make it more challenging to decode the private data solely from the received representations. Third, during SL training, the client's model undergoes continuous updates in each iteration. This dynamic nature results in entirely different representations over iterations, further complicating inference attacks, compared to attacks on finalized models [23], [61].

Aside from the typical SL setting, we also consider a more intricate SL configuration, known as U-shaped SL. In this setting, the last few layers of the model are also kept privately by the clients, and the server's responsibility is limited to training the intermediate layers of a neural network. This introduces two new challenges. First, the absence of access to training examples' labels poses the need for the server to simultaneously recover private features and labels to achieve a reasonable attack. Second, established practices and theoretical insights of transfer learning indicate that the last few layers of a network are crucial to learn domain-specific representations [62]. The lack of these layers makes it harder for the server to recover private training samples.

Contributions. To address these challenges, we propose a novel class of passive attacks on SL, namely, Simulator Decoding with Adversarial Regularization (SDAR). The foundation of SDAR is to let the server train a simulator that emulates the client's private model while simultaneously training a decoder tailored to the trained simulator, on auxiliary data disjoint from the private target data. Ideally, a well-trained decoder should also be capable of decoding the client's private model. However, due to the intrinsic differences in distribution between the client's private data and the server's auxiliary data, the simulator trained by the server tends to overfit the auxiliary data. Consequently, the simulator cannot faithfully replicate the behaviors of the client's model. This renders the decoder incapable of reconstructing the client's data albeit proficient at reconstructing the auxiliary data.

To solve this issue, we draw inspiration from generative adversarial networks (GANs) [16], [17] and innovatively propose to regularize the simulator and decoder through an adversarial loss, encouraging them to learn more generalized representations that are transferable to the client's private data. For vanilla SL where the server has access to the labels of training records, SDAR utilizes the label information in the style of conditional GAN [33]. In the more challenging U-shaped SL setting, where the client privately retains the last few layers of the model along with the record labels, SDAR trains an additional simulator to mimic the behavior of these last few layers. To mitigate overfitting of this supplementary simulator, we introduce label random flipping, which encourages this simulator to learn the general representations rather than mere memorization of the server's data. Furthermore, we utilize the predictions of this simulator on private samples to achieve label inference attacks in the context of the U-shaped setting.

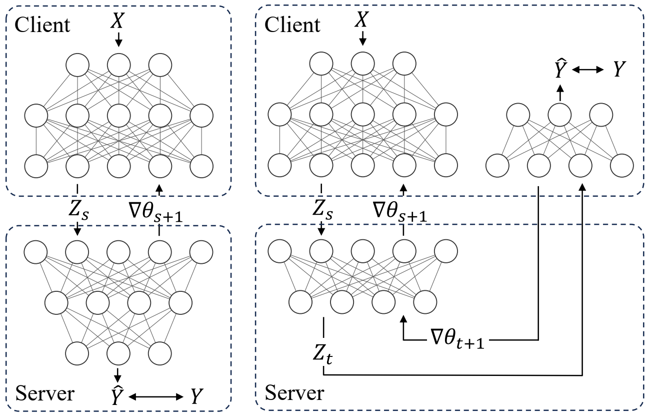
We conduct extensive experiments on various real-world datasets and widely-used model architectures in both SL configurations to demonstrate the effectiveness of the proposed attacks. The results show that under challenging settings where existing passive attacks fail to effectively reconstruct clients' private data, SDAR achieves consistent and distinctive attack performance, and is even able to almost match the performance of active hijacking attacks. This is the first time that passive attacks have been shown to be comparably effective as their active counterparts. Also, SDAR remains effective when the server has limited access to auxiliary dataset or possesses no knowledge of the client's model architecture. Additionally, our results reveal that inference attacks become more challenging with the increase of split level or the decrease of the client's model width. While the former effect has been investigated in the literature [9], [13], [38], [40], we are the first to demonstrate that wider models, due to their higher dimensional intermediate representations, are also more vulnerable to inference attacks. This is a critical insight for practitioners to consider when deploying SL in real-world applications. Finally, we evaluate potential countermeasures and show the robustness of our attacks against such defenses.

II. Preliminaries

Split learning. We consider training an NN model H on dataset $D = \{(x_t, y_t) : t = 1, \ldots, N\}$. H consists of n layers $H = L_n \circ L_{n-1} \ldots \circ L_1$. The key idea of SL [19], [39], [50] is to *split* the execution of H by a *split layer* (namely, L_s) and assign the first half $f = L_s \circ \ldots \circ L_1$ to the client and the second half $g = L_n \circ \ldots \circ L_{s+1}$ to the server. Then, $H = g \circ f$. In the vanilla SL setting, the server also holds the labels of training examples. For a batch of examples X, we define the representations returned by layer L_i as Z_i. Then, on the forward pass, the client sends intermediate representations $Z_s = f(X)$ (also known as smashed data) to the server such that the latter can complete the forward pass via $Z_n = g(Z_s)$. Since the server has access to the record labels Y, it can evaluate the loss $\ell(Z_n, Y)$ where $\ell(\cdot, Y)$ is the loss function given ground truth labels Y. In backpropogation, for parameters θ_i of L_i, chain rule gives

$$\begin{aligned}\nabla\theta_i &= \frac{\partial \ell(Z_n, Y)}{\partial Z_n}\frac{\partial L_n(Z_{n-1})}{\partial \theta_n}\cdots\frac{\partial L_i(Z_{i-1})}{\partial \theta_i} \\ &= \nabla\theta_{i+1}\frac{\partial L_i(Z_{i-1})}{\partial \theta_i}.\end{aligned} \tag{1}$$

By (1), one only needs the gradients of the next layer and the representations returned by the previous layer to differentiate



(a) Vanilla SL (b) U-shaped SL

Fig. 1. Vanilla and U-shaped configurations of SL.

layer L_i. Thus, the server can update parameters θ_g of its model g after receiving Z_s from the client, while the client can update f after receiving $\nabla\theta_{s+1}$ from the server (see Fig. 1a for an example).

U-shaped SL. If the client's record labels are considered private and not shared with the server, SL can be configured as a U-shaped structure [19], as illustrated in Fig. 1b. Under U-shaped SL, the NN model is partitioned into three parts, i.e., $H = h \circ g \circ f$, where $f = L_s \circ \ldots \circ L_1$, $g = L_t \circ \ldots \circ L_{s+1}$, and $h = L_n \circ \ldots \circ L_{t+1}$. The client owns f and h while the server only hosts g. During the training on samples X, the client sends Z_s to the server, which computes Z_t and sends it back to the client. The client then computes the final prediction $Z_n = h(Z_t)$. In the backpropogation phase, the client first updates its model h and sends $\nabla\theta_{t+1}$ to the server. Then, the server updates g and sends $\nabla\theta_{s+1}$ to the client for updating the partial model f on the client side.

SL with multiple clients. Due to its simplicity, the SL framework can be easily scaled to support multiple clients on either horizontally-partitioned data [50] or vertically-partitioned data [4]. Note that in the multi-client SL scenarios, each client still needs to communicate with the server via the above-described protocol, i.e., sharing intermediate representations with the server.

III. Problem Statement

System model. For ease of presentation, we introduce the proposed attacks in the SL setting where a client and a server collaboratively train a deep model H in either the vanilla or the U-shaped configuration. Our attacks can be seamlessly extended to the multi-client SL scenario for inferring the private data from different clients without further modifications, given that the shared information between clients and the server in the multi-client scenario is the same as that in the single-client scenario. Note that image datasets are typically used to train deep models under various SL settings [19].

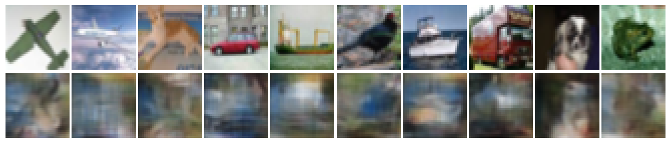
Threat model. We investigate the privacy risks of SL under the honest-but-curious threat model [31]. Namely, we consider a server that honestly adheres to the SL protocol and does not tamper with the training process. Meanwhile, the server tries to infer the client's private information from the received messages. In addition, we assume that the server has background knowledge of an auxiliary public dataset $D' = \{(x'_t, y'_t) : t = 1, \ldots, m\}$ such that $D \cap D' = \emptyset$, where D denotes the client's private dataset and shares a similar distribution with D'. This is a common and reasonable assumption in related studies [13], [38] because the task types and data domain should be negotiated between the client and the server before initiating an SL training. This is a necessary step for the participating parties to determine the appropriate network structure and split level before training, for eliminating possible overfitting or underfitting issues [13], [38]. Consequently, the server can collect a public dataset from the same data domain for attack implementation [13], [38]. Note that we assume no access to any example in the client's private dataset D for the server, which is more stringent than [13] where D' can be a subset of D. In addition, the server and the client should agree on the model architecture of H and the split configuration beforehand for information exchange and model convergence [9]. That is, we start by assuming that the server knows the architecture of f. We later relax this, assuming that the server does not have any knowledge of the architecture of f, but only the input and output dimensions of the model. During model training, the server has neither black-box nor white-box access to the client's models, and can only receive intermediate representations from the client based on the SL protocol.

In *vanilla SL*, the client hosts model f while the server hosts model g and has access to the record labels. For a batch of training records (X, Y), the server aims to reconstruct the private features X given the received intermediate representations Z_s, i.e., $\hat{X} = \mathcal{A}(Y, Z_s, \theta_g, D')$, where $\hat{X}$ is the inferred features, θ_g represents the parameters of g, and $\mathcal{A}$ denotes the attack algorithm. In *U-shaped SL*, the client hosts partial models f and h while the server hosts model g. For a batch of training records (X, Y), the server aims to reconstruct both the private features X and the labels Y given the received Z_s and the gradient vector $\nabla\theta_{t+1}$, i.e., $\hat{X}, \hat{Y} = \mathcal{A}(Z_s, \theta_g, \nabla\theta_{t+1}, D')$.

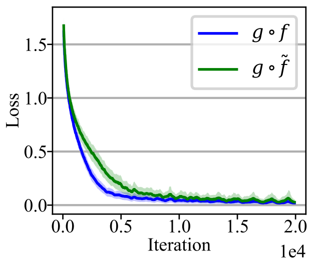
In Section VII, we discuss the threat models of existing attacks in greater details and compare them with ours. We also provide a taxonomy of inference attacks on SL in Table XII.

IV. Passive attacks against SL

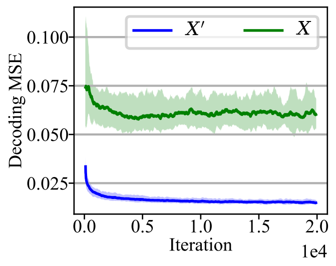
In this section, we introduce SDAR, a novel framework for an honest-but-curious server to infer the client's private data in SL. The core idea of SDAR is to train a simulator model that learns similar representations as the client model in a way that their outputs are indistinguishable from each other. This is achieved by introducing an adversarial discriminator that regularizes the training of the simulator model. After that, a corresponding decoder is trained on D' for decoding the intermediate representations output by the simulator. By regularizing this decoder via an additional adversarial discriminator, this decoder can be generalized to effectively decode the client's intermediate representations despite $D \cap D' = \emptyset$. SDAR is capable of inferring private features in the vanilla SL and both private features and labels in the U-shaped SL.



(a) Private X (upper) vs $\hat{X}$ (lower) reconstructed by the naïve SDA



(b) Training loss of $g \circ f$ vs $g \circ \tilde{f}$



(c) Decoding MSE on X' vs X

Fig. 2. Failure of naïve SDA on CIFAR-10 with ResNet-20 at split level 7. Solid lines are mean values over 5 runs, and values between min/max boundaries are shaded. All later figures follow the same convention.

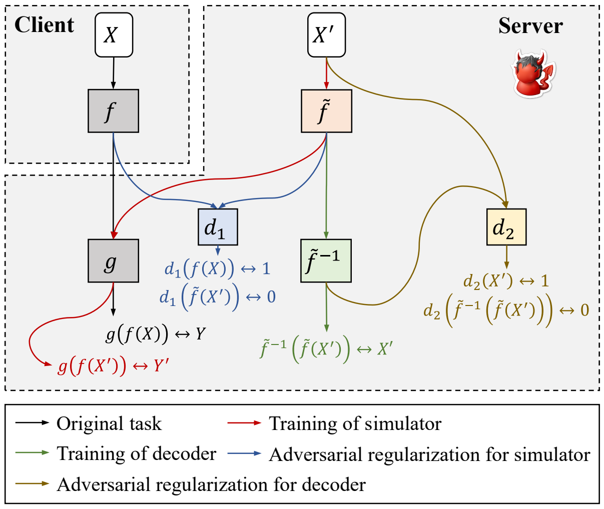

Client
Server
X
X'
f
$\tilde{f}$
g
d_1
$\tilde{f}^{-1}$
d_2
$d_1(f(X)) \leftrightarrow 1$
$d_1(\tilde{f}(X')) \leftrightarrow 0$
$d_2(X') \leftrightarrow 1$
$d_2(\tilde{f}^{-1}(\tilde{f}(X'))) \leftrightarrow 0$
$g(f(X)) \leftrightarrow Y$
$g(f(X')) \leftrightarrow Y'$
$\tilde{f}^{-1}(\tilde{f}(X')) \leftrightarrow X'$
Original task
Training of simulator
Training of decoder
Adversarial regularization for simulator
Adversarial regularization for decoder

Fig. 3. Overview of SDAR in the vanilla SL setting.

A. Feature inference attacks in vanilla SL

We start with a simple attack prototype, denoted as naïve simulator decoding attack (naïve SDA). As the server has access to an auxiliary dataset D', it can first initialize a simulator $\tilde{f}$ on its own and then train the model $g \circ \tilde{f}$ on D'. Depending on particular settings, this simulator $\tilde{f}$ may or may not share the same architecture as f. For now, we assume they share the same architecture but $\tilde{f}$ is initialized independently and randomly as the server has no access to the weights of f. Specifically, for each batch of training examples (X, Y), after the parameters of f, g are updated via the training loss $\ell(g(f(X)), Y)$, the server samples another batch of examples (X', Y') from D' and trains $\tilde{f}$ to minimize loss $\ell(g(\tilde{f}(X')), Y')$ with the model g frozen. The reason to fix g in this step is mainly twofold. First, as an honest-but-curious party, the server should ensure that the parameters of g are only updated based on D as specified in the SL protocol. Second, fixing g while training $\tilde{f}$ forces $\tilde{f}$ to learn representations that are compatible with g. As g is trained collaboratively with the client's model f on D, it is expected to memorize information of D [42] which may be implicitly leaked to $\tilde{f}$ while training $g \circ \tilde{f}$. In this way, the encoder $\tilde{f}$ trained by the server is expected to mimic the behaviors of f. In the meantime, the server trains a decoder $\tilde{f}^{-1}$ with the transposed architecture of $\tilde{f}$ to decode $Z'_s = \tilde{f}(X')$, i.e., the decoder parameters $\theta_{\tilde{f}^{-1}}$ are updated via the loss $\ell_{\text{MSE}}(X', \tilde{f}^{-1}(Z'_s))$. One may expect the decoder $\tilde{f}^{-1}$ can not only decode the output of the simulator $\tilde{f}$ but also effectively decode the output Z_s of f, i.e., $\tilde{f}^{-1}(Z_s) \approx X$ for private features X.

Unfortunately, the naïve SDA fails to achieve this objective as illustrated in Fig. 2a. The main reason behind such failure is that although the training loss of $g \circ \tilde{f}$ on X' converges to a similar minimum as that of $g \circ f$ on X (see Fig. 2b), $\tilde{f}$ fails to learn the same representations as f. This generalization gap is rooted in the distributional discrepancy between the disjoint datasets D and D', and the fact that $\tilde{f}$ is trained solely on D' makes it prone to overfitting to D'. As a result, even if the decoder $\tilde{f}^{-1}$ can precisely decode the simulator $\tilde{f}$ with very low reconstruction error, its reconstruction error on Z_s, the output of the client's private model f on unseen features X, does not converge (see Fig. 2c).

In response, we propose to enhance naïve SDA with adversarial regularization, to encourage the simulator and the decoder to learn more general representations transferable to the client's private data X.

Adversarial regularization for $\tilde{f}$. To regularize the simulator $\tilde{f}$ such that it behaves more similarly to the client's private model f, we introduce a server-side discriminator d_1 that is trained to distinguish $Z'_s = \tilde{f}(X')$ (the fake data) against $Z_s = f(X)$ (the real data). In each iteration, d_1 is updated to minimize the loss

$$\mathcal{L}_{d_1} = \ell_{\text{BCE}}(d_1(Z'_s), 0) + \ell_{\text{BCE}}(d_1(Z_s), 1) \tag{2}$$

to correctly classify Z'_s against Z_s, where ℓ_{BCE} is the binary cross-entropy loss. In the meantime, the simulator is updated in an adversarial way that maximizes the likelihood of being misclassified by the discriminator, i.e., to minimize the loss $\ell_{\text{BCE}}(d_1(\tilde{f}(X')), 1)$. We introduce this adversarial loss as a regularization term with penalty parameter λ_1, i.e., the simulator $\tilde{f}$ is updated to minimize the loss

$$\mathcal{L}_{\tilde{f}} = \ell(g(\tilde{f}(X')), Y') + \lambda_1 \cdot \ell_{\text{BCE}}(d_1(\tilde{f}(X')), 1). \tag{3}$$

Thus, with the training loss on D' as the main objective and the adversarial loss as the regularization term, the simulator is expected to output representations indistinguishable from the output of f, thus better simulating f's behaviors on the private data X.

Adversarial regularization for $\tilde{f}^{-1}$. As shown in Fig. 2c and 2a, the decoder $\tilde{f}^{-1}$ can accurately decode the output of $\tilde{f}$ since $\tilde{f}^{-1}$ is trained in a supervised manner on Z'_s. However, this reconstruction capability can not be generalized to decode Z_s. Specifically, we observe from Fig 2a that the reconstructed samples $\hat{X}$ exhibit obfuscated visual features and can be easily distinguished from real images by a well-trained discriminator. Motivated by this observation, we introduce another discriminator d_2 to distinguish $\tilde{f}^{-1}(Z_s)$ (the fake data) against X' (the real data) owned by the server. Similar to Eq. (2), d_2 is trained by minimizing

$$\mathcal{L}_{d_2} = \ell_{\text{BCE}}(d_2(\tilde{f}^{-1}(Z_s)), 0) + \ell_{\text{BCE}}(d_2(X'), 1). \quad (4)$$

Meanwhile, the decoder $\tilde{f}^{-1}$ needs to be trained in an adversarial manner to maximize the likelihood of being misclassified by the discriminator. Similar to Eq. (3), we introduce a regularization term with penalty parameter λ_2, and the decoder is trained to minimize

$$\mathcal{L}_{\tilde{f}^{-1}} = \ell_{\text{MSE}}(X', \tilde{f}^{-1}(Z'_s)) + \lambda_2 \cdot \ell_{\text{BCE}}(d_2(\tilde{f}^{-1}(Z_s)), 1). \quad (5)$$

In this way, the decoder is trained to reconstruct private images from Z_s with plausible visual features as real images.

Use of labels. In the vanilla SL, the server holds labels of all private training images, indicating that the server has access to the labels of both the private data X and the auxiliary data X'. This enables the server to adopt the decoder $\tilde{f}^{-1}$ and the discriminators d_1, d_2 in a conditional manner similar to the conditional GANs [33]. Take the decoder $\tilde{f}^{-1}$ as an example. Instead of only taking the intermediate representations Z_s, Z'_s as the input, a conditional $\tilde{f}^{-1}$ can additionally take the corresponding labels Y, Y' as the input. These labels can be first transformed into a high-dimensional embedding and then concatenated with the intermediate representations. A similar modification can be applied to the discriminators. This adaptation enables the decoder to more effectively decode the intermediate representations, and enhances the discriminators' capability to distinguish between real and synthetic examples.

With the integration of the aforementioned enhancements, we denote the refined simulator decoding attack as Simulator Decoding with Adversarial Regularization (SDAR). We describe the complete attack in Fig. 3 and provide pseudocode of our attack in Fig. 19 of Appendix A.

B. Feature & label inference attacks in U-shaped SL

In the U-shaped SL, the model H is split into three parts $H = h \circ g \circ f$ such that only the intermediate partial model g is kept by the server, and the other two are trained by the client. An intuitive approach to adapt SDAR to the U-shaped SL is that the server trains an additional simulator $\tilde{h}$ to mimic the behaviors of h, which can be updated by minimizing the training loss of $\tilde{h} \circ g \circ \tilde{f}$ on D' when freezing g. If the simulator $\tilde{h}$ is well-trained on D' such that it is able to generalize well to classify unseen private examples in D, the server can effectively reconstruct private labels $\hat{Y} = \tilde{h}(g(Z_s))$ in addition to private features. However, the last few layers (i.e., h) of the model has a strong expressive capacity of the private information suggested by the practice and theory of transfer learning [62]. Consequently, $\tilde{h}$ can easily overfit to D' such that the training loss on D' converges to a local minimum yet $\tilde{h}$ and $\tilde{f}$ fail to learn the similar behaviors of h and f. In this case, the decoder $\tilde{f}^{-1}$ will not be able to reconstruct the private input X from Z_s effectively when trained along with the overfitted $\tilde{f}$. Thus, it is crucial to ensure that the simulator $\tilde{h}$ learns the general and transferable data representations, instead of overfitting to D'.

To this end, we introduce **random label flipping** in the training of $\tilde{h} \circ g \circ \tilde{f}$. Specifically, for an incoming batch of auxiliary examples (X', Y') sampled from D', we first independently flip each label Y'_i to a random label Y'_i with a probability of p. Namely, for each $Y'_i \in Y'$, we have

$$Y'_i = \begin{cases} Y'_i & \text{w.p. } 1-p \\ \text{Uniform}(\mathcal{Y}) & \text{w.p. } p \end{cases},$$

where $\mathcal{Y}$ is the set of all possible labels. This way, the server generates a new noisy label set $\tilde{Y}'$ and trains $\tilde{h} \circ g \circ \tilde{f}$ based on the new training batch $(X', \tilde{Y}')$. These randomly flipped labels can help regularize the training of $\tilde{h}$, encourage $\tilde{h}$ not to overly fit the auxiliary data, but to produce more general representations that are compatible with the output of g and meanwhile transferable to the private data D. Other components of SDAR except for the use of labels in Section IV-A are directly applicable to the U-shaped SL setting, including the adversarial regularization on $\tilde{f}$ and $\tilde{f}^{-1}$. We describe the complete attack in Fig. 20 of Appendix A.

C. Computational complexity

Let b be the batch size, i.e., the number of examples used for SL training in each iteration. For each iteration, if there's no attack, the SL protocol requires the client and the server to update the model $g \circ f$ via a forward pass and backpropagation. Let F_f and F_g denote the number of floating-point operations (FLOPs) required by a single forward pass on models f and g, respectively. The total computation cost of the SL protocol is $\mathcal{O}(b \cdot (F_f + F_g))$ FLOPs, as the forward pass costs $b(F_f + F_g)$ FLOPs and the backpropagation costs approximately twice as many FLOPs [15]. If the server is to reconstruct the private b training images via naïve SDA, it will then also train $g \circ \tilde{f}$ and $\tilde{f}^{-1}$ using a batch of b auxiliary images, before decoding Z_s via $\tilde{f}^{-1}$. This introduces additional $\mathcal{O}(b(F_g + F_{\tilde{f}} + F_{\tilde{f}^{-1}}))$ FLOPs. Hence, the SL training and server's inference attack via naïve SDA require $\mathcal{O}(b(F_f + F_g + F_{\tilde{f}} + F_{\tilde{f}^{-1}}))$ FLOPs per iteration. Furthermore, if the server adopts full SDAR and introduce discriminators d_1, d_2, additional cost of $\mathcal{O}(bF_{d_1} + bF_{d_2})$ FLOPs will be introduced. Therefore, in total, SL training and SDAR will require $\mathcal{O}(b(F_f + F_g + F_{\tilde{f}} + F_{\tilde{f}^{-1}} + F_{d_1} + F_{d_2}))$ FLOPs of computation, which is asymptotically as efficient as SL training itself if model complexity is considered as a constant. We later compare the computational complexity of SDAR with other state-of-the-art attacks in Section V.

D. Technical novelty

First, SDAR identifies an underexplored vulnerability of SL, i.e., the risks of information leakage from the parameters of the server's model g. Since g is trained together with f on the private examples (X, Y), it unintentionally memorizes information about the private data [42], and a carefully designed simulator trained with g and (X', Y') can accurately simulate the behaviors of f. On the contrary, most existing attacks [9], [38] focus on only exploiting the privacy risks of the client's intermediate representations Z_s, which is often insufficient to reconstruct private data under practical settings.

Second, SDAR utilizes a novel adversarial regularization method to considerably improve the generalization performance of the simulator and decoder on the client's private data, which can achieve consistent and robust attack performance on less vulnerable models where existing attacks [9], [13], [38] cannot work. To the best of our knowledge, this is the first time that adversarial regularization is utilized in the design of inference attacks on FL or SL.

One parallel study, PCAT [13], adopts a similar attack framework coinciding with our naïve SDA discussed in Section IV-A, with some minor improvements. However, as demonstrated in Fig. 2 and our comparative experiments in Section V, PCAT fails to reconstruct reasonable images in challenging settings. Compared to naïve SDA or PCAT, SDAR's main advantage is our novel adversarial regularization framework, boosting the attack performance by improving the simulator and decoder's generalization.

V. Experiments

A. Experimental settings

Datasets. We experiment with four popular benchmarking datasets: CIFAR-10 [28], CIFAR-100 [28], Tiny ImageNet [45], and STL-10 [5]. CIFAR-10 and CIFAR-100 consist of 60,000 $32 \times 32 \times 3$ colored images in 10 and 100 classes, respectively. To demonstrate the scalability of our attacks to larger images, we experiment with image datasets with higher resolutions: Tiny ImageNet is a subset of the ImageNet dataset [6] consisting of 120,000 colored images of size $64 \times 64 \times 3$ of 200 classes, while STL-10 is another subset of the ImageNet dataset with 13,000 colored images of size $96 \times 96 \times 3$ in 10 classes. Due to limited space, we report results on CIFAR-10 and CIFAR-100 in the main text, and defer full results to the full version of this paper [64]. We normalize all images to $[0, 1]$ beforehand to match the input configuration of the convolutional models and partition each dataset into two disjoint subsets D, D' where D belongs to the client, and D' belongs to the server. We start with $|D| = |D'|$ where the server acquires auxiliary data of the same size as the client's private dataset. Then, we experiment with $|D'| \ll |D|$ to discuss the effectiveness of our attacks with much less data.

Models. We experiment SDAR and other baseline attacks on ResNet-20 and PlainNet-20 [22] as model H. Both models have standard architectures specifically designed to classify $32 \times 32 \times 3$ datasets. We chose these models because they are widely adopted in real-world applications, and are recognized to be hard to invert [2]. For both architectures, the complete model H consists of 20 layers (19 convolutional layers of 16/32/64 filters and one additional fully-connected output layer) with batch normalization. Particularly, ResNet-20 is equipped with 9 residual blocks, while PlainNet-20 is a VGG-style CNN model with 9 convolutional building blocks without residual connections. We experiment with different split levels denoted by $s \in \{4, 5, 6, 7\}$ on both models, characterized by the number of building blocks in the client's model f. The deepest client's model that we consider has 7 building blocks as adding any more blocks will render the client to host more parameters than the server, invalidating SL's purpose of enabling a powerful server to relieve the client from heavy computation. In the vanilla SL, all the remaining layers are assigned to the server as shown in Table I. In the U-shaped SL, the last few layers, namely, the average pooling and the fully connected output layer, are also assigned to the client, while the server only hosts the layers in between.

TABLE I. Model Statistics on Various Split Levels of 4–7 in Vanilla Split Learning

Level	Client's f		Server's g	
	No. of layers	No. of parameters	No. of layers	No. of parameters
4	9	29,424	11	244,618
5	11	48,112	9	225,930
6	13	66,800	7	207,242
7	15	124,912	5	149,130

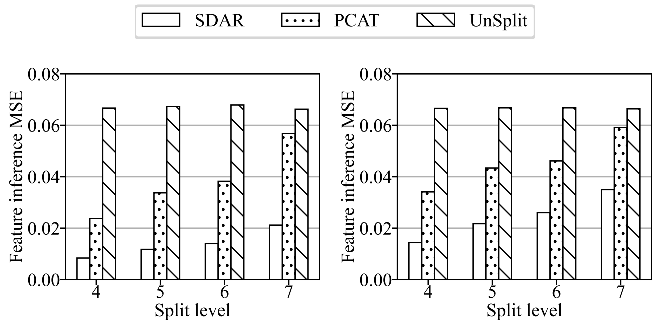
It is worth noting that we experiment with ResNet-20 and PlainNet-20 on larger images in Tiny ImageNet and STL-10, instead of wider models with more filters, because the former is more challenging for inference attacks due to the lower dimensionality of the intermediate representations. By considering the standard convolutional models with 16/32/64 filters and deep split levels up to 7, we target more challenging settings where the client's private model is less exploitable to existing attacks. Our experiments later demonstrate that the wider and shallower models considered by existing attacks [9], [13], [38] are more vulnerable to inference attacks.

For the simulator $\tilde{f}$ (and $\tilde{h}$ if in U-shaped SL) of SDAR, we start with the case where the server uses the same model architecture as that of the client's f (and h if in U-shaped SL). We later show that our attacks are still effective even without such information where the simulator has a different architecture. The server uses the decoder $\tilde{f}^{-1}$ with a structure transposed to $\tilde{f}$ ending with a sigmoid function to produce tensors within $[0, 1]$. The discriminators d_1 that distinguishes Z'_s from Z_s, and d_2 that distinguishes $\hat{X}$ from X', are deep convolutional networks. Architecture details of the attack models are described in Appendix B-C.

Baselines. We compare SDAR with existing attacks, including UnSplit [9], PCAT [13] and FSHA [38]. UnSplit [9] is a passive attack which reconstructs private features by minimizing $\ell_{\text{MSE}}(\tilde{f}(\hat{X}), Z_s)$ via alternating optimization on $\tilde{f}$ and $\hat{X}$. PCAT [13] is also a passive attack that adopts a similar attack framework as our naïve SDA discussed in Section IV-A. FSHA [38], on the other hand, is an active attack framework, where the server actively hijacks the gradients transmitted to the client such that the client's model f is induced to behave similarly to the server's own encoder. To ensure the fairness of comparison, we use the same model architectures and auxiliary datasets for all attackers unless specified otherwise. Implementation details of the baselines are described in Appendix B-D.

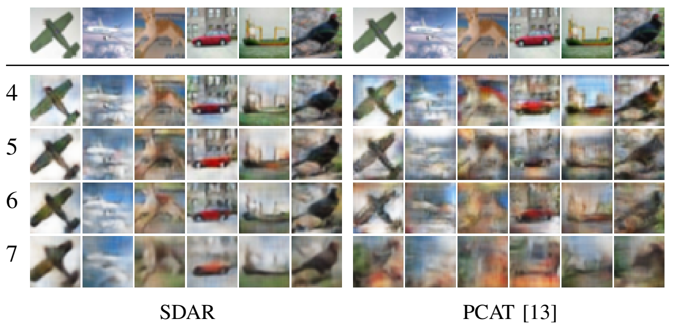
B. Experiments on vanilla SL attacks

Attack effectiveness. We report the results of SDAR and other passive feature inference attacks on CIFAR-10 with ResNet-20 and PlainNet-20 at split levels 4–7 in Fig. 4, and give concrete examples of private data reconstruction in Fig. 5. Fig. 4 shows that both SDAR and PCAT perform better at shallower split levels, and the performance degrades as the split level increases. This is expected as the intermediate representations Z_s become more abstract and less informative about the private features X at deeper layers, hence rendering it harder to reconstruct X from Z_s. We also notice that attacks are slightly more effective with the ResNet-20 than PlainNet-20, which is likely due to the former's residual connections

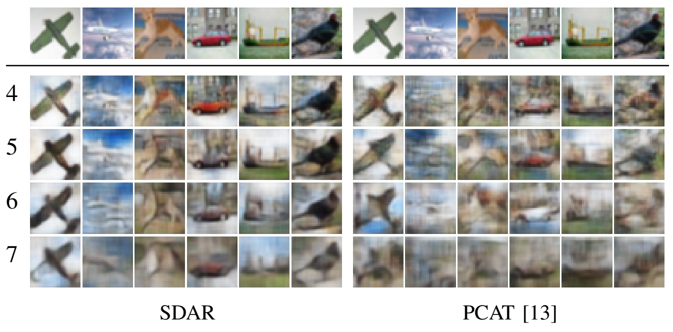


(a) ResNet-20 (b) PlainNet-20

Fig. 4. Mean feature inference MSE on CIFAR-10 with ResNet-20 and PlainNet-20 at the split levels of 4–7 on CIFAR-10 in vanilla SL.

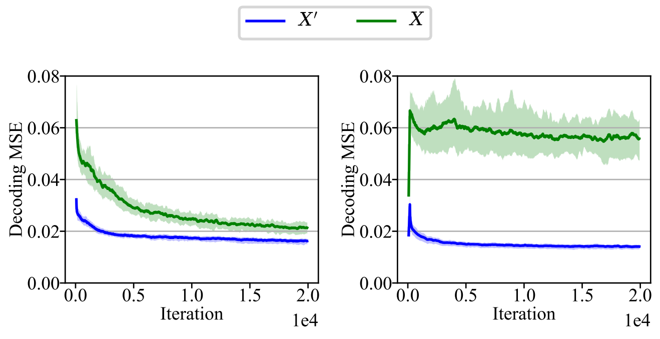


(a) ResNet-20



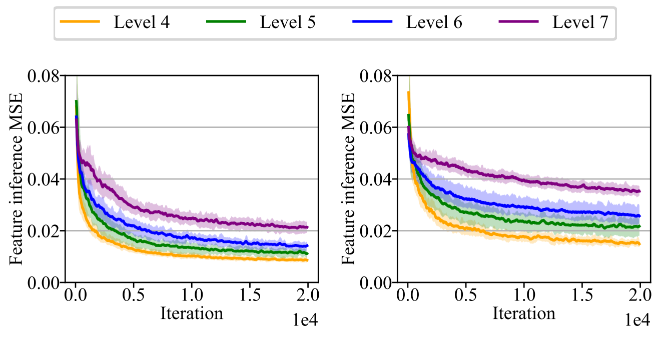
(b) PlainNet-20

Fig. 5. Examples of feature inference attack results with ResNet-20 and PlainNet-20 on CIFAR-10 in vanilla SL. The first row shows the original private images while the following rows show the reconstructed images by SDAR and PCAT at various split levels 4–7. Reconstructions of the best quality among 5 trials are shown.



(a) SDAR (b) PCAT

Fig. 6. SDAR and PCAT's decoding MSE on auxiliary data X' versus private data X with ResNet-20 at split level of 7 on CIFAR-10 in vanilla SL.



(a) ResNet-20 (b) PlainNet-20

Fig. 7. Feature inference MSE over training iterations with ResNet-20 and PlainNet-20 at the split levels of 4–7 on CIFAR-10 in vanilla SL.

that facilitate the flow of information through the network.

Among the passive feature inference attacks, Fig. 4 demonstrates that the proposed attack, SDAR, surpasses existing ones [9], [13] in terms of reconstruction errors at all configurations by a significant margin. We note that SDAR achieves nearly perfect reconstruction at shallower split cuts, and is still able to achieve exceptional reconstruction quality even at deep split levels up to 7 (see Fig. 5). PCAT [13] is only able to recover private features at shallower split levels and of much lower quality than SDAR. In particular, the reconstruction quality of PCAT in the most trivial case (at level 4) is just on par with SDAR at the deepest level of 7. Moreover, the effectiveness of PCAT degenerates dramatically at deeper split levels, resulting in even larger performance gaps compared to SDAR. This is demonstrated in Fig. 5 where PCAT struggles to produce reasonable reconstruction at split levels 6–7, while SDAR is still able to reconstruct images of visibly good quality. In particular, at the deepest and most challenging split level of seven, SDAR reduces the reconstruction MSE by a significant margin of 63% on CIFAR-10 with ResNet-20. We attribute the superior performance of SDAR to the introduction of the adversarial regularization. It ensures the simulator $\tilde{f}$ to generate realistic private features Z_s' that are indistinguishable from the real ones. This leads to its decoder's improved reconstruction quality of unseen private data X. As shown in Fig. 6, while both decoders of SDAR and PCAT can successfully reconstruct auxiliary data X' (as they are trained to do so in a supervised manner), only SDAR's decoder is able to generalize well to the unseen client's private data X. PCAT's decoder suffers from severe overfitting to the auxiliary dataset, as evidenced by the vast discrepancy between its reconstruction MSE on X and X'. At last, we note that the images reconstructed by UnSplit [9] are indistinguishable at all split levels and are thus omitted here. This is expected as $\tilde{f}(\hat{X}) \approx f(X)$ does not necessarily lead to $\hat{X} \approx X$.

To further investigate the dynamics of the proposed SDAR attack, we present its attack MSE over training iterations in Fig. 7. We observe that the attack performance of SDAR converges quickly within the first few epochs, and it quickly surpasses the final attack performance of the baseline attacks. With more training iterations, the server observes more incoming intermediate representations, and the attack performance improves. It is worth noting that the attack performance of

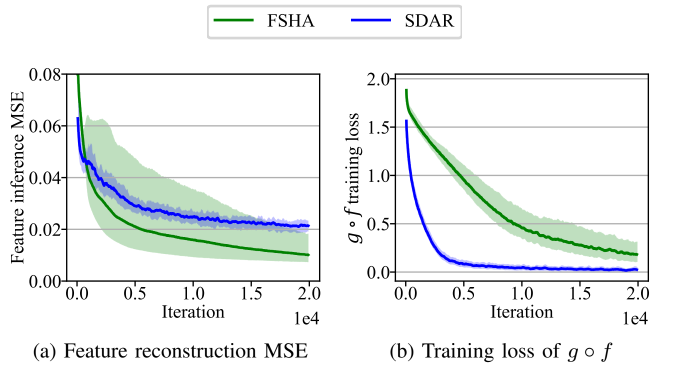


Fig. 8. Attack MSE and training losses on the original task of SDAR vs FSHA on CIFAR-10 with ResNet-20 at split level 7 in vanilla SL.

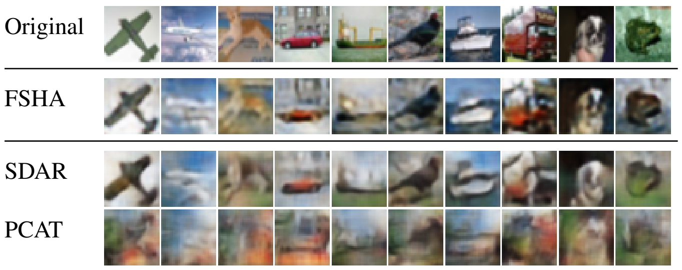


Fig. 9. Examples of feature inference attack results of FSHA, SDAR, and PCAT on CIFAR-10 with ResNet-20 at split level 7 in vanilla SL.

SDAR consistently improves over time at all levels without any signs of overfitting, indicating its robustness and effectiveness.

Comparison to active attacks. We also compare SDAR with the SOTA active feature inference attack, FSHA [38], and report the attack MSE curve over training iterations at split level 7 on CIFAR-10 in Fig 8a. Note that as an active attack where the server actively hijacks the client's gradients to manipulate the training of f, FSHA is expected to exhibit substantially better attack performance than passive attacks. Yet, we observe that our attack, despite being passive, still can achieve comparable attack performance as FSHA. As shown in Fig. 9, the visual quality of the images reconstructed by SDAR is only slightly worse than FSHA, and is significantly better than PCAT. This indicates the strong capability of our proposed attack, which is unprecedented in existing passive attacks. The reason that SDAR can achieve comparable performance to FSHA while being passive may be attributed to the adversarial regularization that enables $\tilde{f}$ to better simulate the client's model f without manipulating the training of f. A particular advantage of SDAR is the preservation of the original training task as the server does not tamper with the SL training protocol. This is demonstrated in Fig. 8b where SDAR's training loss of $g \circ f$ on X converges much faster than FHSA. Consequently, SDAR poses a greater risk to real-world applications compared to FSHA, as the original training task is preserved and the server can perform the attack of robust quality without being detected by the client [8], [12].

Effects of target model width. We have shown that deeper target model f is less vulnerable to inference attacks, and we discuss the effects of model width (number of filters) on the attack performance of SDAR in Fig. 10. Since ResNet-20 and PlainNet-20 use convolutional layers of 16/32/64 filters, we experiment with wider and narrower models by doubling and halving filters in each layer. Despite the common belief that wider models are more robust to attacks due to their higher model complexity, Fig. 10 shows that *attack performance improves with model width* instead. This is because wider models output Z_s of significantly larger dimensions, encoding more information about the private training data. It is worth noting that existing attacks [9], [13], [38] are often evaluated on *overly wide models at shallower split levels*, which, as discussed, are much easier to attack. For example, Gao and Zhang [13] evaluate PCAT on a CNN model with up to 512 filters for $32 \times 32 \times 3$ images in CIFAR-10, with the deepest split level of merely four convolutional layers allocated to the client. Our experiments with the standard model structures demonstrate that SDAR is able to attack much less vulnerable models with significantly better performance.

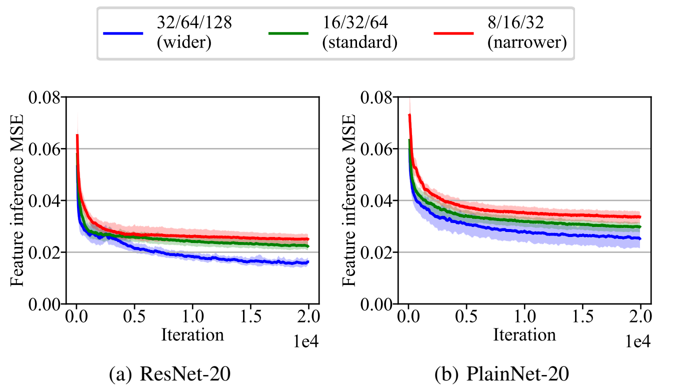


Fig. 10. Effects of target model widths on SDAR performed on CIFAR-100 with ResNet-20 and PlainNet-20 at split level 7 in vanilla SL.

TABLE II. Effects of Auxiliary Data Size on SDAR Performed on CIFAR-10 at Split Level 7 in Vanilla SL

$\lvert D'\rvert/\lvert D\rvert$	1.0	0.5	0.2	0.1	0.05
Attack MSE w/ ResNet-20	0.0212 (0.0019)	0.0225 (0.0014)	0.0232 (0.0027)	0.0248 (0.0024)	0.0297 (0.0011)
Attack MSE w/ PlainNet-20	0.0350 (0.0014)	0.0353 (0.0025)	0.0365 (0.0011)	0.0363 (0.0014)	0.0435 (0.0025)

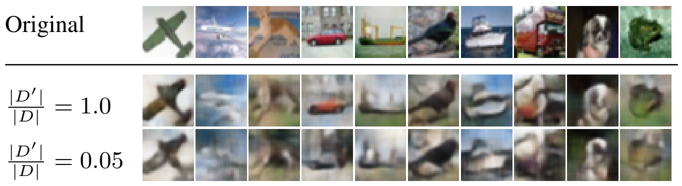


Fig. 11. Examples of feature inference attack results of SDAR performed on CIFAR-10 with ResNet-20 at split level 7 with auxiliary datasets of different sizes in vanilla SL.

Effects of auxiliary data. Naturally, one is curious to ask if the proposed attacks are still effective *when the server cannot curate a large auxiliary dataset that shares the same distribution as the client's dataset*. We hereby discuss the effects of the size and distribution of D' on SDAR. We report the attack performance of SDAR with different sizes of auxiliary dataset D' in Table II. We observe only a limited effect of the shrinkage of the auxiliary dataset on the performance of SDAR, as the attack MSE only slightly increases when D'

TABLE III. Effects of the Removal of Classes from D' on SDAR Performed on CIFAR-100 with ResNet-20 and PlainNet-20 at Split Level 7 in Vanilla SL

No. of classes removed	0	1	5	10
Attack MSE w/ ResNet-20	0.0220 (0.0014)	0.0241 (0.0015)	0.0231 (0.0008)	0.0329 (0.0070)
Attack MSE w/ PlainNet-20	0.0301 (0.0014)	0.0305 (0.0029)	0.0338 (0.0012)	0.0423 (0.0037)

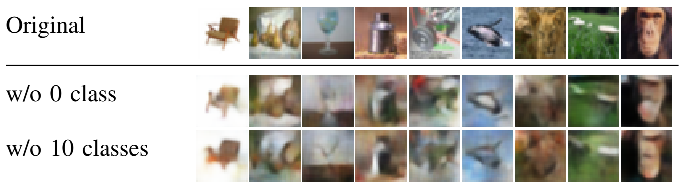


Fig. 12. Examples of feature inference attack results of SDAR performed on CIFAR-100 with ResNet-20 at split level of 7 with classes removed from the auxiliary datasets in vanilla SL.

TABLE IV. Feature Inference MSE on CIFAR-10 in Vanilla SL with or without Access to the Architecture of f

lv.	Same architecture		Different architecture	
	SDAR	PCAT [13]	SDAR	PCAT [13]
4	**0.0084 (0.0002)**	0.0237 (0.0021)	**0.0103 (0.0017)**	0.0257 (0.0034)
5	**0.0118 (0.0008)**	0.0337 (0.0048)	**0.0143 (0.0005)**	0.0289 (0.0038)
6	**0.0140 (0.0009)**	0.0382 (0.0062)	**0.0165 (0.0010)**	0.0433 (0.0114)
7	**0.0212 (0.0019)**	0.0568 (0.0062)	**0.0258 (0.0031)**	0.0545 (0.0039)

shrinks. Fig. 11 also demonstrates precise reconstructions even with auxiliary data of size 5% of the client's dataset. As for the effects of distributional discrepancies between D' and D, we experiment with SDAR on CIFAR-100 while removing examples of up to ten classes (out of 100 classes) from the auxiliary dataset in Table III. With the removal of more classes in D', the attack performance degrades, but to a limited extent. Fig. 12 demonstrates that SDAR can still reconstruct images of visibly good quality even when 10 out of 100 classes are missing in the auxiliary dataset.

Intuitively, with D' of smaller size or with many classes missing, one would expect the simulator/decoder trained on D' to be more prone to overfitting and thus to learn less useful information about client's private data. However, the core idea of SDAR is to reduce such overfitting by adversarial regularization. The positive results presented in this section demonstrate the effectiveness of adversarial regularization and show that SDAR can still work effectively even with a limited size of additional data, or with a substantial distributional discrepancy between D' and D. At last, it is worth noting that SDAR requires the auxiliary dataset's classes to be a subset of the target dataset's classes for simulator training, therefore we cannot evaluate the attack performance when the auxiliary dataset contains classes not present in the target dataset, for example, using CIFAR-100 to attack CIFAR-10. The same limitation also applies to PCAT [13]. However, as discussed previously, it is reasonable to assume that the server has knowledge of the client's data domain and can curate an auxiliary dataset from various sources, such as public datasets or even synthetic data.

Effects of simulator architecture. We have assumed that the server knows the architecture of the client's partial model f and can initialize a simulator with identical architecture. Although this is a realistic assumption in practice, it is interesting to investigate whether SDAR still has superior attack performance when the server does not know the architecture of f. To this end, we conduct additional experiments with SDAR and PCAT [13] in vanilla SL, where the server only knows the input and output dimensions of f. In particular, we use the standard ResNet-20 model for f and g, but we use plain convolutional layers for the server's simulator $\tilde{f}$ as the server does not know the architecture of the client's model but only its input and output dimensions. We report the attack performance of SDAR and PCAT in Table IV. We observe no significant loss in both attacks' performance compared to the scenario where the server's simulator shares an identical architecture as the client, and SDAR is still effective in reconstructing high-quality training examples and outperforms PCAT by a large margin. This demonstrates that although the additional knowledge of the client's model architecture can sometimes benefit the adversary of our proposed attack, such knowledge has limited effects on attack performance and SDAR does not require the server to know the architecture of f to be effective.

TABLE V. Computational Complexity of Split Learning with Various Inference Attacks

Method	FLOPs per iteration
SL w/o attack	$\mathcal{O}(b(F_f + F_g))$
SL w/ PCAT	$\mathcal{O}(b(F_f + F_g + F_{\tilde{f}} + F_{\tilde{f}^{-1}}))$
SL w/ SDAR	$\mathcal{O}(b(F_f + F_g + F_{\tilde{f}} + F_{\tilde{f}^{-1}} + F_{d_1} + F_{d_2}))$
SL w/ UnSplit	$\mathcal{O}(b(F_f + F_g + mF_{\tilde{f}}))$

TABLE VI. Average Runtime per Iteration (s) of Split Learning with Various Inference Attacks on CIFAR-10 with ResNet-20 at Different Split Levels in Vanilla SL

Split level	4	5	6	7
SL w/ PCAT	0.113 (0.000)	0.120 (0.000)	0.124 (0.009)	0.132 (0.008)
SL w/ SDAR	0.164 (0.001)	0.164 (0.004)	0.169 (0.001)	0.174 (0.002)
SL w/ UnSplit	4212.754	5019.434	5718.145	6929.952

Computational complexity. As analyzed in Section IV-C, SL training with SDAR is asymptotically as computational efficient as the standard SL training process, when fixing model complexity. We present the FLOPs required by SL with PCAT, SDAR, and UnSplit attacks in Table V. The computational complexity of SDAR and PCAT follows the analysis in Section IV-C, while in UnSplit, the server needs to perform an inner loop of iterative optimization process to attack each batch of private examples for every iteration. Let the number of optimization steps be m, then the UnSplit requires a total of $\mathcal{O}(b(F_f + F_g + mF_{\tilde{f}}))$ FLOPs per iteration for SL training and private feature reconstruction, significantly more computationally expensive than SDAR and PCAT. We also present the runtime of SL with different inference attacks in a controlled environment with one half of an A100 80GB GPU (as a virtual MIG device) in Table VI. We observe that SDAR is only slightly slower than PCAT, and both are significantly faster than UnSplit. The difference between SDAR and PCAT, albeit limited, indicates the additional costs of adversarial regularization. This shows that, compared to PCAT, SDAR achieves high-quality attack performance with a reasonable trade-off in computational cost.

TABLE VII. Ablation Studies on CIFAR-10 with ResNet-20 at Split Level 7 in the Vanilla Split Learning Setting

	d_1 (for $\tilde{f}$)	d_2 (for $\tilde{f}^{-1}$)	Conditional	Attack MSE
1	✓	✓	✓	**0.0212 (0.0019)**
2	✗	✓	✓	0.0579 (0.0053)
3	✓	✗	✓	0.0234 (0.0015)
4	✗	✗	✓	0.0607 (0.0093)
5	✓	✓	✗	0.0229 (0.0021)

Ablation studies. To conclude this section, we conduct ablation studies in Table VII to evaluate the effectiveness of each component in SDAR. We observe that all components are crucial, as removing any of them results in worse attack performance. In particular, attack performance degenerates the most when the server does not use adversarial regularization (case 4), showing its effectiveness to reduce overfitting and improve the generalization of the simulator/decoder. Between the two adversarial regularizers for $\tilde{f}$ and $\tilde{f}^{-1}$, SDAR is more sensitive to the former (case 2 vs 3). This is because the successful generalization of the decoder heavily relies on the quality of the simulator. At last, we notice a marginal performance gain by utilizing labels of the private examples in the style of CGANs [33] (case 1 vs 5), when the discriminators and decoders are fed with additional information.

C. Experiments on U-shaped SL attacks

Attack effectiveness. Under U-shaped split learning, we report the feature reconstruction MSE and the label inference accuracy of SDAR in Fig. 13, and present examples of private feature reconstructions in Fig. 14. Note that we no longer present results of UnSplit [9] here due to limited space and its inability to produce distinguishable results even in the vanilla SL setting. Similar to feature inference attacks in vanilla SL, SDAR can reconstruct private features of high quality across various split levels, and there is no significant performance drop compared to vanilla SL. The reason is that although the server no longer has access to the last part of the model, i.e., h, it manages to train a simulator $\tilde{h}$ that learns the same representations, as supported by the high label inference accuracy which consistently reaches around 98% on CIFAR-10 across all split levels. As the server trains $\tilde{h} \circ g \circ \tilde{f}$ solely on D' (while g is trained on D), its high test accuracy on unseen examples in D demonstrates that information about D is indeed leaked to the server via the trained parameters of g. In contrast, PCAT [13] achieves visibly worse feature reconstruction performance than in vanilla SL, since PCAT is unable to train an effective simulator to h, as supported by its poor label inference accuracy. With deeper split levels and fewer parameters in the server's g, less information about D is leaked via g, and hence the label inference accuracy of PCAT dramatically degenerates to the level of random guessing when the split level increases to 7.

Effects of random label flipping. Many of the properties of SDAR in vanilla SL discussed in Section V-B, e.g. the effects of target model width, auxiliary data, simulator architecture and the components of adversarial regularizers, directly apply to the U-shaped SL, and here we focus on the effects of random label flipping on the training of $\tilde{h}$. Fig. 15 shows that after removing random label flipping, the label inference accuracy of SDAR drops significantly, and in the worst case, the label inference accuracy quickly vanishes to near-zero, even worse than random guessing. Thus, without label flipping, server's model $\tilde{h}$ will quickly overfit to its own dataset D' without simulating the behavior of h on D. As a result, the simulator $\tilde{f}$ trained together with $\tilde{h}$ will not learn the same representations as f and hence the decoder can no longer generalize to decode the client's f, leading to poor feature inference performance as well. With random label flipping, the server's model $\tilde{h}$ is forced to learn the general representations over time instead of overfitting to D'.

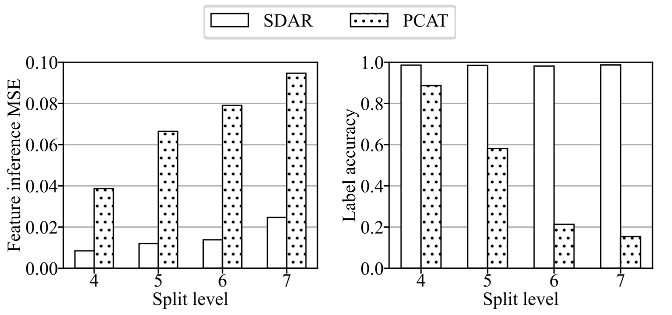


Fig. 13. Mean feature inference MSE (left) and label inference accuracy (right) with ResNet-20 at split levels of 4–7 on CIFAR-10 in U-shaped SL.

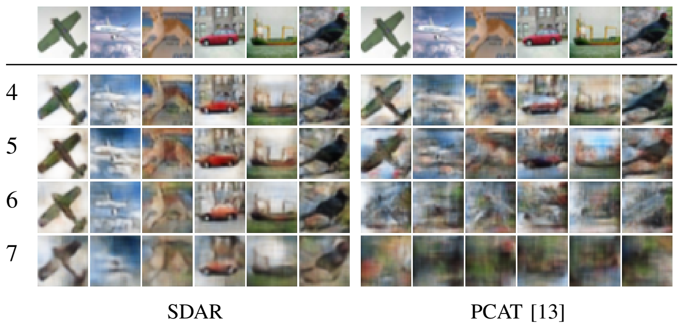


Fig. 14. Examples of feature inference attack results with ResNet-20 on CIFAR-10 in U-shaped SL. See Fig. 5 for detailed descriptions.

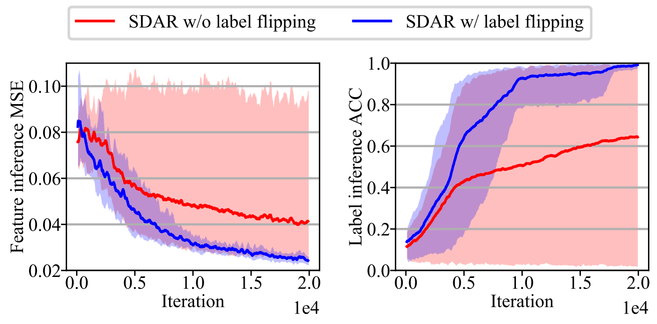


Fig. 15. Effects of random label flipping on SDAR performed on CIFAR-10 with ResNet-20 at split level 7 in U-shaped SL.

D. Non-i.i.d. multi-client scenarios

So far, our experiments focus on the setting where all batches of private training data are independently and identically distributed. This is naturally true for a single-client scenario and remains so for multiple homogeneous clients. In real-world practice, the server often serves multiple clients, who

TABLE VIII. EFFECTS OF CLIENT HETEROGENEITY ON SDAR PERFORMED ON TINY IMAGENET WITH RESNET-20 AT SPLIT LEVEL 7 IN VANILLA SL

No. of hetero. clients	1	2	5	10
Attack MSE	0.0300 (0.0016)	0.0383 (0.0012)	0.0395 (0.0023)	0.0419 (0.0025)

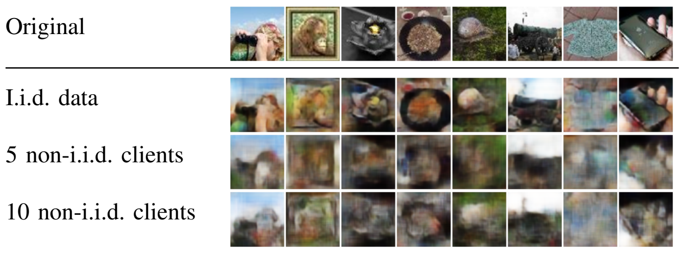


Fig. 16. Examples of feature inference results of SDAR performed in single-client (equivalently i.i.d. multi-client) vs heterogeneous multi-client settings on Tiny ImageNet with ResNet-20 at split level 7 in the vanilla SL setting.

possess heterogeneous data distributions. To investigate the effects of data heterogeneity on SDAR, we conduct additional evaluations under a heterogeneous k-client setting, where $k \in \{2, 5, 10\}$ clients possess different data distributions. In this setting, each client holds $200/k$ classes of Tiny ImageNet, and they take turns to interact with the server. We report SDAR's feature inference performance in Table VIII and provide examples of reconstruction results in Fig. 16. We observe that the attack performance slightly degrades as the number of heterogeneous clients increases, but the visual quality of the reconstructed features remains acceptable. This suggests that SDAR can still be effective in the multi-client setting, even when clients possess heterogeneous data distributions. The performance degradation could be attributed to the fact that in the non-i.i.d. setting, the updates from different clients are not homogeneous, which may result in issues with the convergence of partial models, thus a worse simulator on the server side. Nevertheless, although the incoming batches of representations are not homogeneous, the server knows which client the batch is coming from and can curate a specific auxiliary dataset for each client to ensure $\tilde{f}$ to faithfully simulate the client's model. Therefore, we observe limited performance drop in the heterogeneous multi-client setting.

VI. POTENTIAL COUNTERMEASURES

Change of model architectures. We have demonstrated the effectiveness of SDAR on ResNet-20 and PlainNet-20 (in Fig. 4 and Fig. 13) and their narrower variants (in Fig. 10) at the deep split level of seven. Since narrower models at deeper split levels are relatively more robust to inference attacks, a natural defense against SDAR is to assign more layers to the client's model. We evaluate SDAR at split levels up to nine and present its attack performance in Fig. 17. Note that at split level 9, the client hosts all convolutional layers, and the server has merely 650 parameters. This is an extreme case that is not practical in real-world deployments, as this invalidates SL's purpose of relieving the client's computational burden with a capable server. From Fig. 17, we see that even at level 9, SDAR's attack MSE continues to decrease over time without saturation. Therefore, we conclude that SDAR is robust against

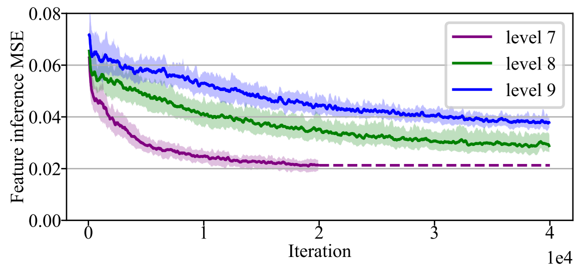


Fig. 17. Attack MSE of SDAR on CIFAR-10 with ResNet-20 at deeper split levels of 7–9 in the vanilla SL configuration.

TABLE IX. SDAR ATTACK MSE AND SL TRAINING ACCURACY ON CIFAR-100 WITH RESNET-20 AT SPLIT LEVEL 7 IN VANILLA SL AT VARIOUS DROPOUT RATES

Dropout rate	0.0	0.1	0.2	0.4	0.8
Attack MSE	0.0220 (0.0013)	0.0212 (0.0001)	0.0226 (0.0001)	0.0237 (0.0019)	0.0250 (0.0026)
Train ACC (%)	96.32 (0.49)	93.03 (0.40)	91.43 (0.20)	86.07 (0.34)	40.19 (0.61)

TABLE X. SDAR ATTACK MSE ON CIFAR-10 WITH RESNET-20 AT SPLIT LEVEL 7 IN VANILLA SL WITH ℓ_1/ℓ_2 REGULARIZATION

Regularization factor	0.0	0.001	0.01	0.1
Attack MSE on SL with ℓ_1 regularization	0.0212 (0.0019)	0.0.0208 (0.0012)	0.0228 (0.0017)	0.0221 (0.0017)
Attack MSE on SL with ℓ_2 regularization	0.0212 (0.0019)	0.0231 (0.0020)	0.0209 (0.0008)	0.0220 (0.0019)

different split levels, and changing the model architecture is not a viable defense against our attacks.

Regularization. Overfitting is a key concern in inference attacks on ML models [34], [42], as ML models may overfit to and unintentionally memorize private training data. Dropout [44] and ℓ_1/ℓ_2 regularization [35] are straightforward yet effective techniques to prevent overfitting, and recent studies [25], [31], [41], [42] have shown that they can also mitigate inference attacks on ML models. Therefore, we consider these two methods as a potential defense and evaluate SDAR against them. As shown in Table IX and Table X, the introduction of dropout or ℓ_1/ℓ_2 regularization with a higher rate or factor slightly degrades the attack performance of SDAR, but the attack MSE remains at a low level, indicating the attack effectiveness against dropout. Table IX also shows that large dropout rates can lead to significant degradation in the training accuracy of SL. The reason that our attacks are robust against regularization defense is due to the fact that the same regularization is also applied in the training of the simulator, so the $\tilde{f}$ can simulate the regularized behavior of the client's f. Therefore, dropout and ℓ_1/ℓ_2 regularization are not viable defenses against SDAR.

Decorrelation. A recent method for enhancing the privacy of SL is to decorrelate the intermediate representations Z_s with the input private features X [52], [53]. This is done by replacing $g \circ f$'s loss function $\ell(g(f(X)), Y)$ with

$$(1 - \alpha)\ell(g(f(X)), Y) + \alpha \cdot \mathrm{dCol}(f(X), X)$$

while training, where $\alpha \in [0, 1]$ is the decorrelation parameter

TABLE XI. SDAR ATTACK MSE ON CIFAR-100 WITH RESNET-20 AT LEVEL 7 IN VANILLA SL WITH THE DECORRELATION DEFENSE

α	0.0	0.1	0.2	0.4	0.8
Attack MSE	0.0220 (0.0013)	0.0257 (0.0012)	0.0314 (0.0023)	0.0414 (0.0017)	0.0433 (0.0024)
Train ACC (%)	96.32 (0.49)	96.31 (0.30)	94.83 (0.21)	94.31 (0.39)	74.57 (2.98)

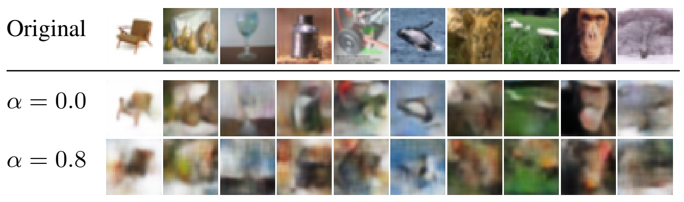


Fig. 18. Examples of reconstruction results on CIFAR-100 with ResNet-20 at split level 7 in vanilla SL with decorrelation defense with $\alpha \in \{0, 0.8\}$.

and dCol$(\cdot, \cdot)$ is the distance correlation [46]. The decorrelation term encourages the intermediate representations Z_s to be decorrelated with the input private features X, thus making it harder for the server to infer private training data from Z_s. Larger α values result in stronger decorrelation and hence better privacy preservation, but also greater loss in the model utility of the original task.

To evaluate the effectiveness of SDAR against this decorrelation defense, we introduce a decorrelation term in Eq. (3) to decorrelate simulator's output $\tilde{f}(X')$ with auxiliary data X' in SDAR, such that the simulator can learn the same decorrelation behaviors of client's f. We perform SDAR on SL with decorrelation with $\alpha \in \{0.1, 0.2, 0.4, 0.8\}$. As shown in Table XI and Fig. 18, although decorrelation manages to degrade the attack quality when α increases, SDAR still produces visibly effective reconstructions even at $\alpha = 0.8$. Also, Table XI shows that the decorrelation defense with larger α results in a notable loss in accuracy of the original task, rendering this defense impractical in real-world applications due to the trade-off between privacy and utility. Therefore, decorrelation is not a viable defense against SDAR.

Homomorphic encryption. The most straightforward idea to mitigate privacy issues in SL is to encrypt the intermediate representations Z_s and send only the ciphertext to the server s.t. the server cannot use it to infer private features. This usually requires homomorphic encryption (HE) [47], [54] where the server can only perform computation on the ciphertext without decrypting it and return the ciphertext of the result to the client. In the context of SL, this means that the server must execute forward pass, loss evaluation, and backward propagation all in ciphertext, and at last sends the gradients $\nabla\theta_{s+1}$ in ciphertext back to the client. However, basic computations in HE schemes are limited to additions and multiplications [27], and it is computationally prohibitive to execute the forward pass and backward propagation in ciphertext, which typically involves non-linear operations and necessitates the use of approximation techniques [47]. Therefore, HE can hardly be applicable to SL, especially for deep NN models.

Secure multi-party computation. Another widely-used cryptographic technique to protect privacy is secure multi-party computation (MPC) [3], [27], [55], [59], where multiple parties jointly compute a function on their private inputs without revealing their inputs to each other. To the best of our knowledge, there has been no prior work improving the privacy of SL using MPC. Even if it is possible to apply MPC such that the client and the server can train their models without revealing private data of the client or model parameters of both parties, such protocol will introduce significant computational overhead and communication cost, completely defeating the purpose and promises of split learning of communication and computation efficiency.

Differential privacy. Currently, the state-of-the-art technique to quantify and reduce information disclosure about individuals is differential privacy (DP) [1], [7], [57], which mathematically bounds the influence of private data on the released information. In the context of SL, this would mean that the intermediate representations Z_s are differentially private with regard to the client's private features X. If we are to achieve DP in split learning, random noise will be injected by the client, such that even if the client arbitrarily modifies the input data X, the intermediate representations Z_s will remain almost unchanged. That is, client's model f will output similar representations for any input data, which practically renders the model trivial. Therefore, DP is not applicable to our problem.

VII. RELATED WORK

Split learning. Split learning (SL) [19], [39], [50], [51] is a privacy preserving protocol for distributed training and inference of deep neural networks, and has gain increasing attention due to its simplicity, efficiency and scalability [26], [43], [48], [60]. Since its proposal, many works have extended SL to various configurations, including U-shaped SL that protects the client's labels [19], SL with multiple clients with horizontal [50] or vertical [4] partition. We refer the audience to Section II and available surveys [26], [43], [48], [60] for detailed review.

Inference attacks against split learning. The security of SL has been a focus of the research community since the proposal of SL. Various privacy attacks have been proposed to infer clients' labels or features under different threat models. We characterize these threat models via three dimensions: passiveness, the data assumption and the model assumption. *Passiveness* refers to whether the adversary fully complies with the SL protocol without actively manipulating the training process (by hijacking gradients, for example). A malicious server that actively manipulates the training process is considered an active or dishonest adversary, while a server that only passively observes the training process is considered a passive adversary, also known as a semi-honest or, interchangeably, an honest-but-curious adversary. *Data assumption* refers to whether the adversary has access to an auxiliary dataset of a similar distribution as the target private training data. *Model assumption* refers to whether the adversary has access to the architecture or weights of the victim client's model. Existing attacks all adopt at least one of the two assumptions. We summarize them in Table XII and discuss the feasibility of inference attacks with neither assumptions in Section VIII.

Pasquini et al. [38] propose FSHA, an active feature inference attack where the server hijacks the gradients sent back to the client to control its updates such that the client's

TABLE XII. PROPERTIES AND THREAT MODELS OF SERVER-SIDE TRAINING-TIME INFERENCE ATTACKS ON SL*

Attack	Infer feature	Infer label	Passive	Data assumption		Model assumption	
				Auxiliary feature	Auxiliary label	Model architecture	Model weights
FSHA (Pasquini et al. [38])	✓	✗	✗	●	○	◐	○
UnSplit (Erdoğan et al. erdougan2022unsplit)	✓	✓	✓	○	○	●	○
EXACT (Qiu et al. [40])	✓	✓	✓	○	○	●	●
PCAT (Gao and Zhang [13])	✓	✓	✓	●	●	◐	○
SDAR (Ours)	✓	✓	✓	●	●	◐	○

*Symbol ● indicates that the attack requires the corresponding assumption to be satisfied, while ○ indicates that the attack does not exploit the corresponding assumption. Symbol ◐ indicates that the attack does not require the corresponding assumption but can benefit from it.

model behaves similarly to the decodable encoder trained by the server as part of an autoencoder. FSHA requires the adversary to have access to an (unlabelled) auxiliary dataset similar to the target data to train the autoencoder. Although FSHA does not require the server to know the architecture of the client's model, it benefits from such knowledge when the encoder shares the same architecture as the client's model. Erdoğan et al. [9] design UnSplit, a passive feature and label inference attack via alternating optimization of a surrogate model and inferred private inputs. UnSplit achieves passiveness and requires no knowledge of the client's data, but requires the client's model architecture to be known to the server. Similarly, Qiu et al. [40] also consider passive feature and label inference attacks without the data assumption, but with the extremely strong model assumption that the server knows the model parameters of the victim clients, thus can utilize the excessive information on client's model parameters and gradients. Concurrently with our work, Gao and Zhang [13] propose a passive feature and label inference attack against SL, namely PCAT, which adopts a similar framework as our naïve SDA discussed in the beginning of Section IV-A, with additional improvements such as label alignment and delayed training. SDAR and PCAT share the same threat model, where an honest-but-curious adversary has access to a labelled auxiliary dataset similar to the target data, but has no access to the weights of the client's model. Both attacks can function without the knowledge of the client's model architecture, but may benefit from such knowledge. We demonstrate with extensive experiments in Section V that SDAR consistently outperforms PCAT across various, and in particular, challenging settings.

On a separate line of research, several works [29], [30] focus on label inference attacks. This is not our focus as we aim to develop effective attacks that can infer both the clients' labels and features. At last, several attacks [23], [61] have been proposed against split inference, also known as collaborative inference. However, split inference is more vulnerable as the intermediate representations are products of a fixed client's model, while in training-time SL attacks, the server has to invert the output of a constantly updated model. Therefore, attacks against split inference are not our focus in this work.

Privacy-preserving improvements of SL. In response to the privacy vulnerabilities of split learning, various privacy-preserving improvements have been proposed. U-shaped SL [19] enhances privacy by preventing label exposure to the server. However, our work demonstrates that SDAR can still infer private labels with high accuracy. Targeting active hijacking attacks such as FSHA [38], Erdoğan et al. [8], Fu et al. [12] detect and defend against the active manipulation of back-propagated gradients. Vepakomma et al. [52], [53] propose to improve the privacy of SL by minimizing the correlation between the private inputs and the shared representations via injecting a correlation regularization term to the loss function of the original SL task, which we investigate in Section VI.

VIII. CONCLUSION

In this paper, we investigate the privacy risks of split learning under the notion of an honest-but-curious server. We identify that existing server-side passive attacks often target impractical and vulnerable settings to gain extra advantage for the adversary. We present new passive attacks on split learning, namely, SDAR, by utilizing GAN-inspired adversarial regularization to learn a decodable simulator of client's private model that produces representations indistinguishable from the client's. Empirically, we show that SDAR is effective in inferring private features of the client in vanilla split learning, and both private features and labels in the U-shaped split learning, under challenging settings where existing passive attacks fail to produce non-trivial results. At last, we propose and evaluate potential defenses against our attacks and highlight the need for improving SL to further protect the client's private data.

Split learning is a promising protocol that enables distributed training and inference of neural networks on devices with limited computational resources. There are many open challenges and potential directions regarding its security. One limitation of SDAR (and PCAT) is that they require the adversary to have access to a *labelled* auxiliary dataset in the same domain as the target data. While state-of-the-art active attacks like FSHA do not require the labels of the auxiliary dataset, it is interesting to investigate whether passive attacks are feasible only with an unlabelled auxiliary dataset. Also, it is worth noting that all attacks discussed in Section VII and listed in Table XII assume at least one of the model and the data assumptions, and it remains an open question whether it is possible to attack SL with neither assumptions. Intuitively, if the server has no knowledge on either the structure/weights of the client's model f or the domain/distribution of its data X, it becomes very challenging, if not impossible, to recover X merely by observing $Z = f(X)$. It's interesting to explore more capable attacks that invert an inaccessible and unknown function operating on an unknown domain. One may also study the privacy guarantee of SL from a theoretical lens and rigorously measure the information leakage of the protocol. At last, it will be an interesting direction to design privacy-preserving mechanisms to mitigate our proposed attacks and improve the privacy of SL.

Acknowledgements

This research is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG3-RP-2022-029). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore. Yuncheng Wu's work is supported by the Fundamental Research Funds for the Central Universities, and the Research Funds of Renmin University of China (24XNKJ08).

References

[1] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *CCS*, 2016, pp. 308–318.

[2] J. Behrmann, W. Grathwohl, R. T. Chen, D. Duvenaud, and J.-H. Jacobsen, "Invertible residual networks," in *ICML*, 2019, pp. 573–582.

[3] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *CCS*, 2017, pp. 1175–1191.

[4] I. Ceballos, V. Sharma, E. Mugica, A. Singh, A. Roman, P. Vepakomma, and R. Raskar, "SplitNN-driven vertical partitioning," *arXiv preprint arXiv:2008.04137*, 2020.

[5] A. Coates, H. Lee, and A. Y. Ng, "An analysis of single layer networks in unsupervised feature learning," in *AISTATS*, 2011.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *CVPR*, 2009.

[7] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy," *Found. Trends Theor. Comput. Sci.*, vol. 9, no. 3–4, pp. 211–407, 2014.

[8] E. Erdogan, A. Küpçü, and A. E. Cicek, "Splitguard: Detecting and mitigating training-hijacking attacks in split learning," in *WPES*, 2022, pp. 125–137.

[9] E. Erdoğan, A. Küpçü, and A. E. Çiçek, "Unsplit: Data-oblivious model inversion, model stealing, and label inference attacks against split learning," in *WPES*, 2022, pp. 115–124.

[10] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *CCS*, 2015, pp. 1322–1333.

[11] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page, and T. Ristenpart, "Privacy in pharmacogenetics: An End-to-End case study of personalized warfarin dosing," in *USENIX Security*, 2014, pp. 17–32.

[12] J. Fu, X. Ma, B. B. Zhu, P. Hu, R. Zhao, Y. Jia, P. Xu, H. Jin, and D. Zhang, "Focusing on pinocchio's nose: A gradients scrutinizer to thwart split-learning hijacking attacks using intrinsic attributes," in *NDSS*, 2023.

[13] X. Gao and L. Zhang, "PCAT: Functionality and data stealing from split learning by pseudo-client attack," in *USENIX Security*, 2023, pp. 5271–5288.

[14] Y. Gao, M. Kim, S. Abuadbba, Y. Kim, C. Thapa, K. Kim, S. A. Camtep, H. Kim, and S. Nepal, "End-to-end evaluation of federated learning and split learning for internet of things," in *SRDS*, 2020, pp. 91–100.

[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[16] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *NeurIPS*, 2014.

[17] ——, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[18] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," in *NeurIPS*, 2017.

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

[20] Y. J. Ha, G. Lee, M. Yoo, S. Jung, S. Yoo, and J. Kim, "Feasibility study of multi-site split learning for privacy-preserving medical systems under data imbalance constraints in covid-19, x-ray, and cholesterol dataset," *Scientific Reports*, vol. 12, no. 1, pp. 1–11, 2022.

[21] A. J. Hall. (2020) Implementing split neural networks on pysyft. [Online]. Available: https://blog.openmined.org/split-neural-networks-on-pysyft/

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[23] Z. He, T. Zhang, and R. B. Lee, "Model inversion attacks against collaborative inference," in *ACSAC*, 2019, pp. 148–162.

[24] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[25] Y. Jiang, X. Luo, Y. Wu, X. Zhu, X. Xiao, and B. C. Ooi, "On data distribution leakage in cross-silo federated learning," *TKDE*, pp. 1–17, 2024.

[26] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[27] M. Keller, "MP-SPDZ: A versatile framework for multi-party computation," in *CCS*, 2020, pp. 1575–1590.

[28] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

[29] O. Li, J. Sun, X. Yang, W. Gao, H. Zhang, J. Xie, V. Smith, and C. Wang, "Label leakage and protection in two-party split learning," in *ICLR*, 2022.

[30] J. Liu and X. Lyu, "Clustering label inference attack against practical split learning," *arXiv preprint arXiv:2203.05222*, 2023.

[31] X. Luo, Y. Wu, X. Xiao, and B. C. Ooi, "Feature inference attack on model predictions in vertical federated learning," in *ICDE*, 2021, pp. 181–192.

[32] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *AISTATS*, 2017, pp. 1273–1282.

[33] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[34] M. Nasr, R. Shokri, and A. Houmansadr, "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning," in *S&P*, 2019, pp. 739–753.

[35] A. Y. Ng, "Feature selection, l1 vs. l2 regularization, and rotational invariance," in *ICML*, 2004.

[36] A. Odena, V. Dumoulin, and C. Olah, "Deconvolution and checkerboard artifacts," *Distill*, vol. 1, no. 10, p. e3, 2016.

[37] OpenMinded, "Pysyft," 2021. [Online]. Available: https://github.com/OpenMined/PySyft

[38] D. Pasquini, G. Ateniese, and M. Bernaschi, "Unleashing the tiger: Inference attacks on split learning," in *CCS*, 2021, pp. 2113–2129.

[39] M. G. Poirot, P. Vepakomma, K. Chang, J. Kalpathy-Cramer, R. Gupta, and R. Raskar, "Split learning for collaborative deep learning in healthcare," *arXiv preprint arXiv:1912.12115*, 2019.

[40] X. Qiu, I. Leontiadis, L. Melis, A. Sablayrolles, and P. Stock, "Evaluating privacy leakage in split learning," *arXiv preprint arXiv:2305.12997*, 2023.

[41] A. Salem, Y. Zhang, M. Humbert, P. Berrang, M. Fritz, and M. Backes, "Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models," in *NDSS*, 2019.

[42] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *S&P*, 2017, pp. 3–18.

[43] A. Singh, P. Vepakomma, O. Gupta, and R. Raskar, "Detailed comparison of communication efficiency of split learning and federated learning," *arXiv preprint arXiv:1909.09145*, 2019.

[44] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *JMLR*, vol. 15, no. 1, pp. 1929–1958, 2014.

[45] "Tiny imagenet," http://cs231n.stanford.edu/tiny-imagenet-200.zip, Stanford University, accessed: 2023-12-28.

[46] G. J. Székely, M. L. Rizzo, and N. K. Bakirov, "Measuring and testing dependence by correlation of distances," *Ann. Stat.*, vol. 35, no. 6, p. 2769, 2007.

[47] H. Takabi, E. Hesamifard, and M. Ghasemi, "Privacy preserving multi-party machine learning with homomorphic encryption," in *NeurIPS*, 2016.

[48] C. Thapa, M. A. P. Chamikara, and S. A. Camtepe, "Advancements of federated learning towards privacy preservation: from federated learning to split learning," *Federated Learning Systems: Towards Next-Generation AI*, pp. 79–109, 2021.

[49] J.-B. Truong, P. Maini, R. J. Walls, and N. Papernot, "Data-free model extraction," in *CVPR*, 2021, pp. 4771–4780.

[50] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," *arXiv preprint arXiv:1812.00564*, 2018.

[51] P. Vepakomma and R. Raskar, *Split Learning: A Resource Efficient Model and Data Parallel Approach for Distributed Deep Learning*. Springer, 2022, pp. 439–451.

[52] P. Vepakomma, A. Singh, O. Gupta, and R. Raskar, "Nopeek: Information leakage reduction to share activations in distributed deep learning," in *ICDM Workshops*, 2020, pp. 933–942.

[53] P. Vepakomma, A. Singh, E. Zhang, O. Gupta, and R. Raskar, "Nopeek-infer: Preventing face reconstruction attacks in distributed inference after on-premise training," in *FG*, 2021, pp. 1–8.

[54] A. Wood, K. Najarian, and D. Kahrobaei, "Homomorphic encryption for machine learning in medicine and bioinformatics," *ACM Computing Surveys*, vol. 53, no. 4, pp. 1–35, 2020.

[55] Y. Wu, S. Cai, X. Xiao, G. Chen, and B. C. Ooi, "Privacy preserving vertical federated learning for tree-based models," *PVLDB*, vol. 13, no. 12, pp. 2090–2103, Jul 2020.

[56] Y. Wu, N. Xing, G. Chen, T. T. A. Dinh, Z. Luo, B. C. Ooi, X. Xiao, and M. Zhang, "Falcon: A privacy-preserving and interpretable vertical federated learning system," *PVLDB*, vol. 16, no. 10, pp. 2471–2484, 2023.

[57] X. Xiao, G. Wang, and J. Gehrke, "Differential privacy via wavelet transforms," *IEEE TKDE*, vol. 23, no. 8, pp. 1200–1214, 2010.

[58] Z. Yang, J. Zhang, E.-C. Chang, and Z. Liang, "Neural network inversion in adversarial setting via background knowledge alignment," in *CCS*, 2019, pp. 225–240.

[59] A. C. Yao, "Protocols for secure computations," in *FOCS*, 1982, pp. 160–164.

[60] X. Yin, Y. Zhu, and J. Hu, "A comprehensive survey of privacy-preserving federated learning: A taxonomy, review, and future directions," *ACM Computing Surveys*, vol. 54, no. 6, pp. 1–36, 2021.

[61] Y. Yin, X. Zhang, H. Zhang, F. Li, Y. Yu, X. Cheng, and P. Hu, "Ginver: Generative model inversion attacks against collaborative inference," in *WWW*, 2023, pp. 2122–2131.

[62] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *NeurIPS*, 2014.

[63] X. Zhao, W. Zhang, X. Xiao, and B. Lim, "Exploiting explanations for model inversion attacks," in *ICCV*, 2021, pp. 682–692.

[64] X. Zhu, X. Luo, Y. Wu, Y. Jiang, X. Xiao, and B. C. Ooi, "Passive inference attacks on split learning with adversarial regularization," *arXiv preprint arXiv:2310.10483*, 2024.

Appendix A
Pseudocode

We provide the pseudocode of SDAR against vanilla SL in Fig. 19, and against U-shaped SL in Fig. 20.

```
1:  procedure CLIENT:
2:      Initializes model $f$ with parameters $\theta_f$
3:      for $i \in \{1, 2, \dots\}$ do
4:          Samples a batch of examples $(X, Y) \in D$
5:          $Z_s \leftarrow f(X)$
6:          Sends $(Z_s, Y)$ to the server                ▷ Timestamp $t_{i1}$
7:          Receives $\nabla\theta_{s+1}$ from the server  ▷ Timestamp $t_{i2}$
8:          Calculates $\nabla\theta_f$ using $\nabla\theta_{s+1}$
9:          $\theta_f \leftarrow \theta_f - \eta\nabla\theta_f$   ▷ Original SL task
10:
11: procedure SERVER:
12:     Initializes model $g$ with parameters $\theta_g$
13:     Initializes models $\tilde{f}, \tilde{f}^{-1}, d_1, d_2$ w/ $\theta_{\tilde{f}}, \theta_{\tilde{f}^{-1}}, \theta_{d_1}, \theta_{d_2}$
14:     for $i \in \{1, 2, \dots\}$ do
15:         Receives $(Z_s, Y)$ from the client           ▷ Timestamp $t_{i1}$
16:         $\hat{Y} \leftarrow g(Z_s)$
17:         $\mathcal{L} \leftarrow \ell(\hat{Y}, Y)$
18:         $\nabla\theta_g \leftarrow \partial\mathcal{L}/\partial\theta_g$
19:         Sends $\nabla\theta_{s+1}$ to the client       ▷ Timestamp $t_{i2}$
20:         $\theta_g \leftarrow \theta_g - \eta\nabla\theta_g$   ▷ Original SL task
21:
22:         Samples a batch of examples $(X', Y') \in D'$
23:         $Z'_s \leftarrow \tilde{f}(X')$; $\hat{Y}' \leftarrow g(Z'_s)$
24:         $\hat{X}' \leftarrow \tilde{f}^{-1}(Z'_s, Y')$; $\hat{X} \leftarrow \tilde{f}^{-1}(Z_s, Y)$
25:         $\mathcal{L}_{\tilde{f}} \leftarrow \ell(\hat{Y}', Y') + \lambda_1 \ell_{\text{BCE}}(d_1(Z'_s, Y'), 1)$
26:         $\mathcal{L}_{d_1} \leftarrow \ell_{\text{BCE}}(d_1(Z'_s, Y'), 0) + \ell_{\text{BCE}}(d_1(Z_s, Y), 1)$
27:         $\mathcal{L}_{\tilde{f}^{-1}} \leftarrow \ell_{\text{MSE}}(X', \hat{X}') + \lambda_2 \cdot \ell_{\text{BCE}}(d_2(\hat{X}, Y), 1)$
28:         $\mathcal{L}_{d_2} \leftarrow \ell_{\text{BCE}}(d_2(\hat{X}, 0) + \ell_{\text{BCE}}(d_2(X', Y'), 1)$
29:
30:         $\nabla\theta_{\tilde{f}} \leftarrow \partial\mathcal{L}_{\tilde{f}}/\partial\theta_{\tilde{f}}$; $\theta_{\tilde{f}} \leftarrow \theta_{\tilde{f}} - \eta_{\tilde{f}}\nabla\theta_{\tilde{f}}$
31:         $\nabla\theta_{d_1} \leftarrow \partial\mathcal{L}_{d_1}/\partial\theta_{d_1}$; $\theta_{d_1} \leftarrow \theta_{d_1} - \eta_{d_1}\nabla\theta_{d_1}$
32:         $\nabla\theta_{\tilde{f}^{-1}} \leftarrow \partial\mathcal{L}_{\tilde{f}^{-1}}/\partial\theta_{\tilde{f}^{-1}}$; $\theta_{\tilde{f}^{-1}} \leftarrow \theta_{\tilde{f}^{-1}} - \eta_{\tilde{f}^{-1}}\nabla\theta_{\tilde{f}^{-1}}$
33:         $\nabla\theta_{d_2} \leftarrow \partial\mathcal{L}_{d_2}/\partial\theta_{d_2}$; $\theta_{d_2} \leftarrow \theta_{d_2} - \eta_{d_2}\nabla\theta_{d_2}$
34:
35:         $\hat{X} \leftarrow \tilde{f}^{-1}(Z_s, Y)$   ▷ Reconstructed private features
```

Fig. 19. SDAR against vanilla split learning. $\ell(\cdot,\cdot)$ is the loss function of the original SL task; s is the split level; η is the learning rate of the original SL task; $\eta_{\tilde{f}}, \eta_{\tilde{f}^{-1}}, \eta_{d_1}, \eta_{d_2}$ are learning rates for the server's models $\tilde{f}, \tilde{f}^{-1}, d_1, d_2$.

Appendix B
Implementation details

A. Experiment setup

The experiments are conducted on machines running Ubuntu 20.04 LTS, equipped with two Intel® Xeon® Gold 6326 CPUs, 256GB of RAM and an NVIDIA® A100 80GB GPU. We implement our attack and other baselines in Python and TensorFlow. Our code base is available at https://github.com/zhxchd/SDAR_SplitNN/ for reproducibility.

B. Split learning setup

We target the same SL setup across all experiments, where the complete model H is either ResNet-20 or PlainNet-20, and we experiment with various split configurations of each model. We show the model architectures of the target models and their split configurations in Fig. 21. For all experiments, we use Adam optimizer to optimize H (that is, $g \circ f$ in vanilla SL and $h \circ g \circ f$ in U-shaped SL) with default initial learning rate $\eta = 0.001$. We use 128 examples per batch as per the original ResNet paper suggests [22] except STL-10, where we use a batch size of 32 as STL-10 only contains 13,000 images

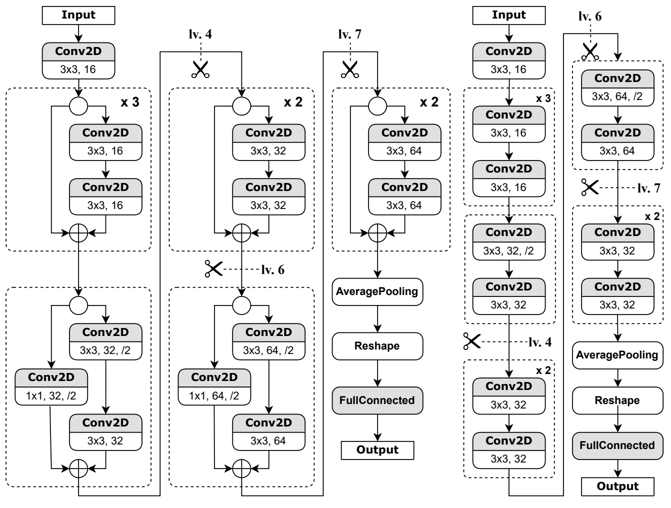
```
1:  procedure CLIENT:
2:      Initializes models $f, h$ with parameters $\theta_f, \theta_h$
3:      for $i \in \{1, 2, \ldots\}$ do
4:          Samples a batch of examples $(X, Y) \in D$
5:          $Z_s \leftarrow f(X)$
6:          Sends $Z_s$ to the server                          ▷ Timestamp $t_{i1}$
7:          Receives $Z_t$ from the server                     ▷ Timestamp $t_{i2}$
8:          $\hat{Y} \leftarrow h(Z_t)$; $\mathcal{L} \leftarrow \ell(\hat{Y}, Y)$
9:          $\nabla\theta_h \leftarrow \partial\mathcal{L}/\partial\theta_h$; $\theta_h \leftarrow \theta_h - \eta\nabla\theta_h$    ▷ Original SL task
10:         Sends $\nabla\theta_{t+1}$ to the server           ▷ Timestamp $t_{i3}$
11:         Receives $\nabla\theta_{s+1}$ from the server      ▷ Timestamp $t_{i4}$
12:         Calculates $\nabla\theta_f$ using $\nabla\theta_{s+1}$
13:         $\theta_f \leftarrow \theta_f - \eta\nabla\theta_f$               ▷ Original SL task
14:
15: procedure SERVER:
16:     Initializes $g$ with parameters $\theta_g$
17:     Initializes $\tilde{h}, \tilde{f}, \tilde{f}^{-1}, d_1, d_2$ with param. $\theta_{\tilde{h}}, \theta_{\tilde{f}}, \theta_{\tilde{f}^{-1}}, \theta_{d_1}, \theta_{d_2}$
18:     for $i \in \{1, 2, \ldots\}$ do
19:         Receives $(Z_s, Y)$ from the client                ▷ Timestamp $t_{i1}$
20:         $Z_t = g(Z_s)$
21:         Sends $Z_t$ to the client                          ▷ Timestamp $t_{i2}$
22:         Receives $\nabla\theta_{t+1}$ from the client      ▷ Timestamp $t_{i3}$
23:         Calculates $\nabla\theta_g$ using $\nabla\theta_{t+1}$
24:         $\theta_g \leftarrow \theta_g - \eta\nabla\theta_g$                ▷ Original SL task
25:         Sends $\nabla\theta_{s+1}$ to the client           ▷ Timestamp $t_{i4}$
26:
27:         Samples a batch of examples $(X', Y') \in D'$
28:         $Z'_s \leftarrow \tilde{f}(X')$; $Z'_t \leftarrow g(Z'_s)$; $\hat{Y}' \leftarrow \tilde{h}(Z'_t)$
29:         $\hat{X}' \leftarrow \tilde{f}^{-1}(Z'_s)$; $\hat{X} \leftarrow \tilde{f}^{-1}(Z_s)$
30:         $\tilde{Y}'_i \leftarrow \begin{cases} Y'_i & \text{w.p. } 1-p \\ \text{Uniform}(\mathcal{Y}) & \text{w.p. } p \end{cases}$    ▷ Random flipping
31:         $\mathcal{L}_{\tilde{h}} \leftarrow \ell(\hat{Y}', \tilde{Y}')$
32:         $\mathcal{L}_{\tilde{f}} \leftarrow \ell(\hat{Y}', \tilde{Y}') + \lambda_1\ell_{\text{BCE}}(d_1(Z'_s), 1)$
33:         $\mathcal{L}_{d_1} \leftarrow \ell_{\text{BCE}}(d_1(Z'_s), 0) + \ell_{\text{BCE}}(d_1(Z_s), 1)$
34:         $\mathcal{L}_{\tilde{f}^{-1}} \leftarrow \ell_{\text{MSE}}(X', \hat{X}') + \lambda_2\ell_{\text{BCE}}(d_2(\hat{X}), 1)$
35:         $\mathcal{L}_{d_2} \leftarrow \ell_{\text{BCE}}(d_2(\hat{X}), 0) + \ell_{\text{BCE}}(d_2(X'), 1)$
36:
37:         $\nabla\theta_{\tilde{h}} \leftarrow \partial\mathcal{L}_{\tilde{h}}/\partial\theta_{\tilde{h}}$; $\theta_{\tilde{h}} \leftarrow \theta_{\tilde{h}} - \eta_{\tilde{h}}\nabla\theta_{\tilde{h}}$
38:         $\nabla\theta_{\tilde{f}} \leftarrow \partial\mathcal{L}_{\tilde{f}}/\partial\theta_{\tilde{f}}$; $\theta_{\tilde{f}} \leftarrow \theta_{\tilde{f}} - \eta_{\tilde{f}}\nabla\theta_{\tilde{f}}$
39:         $\nabla\theta_{d_1} \leftarrow \partial\mathcal{L}_{d_1}/\partial\theta_{d_1}$; $\theta_{d_1} \leftarrow \theta_{d_1} - \eta_{d_1}\nabla\theta_{d_1}$
40:         $\nabla\theta_{\tilde{f}^{-1}} \leftarrow \partial\mathcal{L}_{\tilde{f}^{-1}}/\partial\theta_{\tilde{f}^{-1}}$; $\theta_{\tilde{f}^{-1}} \leftarrow \theta_{\tilde{f}^{-1}} - \eta_{\tilde{f}^{-1}}\nabla\theta_{\tilde{f}^{-1}}$
41:         $\nabla\theta_{d_2} \leftarrow \partial\mathcal{L}_{d_2}/\partial\theta_{d_2}$; $\theta_{d_2} \leftarrow \theta_{d_2} - \eta_{d_2}\nabla\theta_{d_2}$
42:
43:         $\hat{X} \leftarrow \tilde{f}^{-1}(Z_s)$                  ▷ Reconstructed private features
44:         $\hat{Y} \leftarrow \tilde{h}(Z_t)$                       ▷ Inferred private labels
```

Fig. 20. SDAR against U-shaped split learning. $\ell(\cdot,\cdot)$ is the loss function of the original SL task; s, t are the split levels; η is the learning rate of the original SL task; $\eta_{\tilde{h}}, \eta_{\tilde{f}}, \eta_{\tilde{f}^{-1}}, \eta_{d_1}, \eta_{d_2}$ are learning rates for the server's models $\tilde{h}, \tilde{f}, \tilde{f}^{-1}, d_1, d_2$ respectively. p is the label flipping probability. $\mathcal{Y}$ is the set of all possible labels.

in total. We train the model H with split learning protocol for 20000 iterations (i.e., batches). For all attacks, on all datasets, with both models, at all considered split levels, and under both vanilla and U-shaped settings, we run five trials for statistical significance.

C. Implementation details of SDAR

Models. Under our firstly considered threat model where the server knows the model architecture of $H = g \circ f$, the server uses the same model architecture for its simulator $\tilde{f}$ as the client's model f, but with random initialization. When experimenting with the relaxed assumption that the server



(a) ResNet-20 (b) PlainNet-20

Fig. 21. Model architectures and split configurations used in the experiments. Each box marked as "Conv2D" represents a convolutional layer with the specified kernel size, number of filters, and stride (1 if absent), followed by batch normalization and ReLU.

does not know the architecture of the client model (e.g. in Table IV), we use ResNet-20 for H while making the server to use a plain convolutional network as a simulator. This model structure is obtained by removing the skip connections in the client's residual network. For all experiments, the server's decoder $\tilde{f}^{-1}$ is a deep convolutional network, where each building block of $\tilde{f}$ corresponds to a deconvolution block (deconvolutional layer, followed by batch normalization and ReLU) in reverse order. Note that we replace deconvolutional layers with strides 2 with a combination of an upsampling layer and a convolutional layer, to reduce the checkerboard effects [36]. For the discriminator d_1 which distinguishes intermediate representations, we use a deep convolutional network with up to 256 filters, where each convolutional layer is followed by LeakyReLU and batch normalization except the last layer. The discriminator d_2 that distinguishes real and fake images follows a rather standard architecture, consists of convolutional layers, batch normalization and LeakyReLU. In the vanilla setting where labels are used as part of the input to the decoder and discriminators, we first transform the labels into embeddings of 50 units, and then further transform the embeddings by a learnable fully connected layer, before concatenating with the input to the decoder and discriminators. Note that we use the same model structures for the decoder and discriminators for both cases of the target model (i.e. ResNet-20 and PlainNet-20). We list the detailed model structures for the attacker's models $\tilde{f}^{-1}, d_1, d_2$ under vanilla SL in Table XIV.

Hyperparameters. As simulators to f and h respectively, we choose the learning rate of $\tilde{f}$ and $\tilde{h}$ the same as η, and we always keep the learning rate of the decoder half of that of the simulator, i.e., $2\eta_{\tilde{f}^{-1}} = \eta_{\tilde{f}} = \eta_{\tilde{h}} = \eta$. We choose the regularization factors λ_1, λ_2 in a way that the regularization term will not dominate the total loss, but still non-negligible. For the adversarial regularization of $\tilde{f}$, this means that λ_1 should be set s.t. the regularization term $\lambda_1\ell_{\text{BCE}}(d_1(Z'_s), 1)$ should not surpass the classification loss

TABLE XIII. HYPERPARAMETERS USED IN SDAR

Setting	Dataset	Model	λ_1	λ_2	p
Vanilla	All datasets	Both models	0.02	0.00001	NA
U Shaped	CIFAR-10	ResNet-20	0.02	0.00001	0.2
U Shaped	CIFAR-100	ResNet-20	0.04	0.00001	0.2
U Shaped	Tiny ImageNet	ResNet-20	0.04	0.00001	0.2
U Shaped	STL-10	ResNet-20	0.04	0.00001	0.2
U Shaped	CIFAR-10	PlainNet-20	0.04	0.00001	0.1
U Shaped	CIFAR-100	PlainNet-20	0.04	0.00001	0.1
U Shaped	Tiny ImageNet	PlainNet-20	0.04	0.00001	0.4
U Shaped	STL-10	PlainNet-20	0.04	0.00001	0.4

$\ell(\hat{Y}', \tilde{Y}')$. We expect that, when the training stabilizes, the discriminator loss $\ell_{\text{BCE}}(d_1(Z_s'), 1)$ should be around the binary cross entropy loss of random guess, i.e., $\log(2) \approx 0.6931$, to ensure that the simulator and the discriminator can be continually improved via adversarial training. Therefore, to ensure that the regularization term $\lambda_1 \ell_{\text{BCE}}(d_1(Z_s'), 1)$ does not dominate the total loss, λ_1 is chosen to at the order of magnitude of 10^{-2}, such that the regularization term will have the order of magnitude of 10^{-3}, which does not dominate the training classification loss while being non-negligible. For the adversarial regularization of $\tilde{f}^{-1}$, we choose λ_2 to be much smaller, at 10^{-5}, as the reconstruction loss $\ell_{\text{MSE}}(X', \hat{X}')$ is expected to be much smaller than the classification loss, and the discriminator loss $\ell_{\text{BCE}}(d_2(\hat{X}), 1)$ should be around $\log(2)$ as well. After determining the regularization factors λ_1, λ_2, the learning rates for the discriminators are chosen as $\eta_{d_1} = \lambda_1 \eta_{\tilde{f}}$ and $\eta_{d_2} = \lambda_2 \eta_{\tilde{f}}$, to ensure that the step size to minimize generator loss (as a regularization term) is on par with that of the discriminator loss.

Guided by the above considerations, we set $\lambda_1 = 0.02$ and $\lambda_2 = 10^{-5}$ for all experiments in vanilla SL. For U-shaped SL, the attacks are more sensitive to the choices of λ_1 and the flip probability p, and we test with different configurations for different datasets and models and choose the best-performing combination. In particular, we test $\lambda_1 \in \{0.2, 0.4\}$ and $p \in \{0.1, 0.2, 0.4\}$. Note that such hyperparameter search is feasible in practice, as the server can always run the SDAR attack multiple times on the observed representations with different configurations and collect all reconstructed images. The hyperparameters used in the experiments are shown in Table XIII.

Training instability. Due to the inherent instability of GANs, we observe, though very rarely, cases where SDAR fails to converge, resulting in a high reconstruction loss. In such cases, we reinitialize the attacker's models and restart the training process. We only observe such cases in the U-shaped setting, and the reinitialization process is only needed in very few (fewer than 5) trials among our extensive experiments. Note that such process is feasible in practice, as the server can always run the SDAR attack multiple times on the observed representations and collect all reconstructed images.

D. Implementation details of baseline methods

PCAT. We implement PCAT [13] with the exact same model architectures for simulator and decoder as SDAR for a fair comparison. We also give the server in PCAT the same auxiliary dataset as SDAR. Compared to naïve SDA, PCAT uses two unique techniques to improve its performance. First, in vanilla SL where the server knows the labels of the private examples, PCAT aligns these labels by sampling auxiliary examples with the same labels as the received private examples. Second, in both vanilla and U-shaped SL, PCAT introduces a delay in attacking, where the server does not train its simulator and decoder until a certain number of iterations have been executed, to avoid the noisy early-stage training process disturbing the attacks. We implement both techniques, and choose a delay period of 100 iterations, as used in the original paper [13]. Note that by the time our research was conducted, the code of PCAT was not publicly available, so we implement PCAT from scratch based on the description in the original paper [13].

UnSplit. UnSplit [9] infers the private features X by minimizing $\ell_{\text{MSE}}(\tilde{f}(\hat{X}), Z_s)$ via alternating optimization of $\tilde{f}$ and $\hat{X}$, where $\tilde{f}$ is a surrogate model of the same architecture as f but randomly initialized, and $\hat{X}$ is initialized as tensors filled with 0.5. We follow the implementation of the original paper [9] and use the same configuration for the alternating optimization algorithm. We use Adam optimizer with learning rate of 0.001 for both $\tilde{f}$ and $\hat{X}$. The alternating optimization process consists of 1000 rounds, each having 100 $\tilde{f}$ optimization steps and 100 $\hat{X}$ optimization steps. Following [9], total variation is used to regularize the optimization of $\hat{X}$. In each training iteration of SL, UnSplit infers the private training examples X by running the iterative alternating optimization procedure. However, due to the extremely high computational cost of the iterative optimization process, we are unable to run the optimization algorithm for every training iteration, but only attack the last batch of training examples.

FSHA. We implement FSHA [38] with the same model architectures for the encoder, decoder and simulator discriminator as SDAR for a fair comparison. As FSHA utilizes Wassertein GAN loss with gradiant penalty (WGAN-GP) [18] for training, which is not compatible with batch normalization layers in the discriminator, we remove such layers in the discriminator model. In FSHA, the client's model f is hijacked by the server to be updated to simulate the behaviors of the server's own encoder $\tilde{f}$. Following the original paper, we use learning rate 0.00001 for the training of f as a generator and use learning rate 0.0001 for the discriminators with gradient penalty coefficient of 500. The encoder and decoder are trained in an autoencoder fashion [24] with learning rate 0.00001.

APPENDIX C
FULL EXPERIMENTAL RESULTS

In Section V, we only present experimental results on a limited number of datasets due to limited space. The full results of our experiments on all four datasets with both models can be found in the full version of this paper [64]. In addition, we also report the attack MSE and label inference accuracy for U-shaped SL of SDAR over training iterations across all configurations in the full paper [64].

TABLE XIV. ATTACKER'S MODEL STRUCTURES FOR $\tilde{f}^{-1}, d_1, d_2$ IN VANILLA SL*

	Split Level 4	Split Level 5	Split Level 6	Split Level 7

$\tilde{f}^{-1}$	y = Embedding(num_classes, 50)(y)
	y = Dense(x.shape[0] * x.shape[1])(y)
	y = Reshape((x.shape[0], x.shape[1])(y)
	x = Concatenate()([x,y])
	x = UpSampling2D((2,2))(x)
	x = Conv(32, 3, (1,1))(x)
	x = ReLU()(BatchNorm()(x))
	x = ConvTranspose(16, 3, (1,1))(x)
	x = ReLU()(BatchNorm()(x))
	x = ConvTranspose(16, 3, (1,1))(x)
	x = ReLU()(BatchNorm()(x))
	x = ConvTranspose(16, 3, (1,1))(x)
	x = ReLU()(BatchNorm()(x))
	x = Conv(3, 3, (1, 1))(x)
	x = Sigmoid()(x)
		y = Embedding(num_classes, 50)(y)
		y = Dense(x.shape[0] * x.shape[1])(y)
		y = Reshape((x.shape[0], x.shape[1])(y)
		x = Concatenate()([x,y])
		x = ConvTranspose(32, 3, (1,1))(x)
		x = ReLU()(BatchNorm()(x))
		x = UpSampling2D((2,2))(x)
		x = Conv(32, 3, (1,1))(x)
		x = ReLU()(BatchNorm()(x))
		x = ConvTranspose(16, 3, (1,1))(x)
		x = ReLU()(BatchNorm()(x))
		x = ConvTranspose(16, 3, (1,1))(x)
		x = ReLU()(BatchNorm()(x))
		x = ConvTranspose(16, 3, (1,1))(x)
		x = ReLU()(BatchNorm()(x))
		x = Conv(3, 3, (1, 1))(x)
		x = Sigmoid()(x)
			y = Embedding(num_classes, 50)(y)
			y = Dense(x.shape[0] * x.shape[1])(y)
			y = Reshape((x.shape[0], x.shape[1])(y)
			x = Concatenate()([x,y])
			x = ConvTranspose(32, 3, (1,1))(x)
			x = ReLU()(BatchNorm()(x))
			x = ConvTranspose(32, 3, (1,1))(x)
			x = ReLU()(BatchNorm()(x))
			x = UpSampling2D((2,2))(x)
			x = Conv(32, 3, (1,1))(x)
			x = ReLU()(BatchNorm()(x))
			x = ConvTranspose(16, 3, (1,1))(x)
			x = ReLU()(BatchNorm()(x))
			x = ConvTranspose(16, 3, (1,1))(x)
			x = ReLU()(BatchNorm()(x))
			x = ConvTranspose(16, 3, (1,1))(x)
			x = ReLU()(BatchNorm()(x))
			x = Conv(3, 3, (1, 1))(x)
			x = Sigmoid()(x)
				y = Embedding(num_classes, 50)(y)
				y = Dense(x.shape[0] * x.shape[1])(y)
				y = Reshape((x.shape[0], x.shape[1])(y)
				x = Concatenate()([x,y])
				x = UpSampling2D((2,2))(x)
				x = Conv(64, 3, (1,1))(x)
				x = ReLU()(BatchNorm()(x))
				x = ConvTranspose(32, 3, (1,1))(x)
				x = ReLU()(BatchNorm()(x))
				x = ConvTranspose(32, 3, (1,1))(x)
				x = ReLU()(BatchNorm()(x))
				x = UpSampling2D((2,2))(x)
				x = Conv(32, 3, (1,1))(x)
				x = ReLU()(BatchNorm()(x))
				x = ConvTranspose(16, 3, (1,1))(x)
				x = ReLU()(BatchNorm()(x))
				x = ConvTranspose(16, 3, (1,1))(x)
				x = ReLU()(BatchNorm()(x))
				x = ConvTranspose(16, 3, (1,1))(x)
				x = ReLU()(BatchNorm()(x))
				x = Conv(3, 3, (1, 1))(x)
				x = Sigmoid()(x)

d_1	y = Embedding(num_classes, 50)(y)
	y = Dense(x.shape[0] * x.shape[1])(y)
	y = Reshape((x.shape[0], x.shape[1])(y)
	x = Concatenate()([x,y])
	x = Conv(64, 3, (1,1))(x)
	x = LeakyReLU()(x)
	x = Conv(128, 3, (2,2))(x)
	x = LeakyReLU()(BatchNorm()(x))
	x = Conv(256, 3, (1,1))(x)
	x = LeakyReLU()(BatchNorm()(x))
	x = Conv(256, 3, (1,1))(x)
	x = LeakyReLU()(BatchNorm()(x))
	x = Conv(256, 3, (1,1))(x)
	x = LeakyReLU()(BatchNorm()(x))
	x = Conv(256, 3, (2,2))(x)
	x = Flatten()(x)
	x = Dropout(0.4)(x)
	x = Dense(1)(x)
		y = Embedding(num_classes, 50)(y)
		y = Dense(x.shape[0] * x.shape[1])(y)
		y = Reshape((x.shape[0], x.shape[1])(y)
		x = Concatenate()([x,y])
		x = Conv(64, 3, (1,1))(x)
		x = LeakyReLU()(x)
		x = Conv(128, 3, (2,2))(x)
		x = LeakyReLU()(BatchNorm()(x))
		x = Conv(256, 3, (1,1))(x)
		x = LeakyReLU()(BatchNorm()(x))
		x = Conv(256, 3, (1,1))(x)
		x = LeakyReLU()(BatchNorm()(x))
		x = Conv(256, 3, (1,1))(x)
		x = LeakyReLU()(BatchNorm()(x))
		x = Conv(256, 3, (2,2))(x)
		x = Flatten()(x)
		x = Dropout(0.4)(x)
		x = Dense(1)(x)
			y = Embedding(num_classes, 50)(y)
			y = Dense(x.shape[0] * x.shape[1])(y)
			y = Reshape((x.shape[0], x.shape[1])(y)
			x = Concatenate()([x,y])
			x = Conv(64, 3, (1,1))(x)
			x = LeakyReLU()(x)
			x = Conv(128, 3, (2,2))(x)
			x = LeakyReLU()(BatchNorm()(x))
			x = Conv(256, 3, (1,1))(x)
			x = LeakyReLU()(BatchNorm()(x))
			x = Conv(256, 3, (1,1))(x)
			x = LeakyReLU()(BatchNorm()(x))
			x = Conv(256, 3, (1,1))(x)
			x = LeakyReLU()(BatchNorm()(x))
			x = Conv(256, 3, (2,2))(x)
			x = Flatten()(x)
			x = Dropout(0.4)(x)
			x = Dense(1)(x)
				y = Embedding(num_classes, 50)(y)
				y = Dense(x.shape[0] * x.shape[1])(y)
				y = Reshape((x.shape[0], x.shape[1])(y)
				x = Concatenate()([x,y])
				x = Conv(128, 3, (1,1))(x)
				x = LeakyReLU()(x)
				x = Conv(256, 3, (1,1))(x)
				x = LeakyReLU()(BatchNorm()(x))
				x = Conv(256, 3, (1,1))(x)
				x = LeakyReLU()(BatchNorm()(x))
				x = Conv(256, 3, (1,1))(x)
				x = LeakyReLU()(BatchNorm()(x))
				x = Conv(256, 3, (2,2))(x)
				x = Flatten()(x)
				x = Dropout(0.4)(x)
				x = Dense(1)(x)

d_2	y = Embedding(num_classes, 50)(y)
	y = Dense(x.shape[0] * x.shape[1])(y)
	y = Reshape((x.shape[0], x.shape[1])(y)
	x = Concatenate()([x,y])
	x = Conv(64, 3, (1,1))(x)
	x = LeakyReLU()(x)
	x = Conv(128, 3, (2,2))(x)
	x = LeakyReLU()(BatchNorm()(x))
	x = Conv(128, 3, (2,2))(x)
	x = LeakyReLU()(BatchNorm()(x))
	x = Conv(256, 3, (2,2))(x)
	x = LeakyReLU()(x)
	x = Flatten()(x)
	x = Dropout(0.4)(x)
	x = Dense(1)(x)
		y = Embedding(num_classes, 50)(y)
		y = Dense(x.shape[0] * x.shape[1])(y)
		y = Reshape((x.shape[0], x.shape[1])(y)
		x = Concatenate()([x,y])
		x = Conv(64, 3, (1,1))(x)
		x = LeakyReLU()(x)
		x = Conv(128, 3, (2,2))(x)
		x = LeakyReLU()(BatchNorm()(x))
		x = Conv(128, 3, (2,2))(x)
		x = LeakyReLU()(BatchNorm()(x))
		x = Conv(256, 3, (2,2))(x)
		x = LeakyReLU()(x)
		x = Flatten()(x)
		x = Dropout(0.4)(x)
		x = Dense(1)(x)
			y = Embedding(num_classes, 50)(y)
			y = Dense(x.shape[0] * x.shape[1])(y)
			y = Reshape((x.shape[0], x.shape[1])(y)
			x = Concatenate()([x,y])
			x = Conv(64, 3, (1,1))(x)
			x = LeakyReLU()(x)
			x = Conv(128, 3, (2,2))(x)
			x = LeakyReLU()(BatchNorm()(x))
			x = Conv(128, 3, (2,2))(x)
			x = LeakyReLU()(BatchNorm()(x))
			x = Conv(256, 3, (2,2))(x)
			x = LeakyReLU()(x)
			x = Flatten()(x)
			x = Dropout(0.4)(x)
			x = Dense(1)(x)
				y = Embedding(num_classes, 50)(y)
				y = Dense(x.shape[0] * x.shape[1])(y)
				y = Reshape((x.shape[0], x.shape[1])(y)
				x = Concatenate()([x,y])
				x = Conv(64, 3, (1,1))(x)
				x = LeakyReLU()(x)
				x = Conv(128, 3, (2,2))(x)
				x = LeakyReLU()(BatchNorm()(x))
				x = Conv(128, 3, (2,2))(x)
				x = LeakyReLU()(BatchNorm()(x))
				x = Conv(256, 3, (2,2))(x)
				x = LeakyReLU()(x)
				x = Flatten()(x)
				x = Dropout(0.4)(x)
				x = Dense(1)(x)

*Under U-shaped SL where the server no longer knows the labels of the private data, the model structures follow this table after the removal of label embeddings and their concatenation with the input.