Sniper Backdoor: Single Client Targeted Backdoor Attack in Federated Learning

Gorka Abad[†‡], Servio Paguada[†‡], Oğuzhan Ersoy[†], Stjepan Picek[†], Víctor Julio Ramírez-Durán[‡] and Aitor Urbieta[‡]
[†] AISyLab, Radboud University, Nijmegen, The Netherlands
abad.gorka@ru.nl firstname.lastname@ru.nl
[‡] Ikerlan Research Centre, Arrasate-Mondragón, Spain
{jvramirez,aurbieta}@ikerlan.es

***Abstract*—Federated Learning (FL) enables collaborative training of Deep Learning (DL) models where the data is retained locally. Like DL, FL has severe security weaknesses that the attackers can exploit, e.g., model inversion and backdoor attacks. Model inversion attacks reconstruct the data from the training datasets, whereas backdoors misclassify only classes containing specific properties, e.g., a pixel pattern. Backdoors are prominent in FL and aim to poison every client model, while model inversion attacks can target even a single client.**

This paper introduces a novel technique to allow backdoor attacks to be client-targeted, compromising a single client while the rest remain unchanged. The attack takes advantage of state-of-the-art model inversion and backdoor attacks. Precisely, we leverage a Generative Adversarial Network to perform the model inversion. Afterward, we shadow-train the FL network, in which, using a Siamese Neural Network, we can identify, target, and backdoor the victim's model. Our attack has been validated using the MNIST, F-MNIST, EMNIST, and CIFAR-100 datasets under different settings—achieving up to 99% accuracy on both source (clean) and target (backdoor) classes and against state-of-the-art defenses, e.g., Neural Cleanse, opening a novel threat model to be considered in the future.

***Index Terms*—Federated Learning, Deep Learning, Backdoor attack**

I. Introduction

Deep learning (DL) achieves state-of-the-art performance in various machine learning tasks, e.g., computer vision [48], speech recognition [20], and natural language processing [7]. Unfortunately, DL has severe security and privacy flaws that have been exploited in recent years, e.g., backdoors [21] and inference attacks [28]. Therefore, research has also focused on creating secure DL algorithms that prevent leaking data or causing misbehavior [17].

Contrary to centralized DL algorithms, where data and the model are stored in a single point and trained, in 2016, Google developed a privacy-preserving decentralized and collaborative training approach, i.e., *Federated Learning* (FL) [26]. FL is composed of a set of clients and an aggregator (server), where clients store their data and train the same DL model locally for a couple of epochs. Then, the model parameters are shared with the aggregator, which merges them, joining the properties of heterogeneous datasets without accessing them.

Though their decentralized structures, FL protocols are also prone to security and privacy attacks [1], [27]. The most popular attacks in FL are the *backdoor* and *inference* attacks. While backdoor attacks focus on modifying the training set to cause behavior at inference time [21], [22], inference attacks extract private information from the DL model [9], [38]. More precisely, backdoor attacks focus on altering the training set by including specific triggers in the input space, which cause misbehavior of the DL model only under the presence of the trigger. If it is not present, the DL model will behave normally. Backdoor attacks have been adapted to FL, where modifying a single client dataset can cause the joined model to misbehave, poisoning each client in the network [3], [39], [43].

Note that the existing backdoor attacks on FL cause all clients to end up with a backdoored model. However, there could be scenarios where the server targets a single client or a subset of the clients, not all. For example, imagine a scenario where competitive banks want to train a model for credit scores jointly. Among those banks, one is targeted by the attacker, who may support or remain neutral to the other banks and thus does not want to backdoor them. This single-target scenario is unexplored and leads to the following question: *Is it possible to launch a backdoor attack, where only targeted (victim) clients get a backdoored model, whereas the remaining (non-victim) clients get a clean model?*

A. Our Contributions

In this work, we positively answer this question by providing a backdoor attack that can be client-targeted, only poisoning a subset of clients while the models of the other clients remain unaltered. Our attack combines state-of-the-art inference and backdoor attacks to recover datasets from clients, identify the target clients, and inject the backdoor. Precisely, to achieve a targeted backdoor, we redesign the existing inference attacks by making them client-specific, allowing the generation of client-like datasets. Furthermore, we create a shadow network in FL, which, combined with an SNN, allows identifying the victim among all the clients. This results in a client-targeted backdoor, where the victim gets a poisoned model while the rest of the clients get a clean one.

Our main contributions are:

The Horizon Europe, Spanish CDTI, and ELKARTEK programs partially fund this research. Grant agreements 101021911 (IDUNN), CER-20191012 (EGIDA), and KK-2021/00091 (REMEDY), respectively.

- We present, to the best of our knowledge, the first client-targeted backdoor attacks for FL settings, which first identifies the victim, then injects the backdoor only for the victim, but not the other clients.
- We adapt and train a Siamese Neural Network (SNN) for their use with triplet loss, which enables identifying anonymous clients in the FL network with up to 92% accuracy.
- We extend and analyze the capabilities of backdoor attacks, injecting the trigger at near convergence and focusing on a target client, achieving up to 99% accuracy in clean and backdoor test sets and confirming the viability of our attack.
- We apply state-of-the-art defense mechanisms to our protocol. Since most of the defenses are on the server side or do not fit our setup, we modified them to fit our attack scenario, e.g., we adapted *Trojanzoo* [30], a backdoor testing framework, to utilize our attack. More precisely, we consider Neural Cleanse [42] and smoothing [51].

To improve reproducibility, we share the source code of our work.[1]

B. Related Work

FL has gained attention as a privacy-driven alternative to centralized learning—granting the ability to train a DL algorithm without sharing the data and splitting the computational power [37]. However, it has been shown that FL is also vulnerable to attacks that make the DL models misbehave at inference time, i.e., poisoning or backdoor attacks [3], [21], [34], or causing privacy leakages, i.e., inference attacks [28]. Comprehensive studies about the attacks on FL are given in [1], [27]. In this work, we focus on inference and backdoor attacks.

Inference attacks aim to learn private information about the training process, which can be used to check if a data sample is used in the training set (membership inference) [9], [28], [38] or to reconstruct the training set (model inversion) [14]. A backdoor is a particular type of poisoning attack whose goal is to misclassify a sample just under a presence of a property or a characteristic. Backdoor attacks have been widely mitigated for different machine learning tasks, e.g., audio or images [8], [25], [41], [42]. In the FL learning setting, the backdoor attack can be applied by the clients or the aggregator [3], [39], [43], [52]. The backdoor attack in the FL setting is more challenging than in the centralized setting since each client has a weighted effect on the global model. To overcome this issue, attackers can either amplify the weights of backdoored model [3] or distribute the backdoor to several clients [52]. The existing backdoor attacks on FL settings degrade the global model for all clients, whereas, to the best of our knowledge, there is no client-targeted backdoor attack.

In addition to the attacks, several defense protocols have been proposed to mitigate them. In general, the defenses are based on either dataset inspection [8], [41] or model inspection [25], [42]. Moreover, there are defense mechanisms for FL setting [2], [4], [15], [16], [51], which are mainly based on the addition of noise or an alternative reweighting performed by an honest aggregator. The details of the defense mechanisms are given in Section VI.

II. Background

This section starts with an overview of deep learning and federated learning. Afterward, we discuss backdoor attacks, inference attacks, Generative Adversarial Networks, and SNN.

A. Deep Learning & Federated Learning

a) Deep Learning: DL algorithms are parameterized functions $\mathbb{F}_\theta$ that maps an input $\mathbf{x} \in \mathbb{R}^a$ to an output $y \in \mathbb{R}^b$. In the image domain, the dataset is constructed from a collection of images and their labels $\{\mathbf{x}, y\}^n$ of size n where $\mathbf{x}$ is a vector of pixel values and y is a vector of probabilities of belonging to a class $c \in C$. The parameters, θ, are iteratively set by finding the optimal value for which $\mathbb{F}_\theta(\mathbf{x}) = y$, achieved by training. During training, large sets of data are provided, and the distance from the predicted output $\mathbb{F}_\theta(\mathbf{x})$ to the ground truth value y is measured, penalizing predictions that are far away, using a loss function $\mathcal{L}$. Therefore, the optimal values of the parameters θ' are given by the following equation:

$$\theta' = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{n} \mathcal{L}(\mathbb{F}_\theta(\{\mathbf{x}_i, y_i\})). \quad (1)$$

b) Federated Learning: FL is a privacy-driven decentralized scheme for collaborative training of ML models. It was introduced by Google, where they proposed creating a network of clients who own distinct datasets to train a global model without directly sharing their datasets [26]. The network is composed of a server (aggregator) and N clients. Every participant of the network, upon consensus, decides to train the same model W under the same conditions, e.g., learning rate (*LR*) and the number of epochs. After local training, following Eq. (1), clients upload their model updates u_t (the differences with the previous epoch) to the aggregator, who joins them by averaging, and sends the new model W_{t+1} back to each client. The FL procedure is repeated for t epochs until convergence is reached:

$$W_{t+1} \leftarrow W_t + \frac{1}{N}\sum_{i=1}^{N} u_{t+1}^i.$$

B. Backdoor Attacks

Backdoor attacks compromise DL networks during training, causing misbehavior at inference time by injecting poisoned samples into the training set. A poisoned sample has a trigger embedded in the clean data sample that assigns a target label $\hat{y} \in C$ different from the ground truth label $\hat{y} \neq y$. In source class-targeted backdoors, only samples of a given source class are poisoned and assigned the target class, whereas in one-to-all attacks, all source classes are poisoned and assigned the target class. Furthermore, attacks are classified as targeted or untargeted if the attacker targets a specific label or creates an uncontrolled misclassification [27].

[1] https://github.com/GorkaAbad/Sniper-Backdoor

In the image domain, the backdoor trigger is usually a modified pixel or group of pixels, e.g., a pixel square, with a given size and position, e.g., left-top corner or center. The percentage of poisoned samples in the training set is controlled by $\epsilon = \frac{m}{n+m}$ where $m \ll n$ and the poisoned set is shown by $\hat{D}_{train}$. Here, a small value ϵ usually implies that it is more challenging to include backdoor behavior, but it causes a more stealthy attack. During the training procedure with $\hat{D}_{train}$, the backdoor effect is injected into the DL algorithm given by the loss function taking into account the backdoor accuracy:

$$\theta' = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{n} \mathcal{L}(\mathbb{F}_\theta(\{\mathbf{x}_i, y_i\})) + \sum_{j=1}^{m} \mathcal{L}(\mathbb{F}_\theta(\{\hat{\mathbf{x}}_j, \hat{y}_j\})).$$

In FL, when a backdoored model is submitted to the server, the poison weights vanish due to aggregation since a single vector of outline values gets averaged with the rest of the clients, thus making the poisoned weights less relevant. However, the vanishing effect can be overcome by upscaling the weights of the model [3], [39], [43].

C. Inference & Model Inversion Attacks

Inference attacks measure information leakage through a DL model about its training data. There are different ways in which the attacker obtains private information; for example, by observing the input and outcome of a DL model or by observing the inner computation if the attacker has access to the model [28]. In FL scenarios, the attacker could be either the aggregator who has access to the individual updates over the epochs or the clients who have access to the joined model and can control the parameters of the model to update. Depending on the attacker's knowledge and capabilities, the attacker can perform a passive attack only by observing the computations or an active attack that modifies the model's parameters. Similarly, model inversion attacks exploit confidence values obtained during the predictions for reconstructing data from the training dataset of a DL model [14]. Furthermore, in FL, Generative Adversarial Networks (GANs) enable the recovery of user-specific data records [9], [38].

D. Generative Adversarial Networks (GANs)

Deep learning aims to create rich models representing the probability distribution of different data. In contrast, deep-generative models aim to generate data samples with a distribution similar to the one provided. So far, deep generative models leverage the max-likelihood estimation for such a task. However, approximating the probability computation on the max-likelihood estimation is hard. Goodfellow et al. developed a framework for estimating generative models via an adversarial process [19]:

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_z \exp(\sigma^{-1}(D(G(\mathbf{z})))),$$

where J is the cost function, σ is the logistic sigmoid function, G is the generator, and D is the discriminator. G and D are the two parties involved in Generative Adversarial Networks, which follow an estimation process based on simulation training via a *zero-sum game*, also called *minimax* game:

$$\theta^{(G)'} = \underset{\theta^{(G)}}{\operatorname{argmin}} \underset{\theta^{(D)}}{\max} V(\theta^{(D)}, \theta^{(G)}).$$

G takes noise $\mathbf{z} \sim p_Z$ samples from some distribution and creates actual data samples, while D distinguishes fake samples from real ones. Both trains simultaneously until they achieve the Nash equilibrium, where G can generate real-enough data samples that D cannot differentiate. Thus, the distribution of the generated fake samples $p_{G(Z)}$ converges towards the distribution of real data samples. GANs have been widely used in different domains, e.g., image creation [32], NLP [7]. Concerning security, GANs have also played an important role, performing inference attacks [53] or generating adversarial examples [49].

E. Siamese Neural Networks (SNNs)

SNN is a type of architecture constructed by two identical networks (having the same parameters and structure) to find similarities in input by comparing the latent space of their feature vectors [6]. Since SNN involves pairwise data for training, the loss function has to optimize the model to minimize the distance, e.g., Euclidean distance, between similar inputs and maximize it between different inputs. Triplet loss [33] improves learning embeddings of inputs and is usually used to improve SNN performance during training. The same class inputs are close together (in the embedding space) while different class inputs are well separated.[2] Triplet networks gained popularity with the development of FaceNet [33]. Since then, triplet networks have been used in diverse domains, e.g., side-channel analysis [47] or learning image similarity [44].

Triplet loss ($\mathcal{L}$) is formally defined as a function that maximizes the distance between the embedding space with respect to the anchor samples (A), positive (P), and negative (N) by some margin α. Since A and P samples have the same label, triplet loss optimizes the model so that the distance between A and N samples is more significant than between A and P:

$$\mathcal{L}_{A,P,N} = \max(||f(A) - f(P)||^2 - ||f(A) - f(N)||^2 + \alpha, 0).$$

During the minimization of $\mathcal{L}$, $||f(A) - f(P)||^2$ is pushed towards 0, while $||f(A) - f(N)||^2$ is larger than $||f(A) - f(P)||^2 + \alpha$. The loss can also be represented in different forms according to the three triplet categories: (i) *Easy triplets* where the negative sample is sufficiently distant from the anchor compared to the positive sample from the anchor, (ii) *Hard triplets* where the distance between the negative sample and the anchor is less than the positive to the anchor, and (iii) *Semi-Hard triplets* where the distance between the negative and the anchor is greater than the positive to the anchor but is not larger than a margin α, i.e., $||f(A) - f(P)||^2 < ||f(A) - f(N)||^2 < ||f(A) - f(P)||^2 + \alpha$.

[2] Similar to support vector machines [11], where some margin separates different classes' samples.

Triplet samples are constructed via online triplet mining, where the anchor, positive, and negative samples are computed on the fly for every batch of inputs. The correct selection of triplets will influence the quality of the model. Therefore, in our research and following the suggestions of the original paper [33], we use the *semi-hard* triplets.

III. Threat Model

This section discusses the scenario we follow and the assumptions we make. Afterward, we provide details about the adversary's objectives and capabilities.

A. Our Scenario and Assumptions

Following the training procedure of FL, we assume that a group of clients agree on a common learning objective and collaboratively train a shared model. Clients own IID (*independent and identically distributed*) or Non-IID distributed data. To be comprehensive, we considered both data distributions in our attack, and as a use case, the shared model is aimed at classifying images. Regarding the server, we assume it is malicious, aiming to inject a backdoor into the victim's model, also referred to as the target client, while the rest receive a clean (non-backdoored) model. The malicious server could either passively analyze periodic client updates or modify the global model parameters. The presented scenario is commonly adopted in most of the existing attacks [5], [9], [38].

Additionally, aiming to consider a more realistic scenario, we assume that clients simultaneously share their models anonymously with the aggregator, i.e., their identity cannot be matched to the model submitted. For example, clients could leverage Tor [13] for anonymization, as considered in recent works [9], [38].

B. Adversarial Objectives & Capabilities

We consider a set of clients who wishes to train a DL algorithm for finding the optimal parameters, $\mathbb{F}_{\theta'}$, each of them using a non-colluding dataset D_{train}. Each client shares $\mathbb{F}_{\theta}$ with the aggregator, which returns the joined parameters after each epoch. Before achieving convergence of the joined model, the attacker injects and returns a set of malicious parameters to the victim, including backdoor behavior $\hat{\theta} \leftarrow \theta'$. The attacker has access to the training function, F_{θ}, and has to carefully adjust the training process to determine the best values for $\hat{\theta}$. The attacker poisons the holdout training dataset $\hat{D}_{train}$ that includes triggers in the samples and obtains the poisoned parameters $\hat{\theta} \leftarrow \mathbb{F}_{\theta}(\hat{D}_{train})$. For a successful attack, the attacker should achieve high accuracy on the main task, i.e., on the client's validation set and the backdoor task.

Note that the backdoor creation function is independent of our pipeline. Depending on the defensive mechanisms applied by the clients, the attacker could use a simple approach (BadNets [21]) or a more complex input-aware dynamic backdoor [29]. To show the adaptability of our attack, we consider both types of backdoor attacks.

To evaluate the performance of the attack, we utilize four metrics:

1) **Clean accuracy:** measures the overall accuracy of the backdoor model over a clean test set.
2) **Clean accuracy degradation:** measures the drop in clean accuracy of the clean model and the backdoor model.
3) **Source class accuracy:** measures the accuracy of the source class in a clean test set.
4) **Target class accuracy:** measures the target class accuracy over a fully poisoned test set, also known as Attack Success Rate (ASR).

IV. Proposed Client-wise Targeted Backdoor

In this section, we start with an overview of the proposed attack. Afterward, we discuss each component of the attack.

A. Attack Overview

Our attack injects a backdoor into the model of a target client (victim). This section provides an overview of our framework; see Figure 1 for guidance.

① The clients and the server initialize a common model. Clients locally train their models and submit them for aggregation with the server. This process is iteratively repeated until convergence is met. ② During training, and at every epoch t, the malicious server (attacker) collects and saves clients' models (u_t^N) for later use as a historical record. ③ For each client, the attacker initializes the GAN discriminator with a model's weights chosen from the historical record. The discriminator is then trained until it can generate real enough samples. ④ The attacker uses each trained GAN to create a dataset for every client. ⑤ The attacker replicates the FL network, named the shadow network. The clients of the shadow network, i.e., shadow clients, locally train their models using D_{train}. As in the second step, the attacker collects a historical record of the shadow models submitted. ⑥ Using the historical record of shadow models, the attacker trains an SNN. ⑦ After training, the SNN can precisely identify models of clients at different epochs, enabling the victim's identification. ⑧ Lastly, the attacker injects a backdoor into the global model and shares it with the victim.

B. Training the Network

The FL network is composed of clients whose datasets are either IID or Non-IID split. Each client in the network trains the same DL algorithm with their private dataset and shares the locally trained model with the aggregator. The attacker keeps a historical record of each anonymously submitted model and a representative at each epoch. A representative is a simpler representation of the model, acquired by querying a holdout data piece through the convolutional layers of the model, extracting the embedded space. Therefore, the representatives will later ease the training of the SNN. The representatives are computed by querying the same image through the model to get a more stable representative [38].

Finally, the server aggregates each update by FedAvg [26] and shares the joined model back to clients. This procedure is repeated until convergence is reached. See Algorithm 1 for a summary.

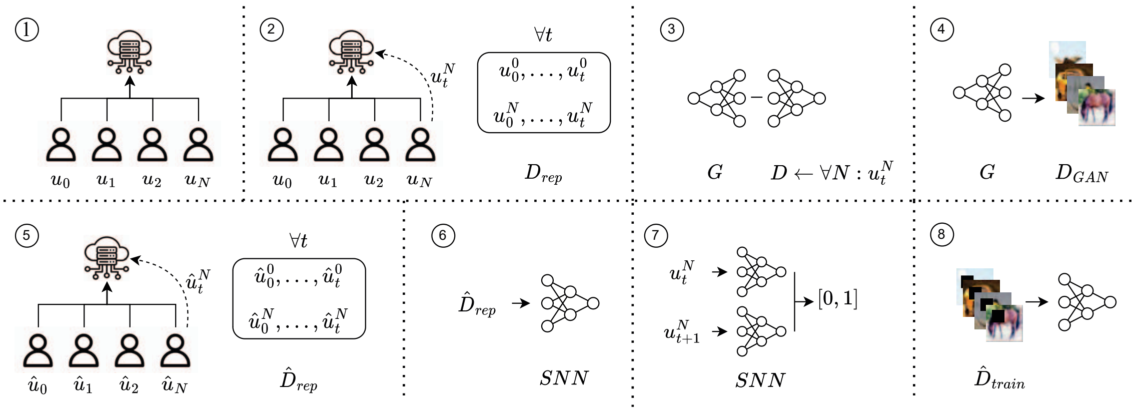


Fig. 1: The overview of the attack: in ① and ②, the FL is trained, and the attacker keeps a collection of submitted models. The attacker generated fake image samples in ③ and ④. In ⑤, the attacker trains a shadow network. In ⑥ and ⑦, the attacker identifies the victim using an SNN. Lastly, in ⑧, the attacker injects the backdoor on the victim's model.

Algorithm 1 FL Network Training

```
Input: A set of clients $K$. The number of clients $N$. Number of epochs $T$. Client local model $u$. Global model $W$.
Output: A collection of anonymous clients' representatives $D_{rep}$. A collection of anonymous clients' models per epoch $M$.
Initialize: $W$
$\mathbf{x} \leftarrow$ get_sample()                          ▷ Get the fixed sample.
for each epoch $t = 1, 2, 3, ..., T$ do
    for each client $k \in K$ do       ▷ $k$ is anonymous for the server.
        $u_{t+1}^k \leftarrow client\_update(k, W_t)$        ▷ Local training of $k$.
        $D_{rep\ t+1}^k \leftarrow u_{t+1}^k(\mathbf{x})$       ▷ Representatives over input $\mathbf{x}$
        $M_{t+1}^k \leftarrow u_{t+1}^k$
    $W_{t+1} \leftarrow W_t + \frac{1}{N} \sum_{i=1}^{N} u_{t+1}^i$
```

C. Creating Synthetic Data

The attacker aims to replicate the FL network to collect information on the clients' behavior. The replica of the FL network is called a shadow network and is composed of shadow clients. For training, shadow clients require datasets that should be as close as possible to the distribution of the clients' datasets. To get data similar to the clients' datasets, we implement a GAN-based model inversion attack, adapting prior work [38]. To mimic the data distribution of a specific client, we initialize the discriminator's weights by the weights of the victim model, as suggested in [9].

It is essential to carefully choose the model from which the weights are transferred to the discriminator. In FL, the models merge their unique properties due to aggregation at every iteration. Intuitively, models at early epochs have not yet acquired other models' properties. Consequently, choosing the epoch from which the model will replace the discriminator is essential for mimicking clients' data.

Selecting the right epoch t is dependent on the use case. For simpler models and datasets, a few epochs are enough to learn the properties of the own model without fully merging the rest. However, more complex setups could require more training. We observe that carefully choosing t is essential for the Non-IID case. In IID settings, models are similar from the beginning; thus, this effect is almost invisible. For details, see Section V. The procedure is summarized in Algorithm 2.

Algorithm 2 Creating Synthetic Data

```
Input: $K$ set of clients. $M$ collection of anonymous clients' models. Trained global model $W'$. $T$ number of epochs. $D$ is the discriminator, and $G$ is the generator.
Output: $D_{GAN}$ collection of GAN generated datasets.
Initialize: $t = 1$                                   ▷ Set a low value of $t$.
for each client $k = 1, 2, 3, ..., K$ do
    Initialize: $G$ and $D$.
    $D \leftarrow W_t^{'k}$
    for each epoch $t = 1, 2, 3, ..., T$ do
        $\mathbf{z} \leftarrow generate\_noise()$
        $train(G, M, \mathbf{z})$
    $z \leftarrow generate\_noise()$
    $\mathbf{x} \leftarrow G(\mathbf{z})$                                 ▷ Create fake data.
    $\{\mathbf{x}, y\} \leftarrow W'(\mathbf{x})$                              ▷ Label data.
    $X^k \leftarrow \{\mathbf{x}, y\}$
```

D. Shadow Training

Shadow models were introduced by Shokri et al. [35] for a membership inference attack. We modify the base idea to fit the FL requirements. Under our settings, we replicate the entire FL network in an isolated environment, namely a shadow network. It is composed of shadow clients, models, datasets, and a server. The attacker gains white-box access to the shadow training procedure by mimicking the FL network. We emphasize that the actual clients, their datasets, and the server are used in the shadow network. The shadow clients have the GAN-generated samples D_{GAN} as their training set and locally train the same DL algorithm in the "real" FL procedure.

Similarly, we also extract the shadow representatives of the shadow models (which are not anonymous) and keep a record of them for each shadow client and epoch. By shadow training, the attacker gains knowledge of the relationship between datasets and models in a similar way to the actual procedure. The procedure is summarized in Algorithm 3.

Algorithm 3 Shadow Training

```
Input: A set of GAN generated dataset $D_{GAN}$. Set of shadow clients $\hat{K}$. Number of epochs $T$. The number of shadow clients $\hat{N}$. $\hat{u}$ shadow client local model. $\hat{W}$ shadow global model.
Output: $\hat{D}_{rep}$ shadow clients' representatives dataset.
$\mathbf{x} \leftarrow get\_sample()$    ▷ Get the fixed sample for calculating clients' representatives.
Initialize: $\hat{W}$
for each epoch $t = 1, 2, 3, ..., T$ do
    for each client $\hat{k} \in \hat{K}$ do
        $\hat{u}^{k}_{t+1} \leftarrow client\_update(\hat{k}, \hat{W}_t, D^{\hat{k}}_{GAN})$
        $\hat{D}^{\hat{k}}_{rep\ t+1} \leftarrow \hat{u}^{\hat{k}}_{t+1}(\mathbf{x})$
    $\hat{W}_{t+1} \leftarrow \hat{W}_t + \frac{1}{\hat{N}} \sum_{i=1}^{\hat{N}} \hat{u}^{i}_{t+1}$
```

E. SNN Training & Update Identification

SNN has already been used for inference attacks, e.g., [38]. During the training of the SNN, the authors used two representatives of different models as input. The SNN seeks similarities between them and outputs a value between "0" and "1", where zero means very similar.

In our attack, we use SNN to identify the victim from the models' representatives gathered previously. However, first, the attacker needs to train the SNN. The attacker owns a record of anonymous model updates from the "real" network and a copy of not anonymous (labeled) updates from the shadow network. The attacker aims to find relations between the labeled dataset to identify the unlabeled dataset. Since data is multidimensional, we use triplet mining for SNN, which requires three inputs for training, which are constructed on the fly, i.e., online triplet mining. By doing so, the SNN can measure the similarity between two representatives, allowing it to match a client's identity with its representative.

Before injecting the backdoor, it is necessary first to identify the victim. Clients' updates are dissimilar at the first epochs, and they get similar as the aggregation process is repeated—a model from the first epoch and another from the last are highly different. We develop a client identification algorithm, greedy searching for the closest relation between representatives across epochs, which links models from early and last epochs. We use the trained SNN and the anonymous set of model representatives acquired in Section IV-B, which stores anonymous representations of every client model at every epoch. Precisely, a representative is chosen $rep^{i}_{t=0}$ at the first epoch $t = 0$, and compared with another one in the subsequent epoch $t = 1 : \{0, 1\} \leftarrow SNN(rep^{i}_{t=0}, rep^{j}_{t=1})$. If the representatives are similar $1 \simeq SNN(rep^{i}_{t=0}, rep^{j}_{t=1})$, then it belongs to the same client $i = j$. The process is repeated for all consecutive epochs in training. Subsequently, the algorithm maps the representatives across epochs, easing the identification of the upcoming model.

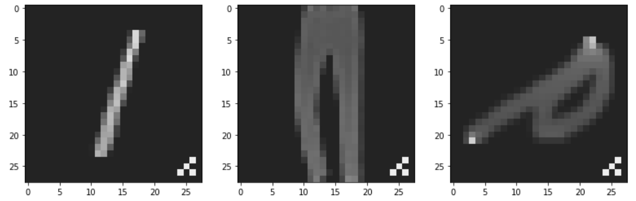
Furthermore, since the generated dataset is created from a client model, this process allows the mapping between the dataset and the client, easing the identification of the victim.

F. Backdoor Attack

Backdoor attacks directly inject the adversarial effect into the model, fired by a trigger in the input sample. The backdoor is created by injecting a four-pixel pattern in the bottom right corner into the samples (Figure 2) or with a sample-specific pattern (Figure 3). Different trigger types, positioning, and combinations lead to variations in the clean data and backdoor accuracy, which we omit here. We refer the reader to [21] for the details of the effectiveness of different triggers.

Backdoor attacks poison the dataset for several epochs in ML and FL, starting from the first epoch [21]. However, our proposal injects the backdoor by retraining the model for a few epochs, which is then sent to the victim client, while the rest receive the non-poisoned one, see Algorithm 4.

With the victim identified, the attacker can inject the backdoor into the global model and share it with the victim. Thus, the attacker injects the backdoor before the FL training ends, i.e., before convergence. The attacker estimates the convergence of the model considering prior models. The attacker defines a stop criterion to know at which epoch the backdoor should be injected. For example, the stop criterion could be a negligible change in the model performance from previous epochs, indicating that the model is near convergence.



(a) MNIST (b) F-MNIST (c) EMNIST

Fig. 2: Backdoored images.

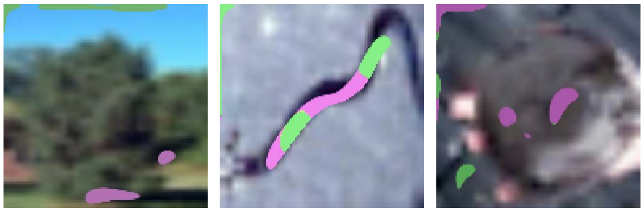


Fig. 3: Samples containing dynamic backdoor triggers.

V. Experimental Results

This section provides an overview of the evaluated datasets and experimental setup, followed by the results and discussion.

A. Datasets

We evaluate the performance of our attack on the MNIST [24], EMNIST [10], F-MNIST [50], and CIFAR-100 [23] datasets. MNIST is a common benchmark dataset in computer vision containing labeled grayscale images from handwritten digits. Dataset labels range from "0" to "9".

Algorithm 4 Client identification & Backdoor

```
Input: Unidentified clients' representatives $D_{rep}$. Target class $c_t$. Source class $c_s$. GAN generated datasets $D_{GAN}$. Poisoned data rate $\epsilon$.
Output: Backdoored model $W_{\hat{\theta}}$.
for each unidentified client representative pair $x, x' \in D_{rep} : x \neq y$ do
    $SNN(x, x')$        ▷ Similarity calculation as in Section IV-E
Define: $u_v$        ▷ Define a victim client
$\hat{D}_{train} \leftarrow backdoor(c_s, c_t, D^v_{GAN}, \epsilon)$
$W_{\hat{\theta}} \leftarrow train(\hat{D}_{train}, u_v)$
Send $W_{\hat{\theta}}$ to victim client $v$.
```

EMNIST is a grayscale dataset containing handwritten characters of the alphabet containing 26 classes of images. F-MNIST is a grayscale dataset containing ten types of clothing. Every dataset contains 70 000 28×28×1 grayscale samples, 60 000 for training, and 10 000 for the test set. CIFAR-100 dataset consists of 60 000 32×32 color images in 100 classes with 600 images per class for the training set and 10 000 samples for the test set.[3] Our selection of datasets allows us to consider standard settings and investigate scenarios with different number classes.

B. FL Network Settings

For MNIST, F-MNIST, and EMNIST datasets, the model is a convolutional neural network (CNN), with three convolutional layers and one fully connected, with stochastic gradient descent, LeakyRelu as an activation function, and batch normalization in each layer except the last. Experimentally set training settings are shown in Table I. The architecture is shown in Table II, and it is a commonly used convolution network for image classification tasks [40]. For the CIFAR-100 dataset, we use VGG11 with batch normalization [36] architecture, and we use transfer learning from pretrained weights from ImageNet1K [12] by freezing the convolutional layers during training.

TABLE I: Training settings.

Dataset	*LR*	Momentum	Local Epoch	FL Epoch	No. of Clients	No. of Classes
MNIST	0.1	0.9	2	50	5	10
F-MNIST	0.00001	0.0	1	200	5	10
EMNIST	0.01	0.9	2	200/30†	13	26
CIFAR100	0.001	0.9	1	23	10	100

† represents the number of epochs for Non-IID/IID.

After training, with the Non-IID setting, the network achieves 95% accuracy on MNIST, 78% on F-MNIST, 80% on EMNIST, and 65% on CIFAR-100, see Figures 10b, 11b, 12b, and 13b in Appendix VIII.[4] Regarding the IID setting, the model reaches better results, 99% on MNIST, 80% on F-MNIST, 88% on EMNIST, and 76% on CIFAR-100, see Figures 10a, 10a, 12a, and 13a in Appendix VIII. Models are trained over non-colluding (Non-IID) or overlapping (IID) labeled data and evaluated with a test dataset containing all the labels. As epochs progress, models perform better over the test set, acquiring properties from other datasets.

TABLE II: CNN architecture.

Layer	Out Shape	# Param
Conv2D	(None, 64, 14, 14)	640
LeakyReLu	(None, 64, 14, 14)	-
Conv2D	(None, 128, 7, 7)	73 856
BatchNorm2D	(None, 128, 7, 7)	256
LeakyReLu	(None, 128, 7, 7)	-
Conv2D	(None, 256, 3, 3)	295 168
BatchNorm2D	(None, 256, 3, 3)	512
LeakyReLu	(None, 256, 3, 3)	-
Linear	(None, 10†)	23 050
Total		393 482

† "10" changes with the number of classes.

For the design of the GAN, we follow the deep convolutional GAN architecture presented by Radford et al. [31]. For MNIST, F-MNIST, and EMNIST datasets, the generator is composed of 4 deconvolutional layers with batch normalization and ReLu activation functions followed by a tanh activation, see Table III. The discriminator has four convolutional layers with batch normalization and Leaky ReLu activation functions with a sigmoid function in the last layer; see Table IV.

For CIFAR-100, the generator has five deconvolutional layers instead of four, with batch normalization and ReLu activation functions followed by a tanh. The discriminator also contains five convolutional layers with batch normalization and Leaky ReLu activation functions followed by a sigmoid.

The discriminator is initialized with the weights of the client's model for generating client-specific data. Precisely, the weights of the convolutional layers are assigned to the discriminator. During training, the generator creates fake images that the discriminator has to differentiate from the actual ones.

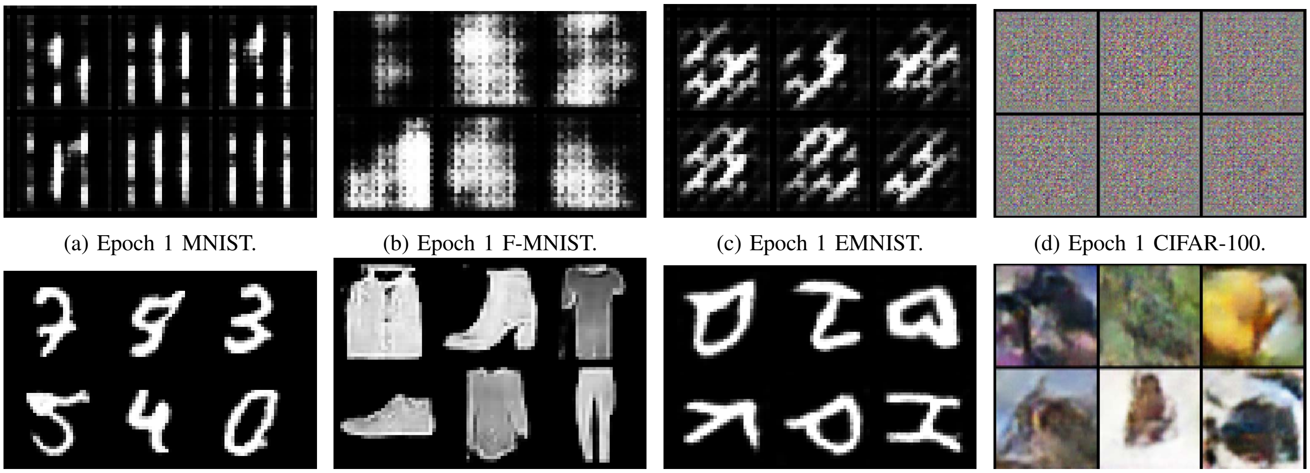
We train the GAN for 950 epochs in the CIFAR-100 dataset and 1 000 epochs for the rest of the datasets. The *LR* is set to 0.002, and Adam is used as the optimizer, as suggested in [31]. Using the global model at a certain epoch, the attacker generates 5 000 labeled images per client (Figure 4) that would be used in the shadow training.

C. Shadow Network Training Settings

A shadow network replicates the original FL network introduced in Section IV-B and utilizes the same hyperparameters as in Section V-B. The shadow network contains shadow clients and shadow datasets. Shadowing the FL network allows the attacker to gain white-box access to the entire training procedure. Shadow clients own shadow datasets, i.e., the synthetic dataset generated previously containing 5 000 data samples. Shadow clients train their shadow models and upload them to the shadow server. The attacker extracts each shadow model representative and matches them with a shadow client at every epoch. Since the shadow models are trained with synthetic data, their shadow models' accuracies are lower than

[3] We upscale CIFAR-100 images to 128×128 to improve training quality.

[4] Results are averaged over ten executions.



(a) Epoch 1 MNIST.

(b) Epoch 1 F-MNIST.

(c) Epoch 1 EMNIST.

(d) Epoch 1 CIFAR-100.

(e) Epoch 1 000 MNIST.

(f) Epoch 1 000 F-MNIST.

(g) Epoch 1 000 EMNIST.

(h) Epoch 950 CIFAR-100.

Fig. 4: GAN generated MNIST, F-MNIST, EMNIST, and CIFAR-100 images at different epochs.

TABLE III: Generator architecture.

Layer	# Channels	
	CIFAR-100	Others
ConvTranspose2D	(None, 512, 4, 4)	(None, 256, 64, 64)
BatchNorm2D	(None, 512, 4, 4)	(None, 256, 64, 64)
ReLu	(None, 512, 4, 4)	(None, 256, 64, 64)
ConvTranspose2D	(None, 256, 8, 8)	(None, 128, 7, 7)
BatchNorm2D	(None, 256, 8, 8)	(None, 128, 7, 7)
ReLu	(None, 256, 8, 8)	(None, 128, 7, 7)
ConvTranspose2D	(None, 128, 16, 16)	(None, 64, 14, 14)
BatchNorm2D	(None, 128, 16, 16)	(None, 64, 14, 14)
ReLu	(None, 128, 16, 16)	(None, 64, 14, 14)
ConvTranspose2D	(None, 64, 32, 32)	-
BatchNorm2D	(None, 64, 32, 32)	-
ReLu	(None, 64, 32, 32)	-
ConvTranspose2D	(None, 3, 64, 64)	(None, 1, 28, 28)
Tanh	(None, 3, 64, 64)	(None, 1, 28, 28)

TABLE IV: Discriminator architecture.

Layer	# Channels	
	CIFAR-100	Others
Conv2D	(None, 64, 32, 32)	(None, 64, 14, 14)
LeakyReLu	(None, 64, 32, 32)	(None, 64, 14, 14)
Conv2D	(None, 128, 16, 16)	(None, 128, 7, 7)
BatchNorm2D	(None, 128, 16, 16)	(None, 128, 7, 7)
LeakyReLu	(None, 128, 16, 16)	(None, 128, 7, 7)
Conv2D	(None, 256, 8, 8)	(None, 256, 3, 3)
BatchNorm2D	(None, 256, 8, 8)	(None, 256, 3, 3)
LeakyReLu	(None, 256, 8, 8)	(None, 256, 3, 3)
Conv2D	(None, 512, 4, 4)	-
BatchNorm2D	(None, 512, 4, 4)	-
LeakyReLu	(None, 512, 4, 4)	-
Conv2D	(None, 1, 1, 1)	(None, 1, 1, 1)
Sigmoid	(None, 1, 1, 1)	(None, 1, 1, 1)

the actual models. However, the shadow models' accuracy is irrelevant and is just used to extract identified representatives. After this process, the attacker owns a dataset of shadow model representatives.

D. Triplet SNN Training Settings

Each model of the SNN is composed of three fully connected layers with dropout layers between them. This simple, yet effective design, grants excellent performance with low amounts of data. More complex models could easily overfit. Furthermore, we require triplet mining to improve the quality of the network. The network inputs are an anchor, a positive, and a negative 8 192-dimensional samples for the CIFAR-100 use case and 2 304-dimensional samples for the rest of the cases. The online triplet mining procedure selects the best triplet combination, being the anchor and the positive sample representatives from the same client, while the negative is from another client.

The outputs from the last layers are embedded in a five-dimensional space (experimentally set as a trade-off between network complexity and data dimensionality) and sent to a distance computing layer that calculates the Euclidean distance. After training, given two inputs, the SNN yields values close to 0 if they are similar and close to 1 otherwise. Our experiments show the network reaches an accuracy of 97% and IID and 80% with Non-IID for MNIST, 80% and IID and 78% with Non-IID for F-MNIST, 90% and IID and 88% with Non-IID for EMNIST, after 20 epochs, α 0.2, and an *LR* of 0.0001, with Adam as the optimizer. For the CIFAR-100 case, we train the SNN for 100 epochs, achieving an accuracy of 85% in the IID case and 89% in the Non-IID case. We can observe a slight degradation in the SNN accuracy with IID data caused by the high similarity between models.

E. Backdoor Attack Settings

As stated before, the backdoor generation function is independent of our pipeline, allowing the attacker to adapt smartly to defense mechanisms. To show the attack's modularity, we implement two backdoors: BadNets [21] and input-aware dynamic backdoors [29].

In BadNets, the trigger is generated by adding white pixels on top of the original image. To be precise, we used four white pixels placed in the bottom right corner of the image, as shown in Figure 2. To make the attack class targeted, we only poison images of a given source class. Note that the pixel pattern is the same for all samples.

Regarding the input-aware dynamic approach, the backdoor pattern is created individually per sample using a trigger generator network. The generated poisoned samples are shown in Figure 3. We use the *Trojanzoo* [30] framework for this process.

After generating the poisoned training set by some of the abovementioned methods, the attacker injects the backdoor into the global model. A value ϵ controls the amount of poisoned data in the dataset. A successful backdoor should maintain high accuracy for both source and target classes, measured by the metrics defined in Section III-B.

We experimentally test the input-aware dynamic backdoor in the CIFAR-100 dataset and the BadNets approach in the rest of the datasets. The input-aware dynamic backdoor is trained for 100 epochs with a *LR* of 0.001 and Adam as the *SGD* optimizer.

For BadNets, we train for ten epochs, *LR* 0.0001, *SGD* as the optimizer, and momentum 0.9 for MNIST and EMNIST for both IID and Non-IID settings. For F-MNIST, we retrain the model for 20 epochs, LR 0.01, SGD as an optimizer, and momentum of 0.9 for both IID and Non-IID. Using these hyperparameters, we define two attack scenarios: *0 to 9*, where "0" is the source class and "9" is the target class, and *1 to 7* where "1" is the source and "7" the target class, respectively, for MNIST. In the F-MNIST dataset, "0" corresponds to "T-shirt", "1" to "Trousers", "7" to "Sneakers", and "9" to Ankle boot". Regarding EMNIST, "0" corresponds to "a", "1" to "b", "7" to "h", and "9" to "j".

We further evaluate the attack for different ϵ values and validate the attack performance by checking the target class ASR. For BadNets, we observe high ASR in all settings, see Figure 5, and almost no degradation in either the source or target class with respect to the accuracy of the main task (Figure 6). With IID and non-IID settings, the achieved ASR in the target class is up to 99% in MNIST, 94% in F-MNIST, and 96% in EMNIST. However, the ASR is slightly lower in the input-aware dynamic backdoor in the CIFAR-100 dataset, achieving up to 84% in Non-IID and 77% in IID; see Figure 7.

Note that the attacker should be careful when setting a large value of ϵ, which could cause degradation on the main task while achieving higher accuracy on the backdoor task. Furthermore, using a very large ϵ value such as $\epsilon = 1$, i.e., no source class in the training set, could cause the model to "forget" the source class, causing the model to perform poorly in that specific class. This finding could potentially be used to defend against such targeted attacks, as discussed in Section VI.

F. Comparison with the Existing Backdoor Attacks in FL

This section compares our results with the state-of-the-art backdoor attacks in FL. Before discussing the results, it is important to note that none of the existing methods target a single client; we try our best to find comparable results.

Sun et al. [39] introduced backdoors attacks in FL, where a subset of clients are chosen as attackers who train their model on poisoned data. They used a CNN model and EMNIST as the dataset, similar to ours. Despite training with a different number of clients, more epochs, and different LR, we still achieve similar results on the main and backdoor task, up to 80% ASR, and less than 1% degradation on the main task. Since backdoor models have a substantially larger norm than non-infected models, they test their attack against a norm threshold as a defense, drastically reducing the ASR while maintaining high accuracy on the main task.

Wang et al. [43] investigated the same threat in FL in the image and text domains. The authors leverage *edge case* samples—data pieces that rarely appear in the training dataset—as candidates for injecting the backdoor. They evaluate their attack against different existing techniques, bypassed by a projected gradient descent attack or clipping the norm of the model. Without defenses, despite the use of a different dataset (CIFAR-10 with some added edge case samples), they achieve comparable accuracy on the main (99%) and backdoor accuracies (80%).

Lastly, *DBA* [52] is a distributed attack that separates the triggers into pieces and shares each with the attackers in the FL network. At the same epochs, the attackers will share the poisoned model—trained on a poisoned dataset containing the trigger piece—merging the trigger pieces at aggregation. The result is a fully backdoored model, more stealthy than other centralized backdoor attacks. Their attack achieves 91% ASR on the MNIST dataset, with the trigger placed at the upper left corner, similar to our approach.

VI. Defenses

In this section, we explain the state-of-the-art defense mechanisms against backdoor attacks and evaluate and discuss their effectiveness and applicability against our attack.

A. Generic Defense Methods

To prevent backdoor attacks, several defense mechanisms have been developed recently. Specifically for centralized ML, *dataset inspection* techniques analyze the training set to remove outliners, assuming that the poisoned samples compose a small separate cluster [8]. Similarly, *Neo* [41] generates different variants of the input sample, masking the dominant color, and checks the attack success rate for every pixel in each variant. The model is flagged as malicious if the attack success rate exceeds some threshold. *STRIP* [18] perturbs the samples by adding different patterns and observing the randomness of the model's outcome, which measures its entropy, and can detect if a model is poisoned or not. Dataset inspection defenses do not hold for FL settings, where the datasets are private; thus, we do not consider them for our experimentation.

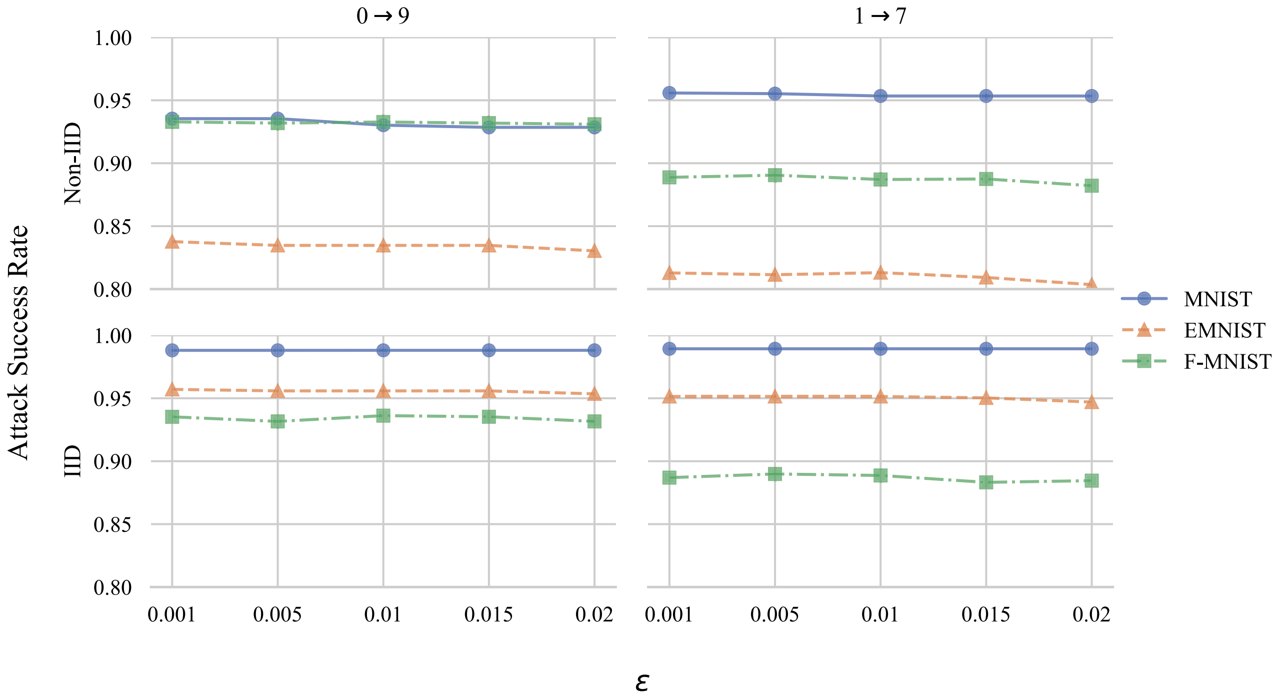


Fig. 5: ASR on the target class under different settings.

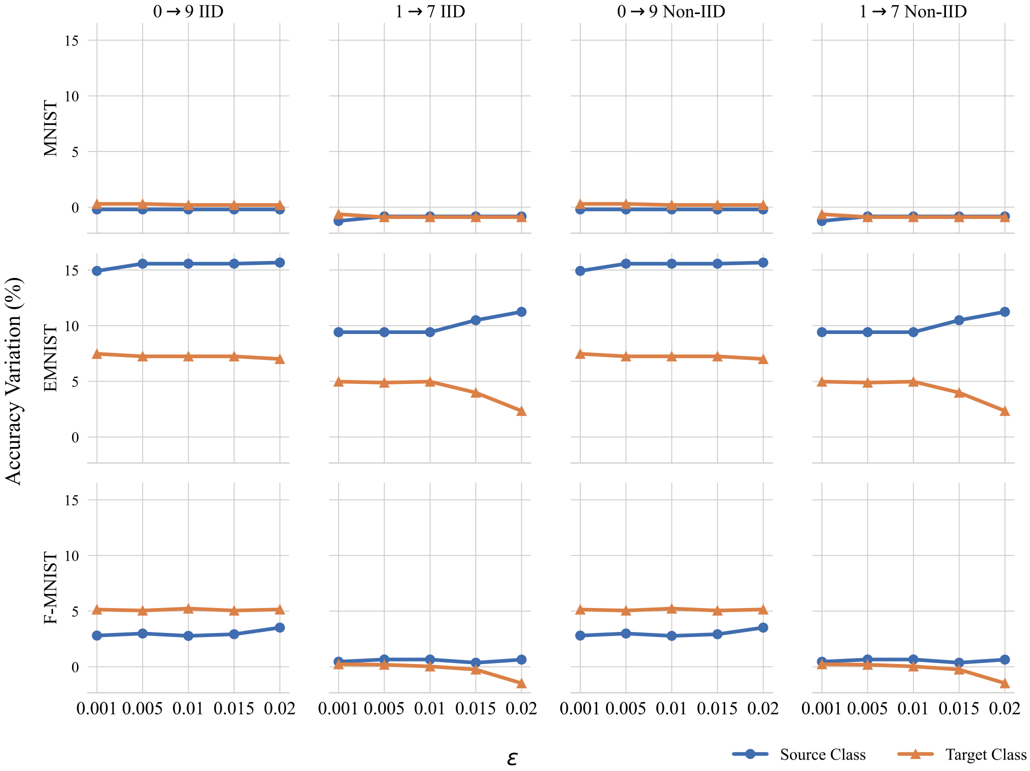


Fig. 6: Accuracy degradation (%) on the target and source classes in a clean test set under different settings.

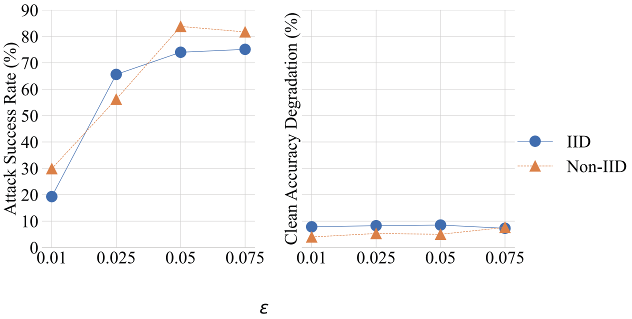


Fig. 7: ASR in the CIFAR-100 dataset under different settings.

Neural Cleanse [42] and *ABS* [25] defend against backdoor attacks using an optimization method to find the smallest perturbation that causes the model to behave abnormally. Since both defense mechanisms work similarly, in this work, we implement the Neural Cleanse defense mechanism and evaluate its effectiveness against our attack. Neural Cleanse is based on the intuition that a poisoned model requires much less modification to cause misclassification into the trigger label than the rest. Iteratively, Neural Cleanse creates the same number of potential triggers as classes in the model, requiring minimal pixel changes to cause misclassification. Subsequently, an outline detection algorithm selects smaller triggers than the rest as malicious, assigns an anomaly score, and successfully reconstructs the target label and trigger. The authors suggest using "2" as a threshold to find malicious labels.

We implement this defense against our attack using *Trojanzoo* [30] for the BadNets attack, which has to be slightly modified to support source class targeted backdoors. However, the input-aware dynamic backdoor is not tested again on Neural Cleanse since the triggers are uniquely generated for each sample, preventing the reconstruction performed by Neural Cleanse [29]. We set up Neural Cleanse with the suggested hyperparameters and set it as default in the Trojanzoo implementation. We observe that Neural Cleanse cannot identify the target class as malicious, i.e., classes with an anomaly score greater than "2", in none of our attack settings (Figure 8). The attack reports primarily false positives, assigning high anomaly scores to several classes, sometimes even not including the targeted one. We observe that this effect is due to fixing the source class in our attack, given the intuition provided by the authors of Neural Cleanse: *"Our key intuition of detecting backdoors is that in an infected model, it requires much smaller modifications to cause misclassification into the target label than into other uninfected labels"* [42]. In attacks where the source class is not fixed, all the classes are converted to the target class, thus making the above statement true. However, when only a single class is used to trigger, only small modifications to the input of the source class make it easier to cause misclassification in the target model. Therefore, we can conjecture that: *"In an infected model, detecting backdoors in an infected model requires much smaller modifications to cause misclassification into the target label from the source class than into other uninfected labels or other classes distinct from the source"*.

B. Defense Methods Specific to the FL Setting

FL specific methods, e.g., *Krum* [4], *FoolsGold* [16], *Baffle* [2], *CRLF* [51], and Fu et al. [15] are adopted by the entire network and consider several assets of it. Krum is a robust aggregation mechanism that allows training even with colluding *Byzantine clients* in the network; using a majority-based scheme, aggregate the models that are closer, i.e., more similar. On the same basis, FoolsGold uses cosine similarity between models to discard dissimilar ones. The intuition is that malicious clients collaborating to achieve the same goal have similar cosine models, which differ from honest clients. Regarding our threat model, assuming that the server is malicious, defenses that require server participation are not applicable to our settings, i.e., the FL network cannot assume that those will be implemented. Furthermore, the attack is not launched from the client, which is *always* sharing an honest model.

Similarly, Baffle defends against backdoor attacks by a feedback-based voting mechanism, where every client tests their model with their dataset and submits the results to the server. Thus, the server expects an incremental improvement in accuracy. If most clients report a negative impact on the accuracy, the global model is flagged as malicious. This defense does not work on our attack setting since only one client is backdoored and would report that the accuracy has lowered. However, the majority of clients would report high accuracy since they are not backdoored, causing a false negative. On the same basis, Fu et al. [15] developed a robust aggregation algorithm using residual-based reweighting, which defends against backdoor attacks. This defense mechanism does not apply to our setup since the server supposed to aggregate with the new weighting system is malicious, and the clients cannot validate the correctness of the weighted aggregation.

Xie et al. [51] developed a clipping and smoothing method to defend against backdoor attacks in FL named CRLF. They theorized that norm clipping the model's weights at every epoch and adding Gaussian noise during training time in combination with randomized parameter smoothing at test time could prevent backdoors. We test this defense against our attack by cleanly training a CNN model for the MNIST and EMNIST datasets, keeping the same setting as in our approach, and for defense, we set $\sigma = 0.01$ to generate 1 000 smoothed models and error tolerance of $\alpha = 0.01$, the maximum norm at 100 and the clipping threshold at 15. After applying the smoothing, we did not observe any significant reduction, i.e., less than 1%, in the ASR; see Figure 9 in Appendix VIII. Our experiments show that smoothing by itself is not successful in preventing backdoor attacks. However, clipping cannot be applied since it requires an honest server, which cannot be guaranteed in our settings.

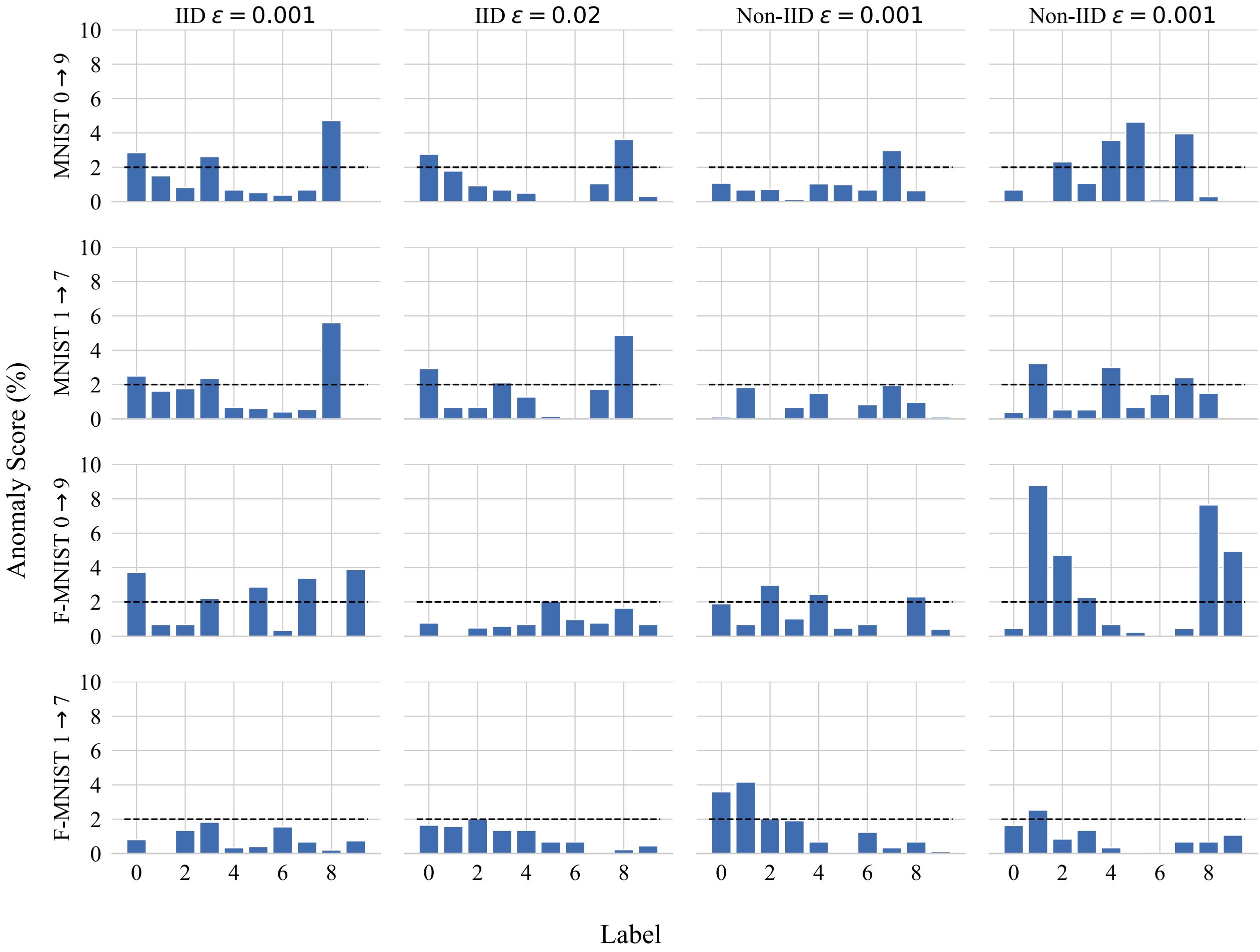


Fig. 8: Anomaly detection of different attacks using Neural Cleanse [42].

C. Discussion

We tested our attack against the state-of-the-art defenses, and we now briefly discuss two potential evasion techniques against our attack and the limitations of our attack.

1) Backdoor models slightly degrade the accuracy of the model. In our approach, since just a subset of clients is backdoored, comparing the accuracy of every model will create inconsistency. Defenses following that approach from [45] assume that the server is trusted, which does not hold for our attack. However, adapting the defense to be executed by a trusted third party could be a feasible defense mechanism.
2) Differential privacy is widely used to prevent inference attacks [46]. Adding noise to the model's weights affects the reconstructed data by model inversion attacks, making the obtained data noisy. Since our attack relies heavily on reconstructing clients' datasets, this defense could make it difficult to launch our attack.

Lastly, our attack is subject to some limitations. First, the attack is costly both in time and computational resources. Therefore, an attacker who lacks these may not be able to perform the attack. Second, despite many attacks considering the server as malicious, it could be a stronger assumption than just assuming a malicious client. Therefore, our attack could not be implemented in scenarios where the server's trust is guaranteed.

VII. Conclusions & Future Work

This paper studies the viability of client-targeted backdoor attacks and demonstrates the high performance of the attack. The first phase of our proposal investigated the creation of client data through a model inversion attack. The second part combines similarity matching and different backdoor techniques to target a single client in an FL network. Our findings suggest that the combination of model inversion attacks and backdoors is a powerful duple, laying the groundwork for new threats. It also demonstrates that the client-targeted backdoor attack poses a real threat to an FL system, highlighting the importance of further research and proposing specific defense strategies against them. Similarly, state-of-the-art defenses either fail to defend our attack or do not apply to the setup. Therefore, we find that developing defense mechanisms that consider client-specific backdoors is necessary.

The generalizability of our results is subject to certain limitations. For instance, broader experimentation with different, more complex models, datasets, and numbers of clients is a natural progression of this work. Nevertheless, this study reinforces the idea that an attacker can cause severe model degradation in a client-targeted manner. Furthermore, we unravel two research directions to be addressed in the future. The first one simplifies the attack's time and computational complexity and makes it suitable for a broader range of setups. The second is to relax the defined assumptions, thus empowering the attack and adapting it to more realistic scenarios. These findings could lead the research toward performing a client-targeted backdoor where a client is an attacker.

References

[1] Abad, G., Picek, S., Ramírez-Durán, V.J., Urbieta, A.: On the security & privacy in federated learning (2021). https://doi.org/10.48550/ARXIV.2112.05423, https://arxiv.org/abs/2112.05423

[2] Andreina, S., Marson, G.A., Möllering, H., Karame, G.: Baffle: Backdoor detection via feedback-based federated learning. In: 2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS). pp. 852–863. IEEE (2021)

[3] Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D., Shmatikov, V.: How to backdoor federated learning. In: International Conference on Artificial Intelligence and Statistics. pp. 2938–2948. PMLR (2020)

[4] Blanchard, P., El Mhamdi, E.M., Guerraoui, R., Stainer, J.: Machine learning with adversaries: Byzantine tolerant gradient descent. Advances in Neural Information Processing Systems **30** (2017)

[5] Boenisch, F., Dziedzic, A., Schuster, R., Shamsabadi, A.S., Shumailov, I., Papernot, N.: When the curious abandon honesty: Federated learning is not private. arXiv preprint arXiv:2112.02918 (2021)

[6] Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., Shah, R.: Signature verification using a" siamese" time delay neural network. Advances in neural information processing systems **6** (1993)

[7] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. Advances in neural information processing systems **33**, 1877–1901 (2020)

[8] Chen, B., Carvalho, W., Baracaldo, N., Ludwig, H., Edwards, B., Lee, T., Molloy, I., Srivastava, B.: Detecting backdoor attacks on deep neural networks by activation clustering. arXiv preprint arXiv:1811.03728 (2018)

[9] Chen, J., Zhang, J., Zhao, Y., Han, H., Zhu, K., Chen, B.: Beyond model-level membership privacy leakage: an adversarial approach in federated learning. In: 2020 29th International Conference on Computer Communications and Networks (ICCCN). pp. 1–9. IEEE (2020)

[10] Cohen, G., Afshar, S., Tapson, J., Van Schaik, A.: Emnist: Extending mnist to handwritten letters. In: 2017 international joint conference on neural networks (IJCNN). pp. 2921–2926. IEEE (2017)

[11] Cortes, C., Vapnik, V.: Support-vector networks. Machine learning **20**(3), 273–297 (1995)

[12] Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. pp. 248–255. Ieee (2009)

[13] Dingledine, R., Mathewson, N., Syverson, P.: Tor: The second-generation onion router. Tech. rep., Naval Research Lab Washington DC (2004)

[14] Fredrikson, M., Jha, S., Ristenpart, T.: Model inversion attacks that exploit confidence information and basic countermeasures. In: Proceedings of the 22nd ACM SIGSAC conference on computer and communications security. pp. 1322–1333 (2015)

[15] Fu, S., Xie, C., Li, B., Chen, Q.: Attack-resistant federated learning with residual-based reweighting. arXiv preprint arXiv:1912.11464 (2019)

[16] Fung, C., Yoon, C.J., Beschastnikh, I.: Mitigating sybils in federated learning poisoning. arXiv preprint arXiv:1808.04866 (2018)

[17] Gao, Y., Doan, B.G., Zhang, Z., Ma, S., Zhang, J., Fu, A., Nepal, S., Kim, H.: Backdoor attacks and countermeasures on deep learning: A comprehensive review. arXiv preprint arXiv:2007.10760 (2020)

[18] Gao, Y., Xu, C., Wang, D., Chen, S., Ranasinghe, D.C., Nepal, S.: Strip: A defence against trojan attacks on deep neural networks. In: Proceedings of the 35th Annual Computer Security Applications Conference. pp. 113–125 (2019)

[19] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. Advances in neural information processing systems **27** (2014)

[20] Greff, K., Srivastava, R.K., Koutník, J., Steunebrink, B.R., Schmidhuber, J.: Lstm: A search space odyssey. IEEE transactions on neural networks and learning systems **28**(10), 2222–2232 (2016)

[21] Gu, T., Liu, K., Dolan-Gavitt, B., Garg, S.: Badnets: Evaluating backdooring attacks on deep neural networks. IEEE Access **7**, 47230–47244 (2019)

[22] Kairouz, P., McMahan, H.B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A.N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., et al.: Advances and open problems in federated learning. arXiv preprint arXiv:1912.04977 (2019)

[23] Krizhevsky, A.: Learning multiple layers of features from tiny images. Tech. rep. (2009)

[24] LeCun, Y.: The mnist database of handwritten digits. http://yann. lecun. com/exdb/mnist/ (1998)

[25] Liu, Y., Lee, W.C., Tao, G., Ma, S., Aafer, Y., Zhang, X.: Abs: Scanning neural networks for back-doors by artificial brain stimulation. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 1265–1282 (2019)

[26] McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Artificial intelligence and statistics. pp. 1273–1282. PMLR (2017)

[27] Mothukuri, V., Parizi, R.M., Pouriyeh, S., Huang, Y., Dehghantanha, A., Srivastava, G.: A survey on security and privacy of federated learning. Future Generation Computer Systems **115**, 619–640 (2021)

[28] Nasr, M., Shokri, R., Houmansadr, A.: Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In: 2019 IEEE symposium on security and privacy (SP). pp. 739–753. IEEE (2019)

[29] Nguyen, T.A., Tran, A.: Input-aware dynamic backdoor attack. Advances in Neural Information Processing Systems **33**, 3454–3464 (2020)

[30] Pang, R., Zhang, Z., Gao, X., Xi, Z., Ji, S., Cheng, P., Luo, X., Wang, T.: Trojanzoo: Towards unified, holistic, and practical evaluation of neural backdoors. In: 2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P). pp. 684–702. IEEE (2022)

[31] Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434 (2015)

[32] Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M., Sutskever, I.: Zero-shot text-to-image generation. In: International Conference on Machine Learning. pp. 8821–8831. PMLR (2021)

[33] Schroff, F., Kalenichenko, D., Philbin, J.: Facenet: A unified embedding for face recognition and clustering. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 815–823 (2015)

[34] Shafahi, A., Huang, W.R., Najibi, M., Suciu, O., Studer, C., Dumitras, T., Goldstein, T.: Poison frogs! targeted clean-label poisoning attacks on neural networks. arXiv preprint arXiv:1804.00792 (2018)

[35] Shokri, R., Stronati, M., Song, C., Shmatikov, V.: Membership inference attacks against machine learning models. In: 2017 IEEE symposium on security and privacy (SP). pp. 3–18. IEEE (2017)

[36] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)

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

[38] Song, M., Wang, Z., Zhang, Z., Song, Y., Wang, Q., Ren, J., Qi, H.: Analyzing user-level privacy attack against federated learning. IEEE Journal on Selected Areas in Communications **38**(10), 2430–2444 (2020)

[39] Sun, Z., Kairouz, P., Suresh, A.T., McMahan, H.B.: Can you really backdoor federated learning? arXiv preprint arXiv:1911.07963 (2019)

[40] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolu-

tions. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1–9 (2015)

[41] Udeshi, S., Peng, S., Woo, G., Loh, L., Rawshan, L., Chattopadhyay, S.: Model agnostic defence against backdoor attacks in machine learning. IEEE Transactions on Reliability (2022)

[42] Wang, B., Yao, Y., Shan, S., Li, H., Viswanath, B., Zheng, H., Zhao, B.Y.: Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In: 2019 IEEE Symposium on Security and Privacy (SP). pp. 707–723. IEEE (2019)

[43] Wang, H., Sreenivasan, K., Rajput, S., Vishwakarma, H., Agarwal, S., Sohn, J.y., Lee, K., Papailiopoulos, D.: Attack of the tails: Yes, you really can backdoor federated learning. Advances in Neural Information Processing Systems **33**, 16070–16084 (2020)

[44] Wang, J., Song, Y., Leung, T., Rosenberg, C., Wang, J., Philbin, J., Chen, B., Wu, Y.: Learning fine-grained image similarity with deep ranking. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1386–1393 (2014)

[45] Wang, Y., Zhu, T., Chang, W., Shen, S., Ren, W.: Model poisoning defense on federated learning: A validation based approach. In: International Conference on Network and System Security. pp. 207–223. Springer (2020)

[46] Wei, K., Li, J., Ding, M., Ma, C., Yang, H.H., Farokhi, F., Jin, S., Quek, T.Q., Poor, H.V.: Federated learning with differential privacy: Algorithms and performance analysis. IEEE Transactions on Information Forensics and Security **15**, 3454–3469 (2020)

[47] Wu, L., Perin, G., Picek, S.: The best of two worlds: Deep learning-assisted template attack. Cryptology ePrint Archive (2021)

[48] Wu, Z., Shen, C., Van Den Hengel, A.: Wider or deeper: Revisiting the resnet model for visual recognition. Pattern Recognition **90**, 119–133 (2019)

[49] Xiao, C., Li, B., Zhu, J.Y., He, W., Liu, M., Song, D.: Generating adversarial examples with adversarial networks. arXiv preprint arXiv:1801.02610 (2018)

[50] Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747 (2017)

[51] Xie, C., Chen, M., Chen, P.Y., Li, B.: Crfl: Certifiably robust federated learning against backdoor attacks. In: Proceedings of the 38th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 139, pp. 11372–11382. PMLR (18–24 Jul 2021), http://proceedings.mlr.press/v139/xie21a.html

[52] Xie, C., Huang, K., Chen, P.Y., Li, B.: Dba: Distributed backdoor attacks against federated learning. In: International Conference on Learning Representations (2019)

[53] Zhang, J., Zhang, J., Chen, J., Yu, S.: Gan enhanced membership inference: A passive local attack in federated learning. In: ICC 2020-2020 IEEE International Conference on Communications (ICC). pp. 1–6. IEEE (2020)

VIII. Additional Experimental Results

Figure 9 represents the results of the backdoor attack after smoothing is applied as a defense mechanism. The experimentation shows negligible degradation in accuracy.

Figures 10, 11, 12, and 13 show the accuracy of training on the test set for MNIST, F-MINST, EMNIST, and CIFAR-100 respectively.

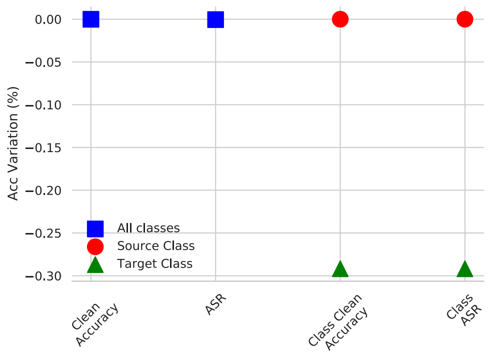
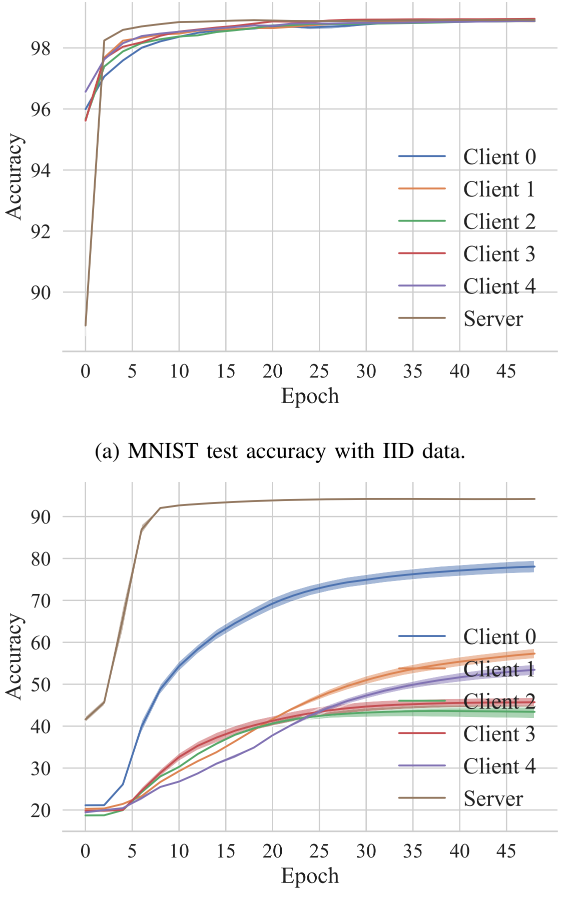


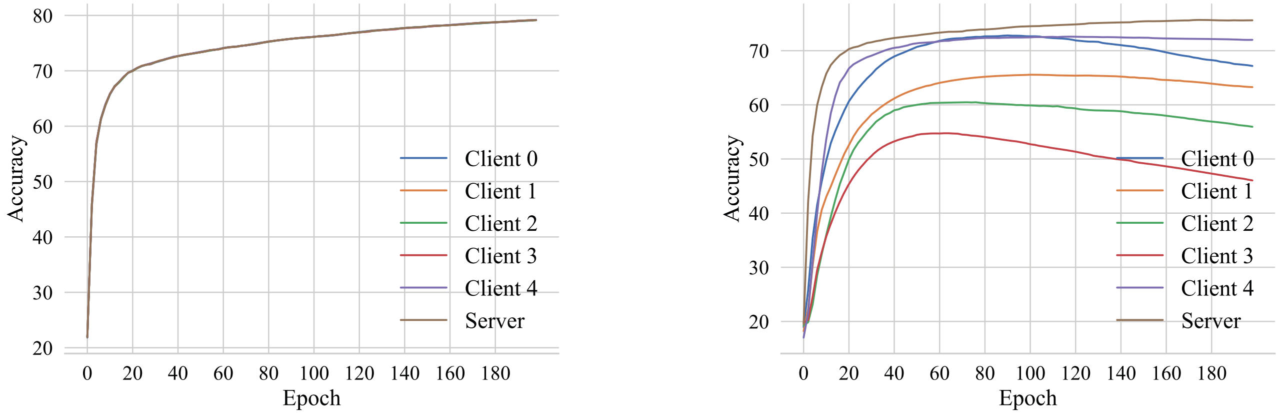
Fig. 9: Accuracy variation after smoothing for $\epsilon = 0.1$ MNIST $1 \rightarrow 7$ setting.



(a) MNIST test accuracy with IID data.

(b) MNIST test accuracy with Non-IID data.

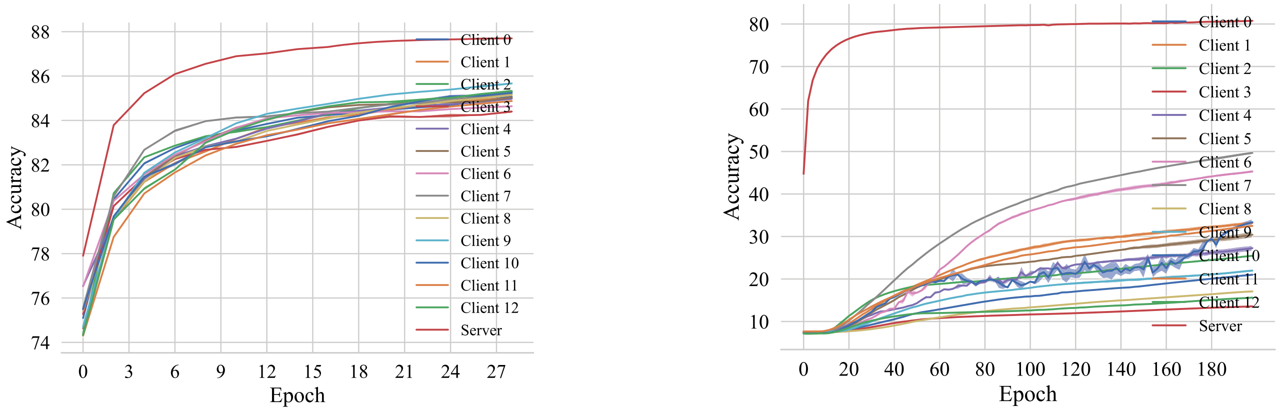
Fig. 10: CNN averaged testing accuracy for MNIST.



(a) F-MNIST test accuracy with IID data.

(b) F-MNIST test accuracy with Non-IID data.

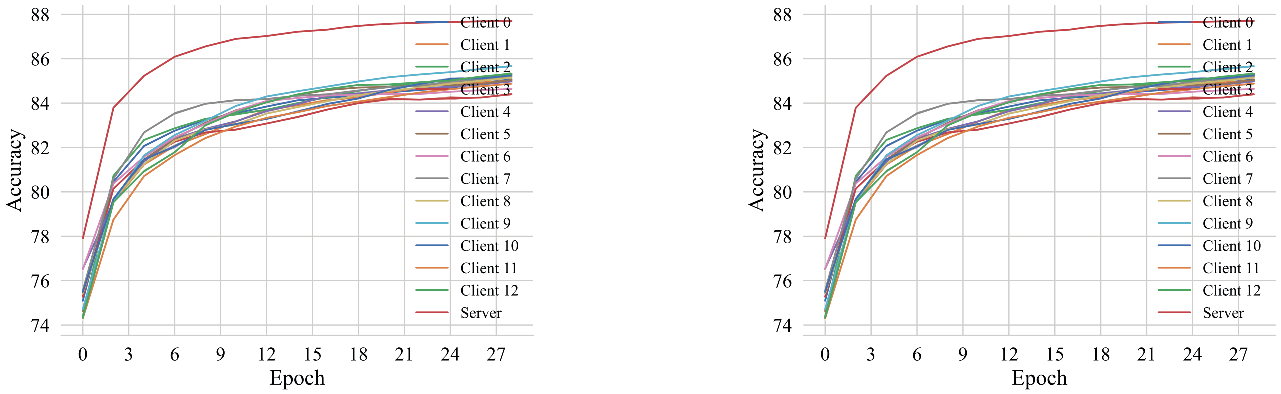
Fig. 11: CNN averaged testing accuracy for F-MNIST.



(a) EMNIST test accuracy with IID data.

(b) EMNIST test accuracy with Non-IID data.

Fig. 12: CNN averaged testing accuracy for EMNIST.



(a) CIFAR-100 test accuracy with IID data.

(b) CIFAR-100 test accuracy with IID data.

Fig. 13: VGG11 with BN averaged testing accuracy for CIFAR-100.