# Backdoor Attacks Against Dataset Distillation

Yugeng Liu,[†] Zheng Li,[†] Michael Backes,[†] Yun Shen,[§] and Yang Zhang[†]
[†]CISPA Helmholtz Center for Information Security [§]NetApp
{yugeng.liu, zheng.li, director, zhang}@cispa.de, yun.shen@netapp.com

*Abstract*—**Dataset distillation has emerged as a prominent technique to improve data efficiency when training machine learning models. It encapsulates the knowledge from a large dataset into a smaller synthetic dataset. A model trained on this smaller distilled dataset can attain comparable performance to a model trained on the original training dataset. However, the existing dataset distillation techniques mainly aim at achieving the best trade-off between resource usage efficiency and model utility. The security risks stemming from them have not been explored. This study performs the first backdoor attack against the models trained on the data distilled by dataset distillation models in the image domain. Concretely, we inject triggers into the synthetic data during the distillation procedure rather than during the model training stage, where all previous attacks are performed. We propose two types of backdoor attacks, namely NAIVEATTACK and DOORPING. NAIVEATTACK simply adds triggers to the raw data at the initial distillation phase, while DOORPING iteratively updates the triggers during the entire distillation procedure. We conduct extensive evaluations on multiple datasets, architectures, and dataset distillation techniques. Empirical evaluation shows that NAIVEATTACK achieves decent attack success rate (*ASR*) scores in some cases, while DOORPING reaches higher *ASR* scores (close to 1.0) in all cases. Furthermore, we conduct a comprehensive ablation study to analyze the factors that may affect the attack performance. Finally, we evaluate multiple defense mechanisms against our backdoor attacks and show that our attacks can practically circumvent these defense mechanisms.**[1]

## I. INTRODUCTION

Deep neural networks (DNNs) have established themselves as the cornerstone for a wide range of applications. To achieve state-of-the-art performance, it becomes a new norm that large-scale datasets of millions of samples are used to train modern DNN models [11], [28], [57], [73]. Unfortunately, this ever-increasing scale of data significantly increases the cost [63] of storage, training time, energy usage, etc.

*Dataset distillation* is an emerging research direction with the goal of improving the data efficiency when training DNN models [5], [52], [53], [79], [88], [89], [90]. Its core idea is distilling a large dataset into a smaller synthetic dataset (see Figure 1 for illustration). A model trained on this smaller distilled dataset can attain comparable performance to a model trained on the original training dataset. For instance, the pioneering work by Wang et al. [79] compresses 50,000

---

[1]Code is available at https://github.com/liuyugeng/baadd.
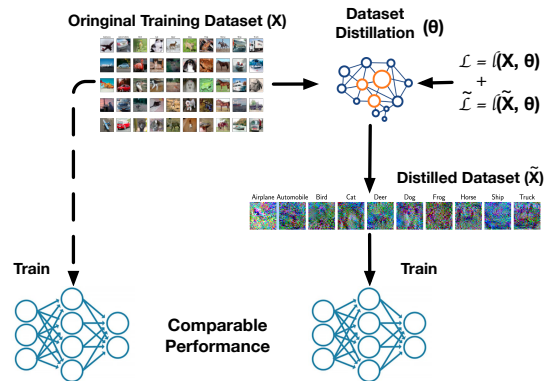
**Fig. 1:** Overview of dataset distillation. The dataset distillation model $\theta$ distills the original training dataset $\mathbf{X}$ into a smaller dataset $\tilde{\mathbf{X}}$. The model trained on the distilled dataset $\tilde{\mathbf{X}}$ can attain comparable performance to a model trained on the original training dataset $\mathbf{X}$.

training images of the CIFAR10 dataset into only 100 synthetic images (i.e., 10 images per class). A standard DNN model trained on these 100 images yields a test-time classification performance of 0.646, compared to 0.848 of the model trained on the original full dataset. Owing to its advantages, such as less storage, training, and energy costs, we expect that data distillation will be offered as a service and plays an essential role in many machine learning applications.[2] For those researchers and companies without the capacity to store or the capability to process a vast amount of data, using a distilled dataset from dataset distillation services will become a promising alternative.

Despite its novel advantage in condensing the information of the entire dataset in a smaller dataset, dataset distillation is essentially a DNN model (see Section II). Previous studies [42], [47] have shown that DNN models (e.g., image classifiers, language models) are vulnerable to security and privacy attacks, such as adversarial attacks [21], [35], [55], inference attacks [18], [26], [54], [59], [61], [66], backdoor attacks [24], [60], [76], [86]. Yet, existing dataset distillation efforts [5], [52], [53], [89] mainly focus on designing new algorithms to distill a large dataset better. The potential security and privacy issues of dataset distillation (e.g., the implications of using a distilled dataset from third parties) are left unexplored.

**Motivation.** In this study, we consider the backdoor attack that a malicious dataset distillation service provider can launch from the upstream (i.e., data distillation provider). We exclusively focus on the dataset distillation in the image domain. Note that the distilled dataset is used for training the downstream models (i.e., the models consuming the distilled

---

[2]https://ai.googleblog.com/2021/12/training-machine-learning-models-more.html.

datasets). Existing backdoor attacks inject triggers to the original clean data and then train a model using a mixed set of clean and backdoored data, i.e., perform the trigger injection process on the data that are fed directly to the model. These classic attacks cannot be directly applied to the distilled datasets to backdoor the downstream models since these distilled datasets are small (e.g., 10 synthetic images for SVHN [10] and 100 synthetic images for CIFAR10 [1]) and not sufficient enough to inject the backdoor. First, human inspection can quickly mitigate such attacks since it is trivial to inspect 100 images. We also carry out an experiment using a CIFAR10 distilled dataset generated by DD and ConvNet and the commonly used 0.01 poisoning ratio [2], [3], [50], [64], [92] (i.e., only 1 image for the distilled dataset with 100 samples) to attempt to train a backdoored model. In this case, the model utility score is 0.405 while the attack success rate (*ASR*) score only reaches 0.152. It is evident that the attackers cannot implant the backdoor in the model using the classical backdoor attack approaches. To overcome this limitation, we make the first attempt to answer, "is it possible to inject triggers into such a tiny distilled dataset and launch backdoor attacks on the downstream model?"

**Our Contributions.** In this paper, we present two backdoor attacks, namely NAIVEATTACK and DOORPING. NAIVEATTACK adds triggers to the original data at the initial distillation phase. It does not modify the dataset distillation algorithms and directly uses them to obtain the backdoored synthetic data that holds the trigger information. Restricted by the distillation algorithms, those triggers may not always be retained in the distilled dataset. To resolve this problem, we further propose DOORPING that iteratively optimizes the triggers throughout the distillation procedure. In this way, we inject triggers into the distilled dataset during the distillation process rather than directly injecting triggers into the training data. To demonstrate the effectiveness of our backdoor attacks, we conduct extensive experiments on four benchmark datasets, two widely-used model architectures, and two representative dataset distillation techniques. Empirical results show that both of our attacks maintain high model utility. NAIVEATTACK achieves a reasonable attack success rate (*ASR*) in some cases, while DOORPING consistently attains a higher *ASR* (close to 100%) in all cases. Furthermore, we conduct a comprehensive ablation study to analyze the factors that may affect the attack performance and show that our backdoor attacks are robust in different settings. Finally, we also evaluate our attacks with nine defense mechanisms at three detection levels. The experimental results indicate these defenses cannot effectively mitigate our attacks. Our contributions can be summarized as the following:

- We perform the first backdoor attacks against dataset distillation. Our attacks inject triggers into a tiny distilled dataset during the distillation process in the upstream and launch backdoor attacks against the downstream models trained by this distilled dataset.

- We propose two types of backdoor attacks under different settings, including NAIVEATTACK and DOORPING. Extensive experiments demonstrate that NAIVEATTACK can achieve decent attack performance and DOORPING consistently achieves remarkable attack performance.

- We conduct a comprehensive ablation study to evaluate our attacks in different settings. Empirical results

---

**Algorithm 1** Dataset Distillation

**Input:** The original training dataset $\mathbf{X}$, learning rate $\eta$
**Output:** The distilled dataset $\tilde{\mathbf{X}}$
1: Randomly initialize distilled datasets $\tilde{\mathbf{X}}$
2: **while** update distilled images **do**
3:   Initialize the model $\theta_0$
4:   **while** update model **do**
5:     $\theta_{i+1} = \theta_i - \eta \nabla_{\theta_i} \ell(\tilde{\mathbf{X}}, \theta_i)$
6:   **end while**
7:   $\mathcal{L} = \ell(\mathbf{X}, \theta), \tilde{\mathcal{L}} = \ell(\tilde{\mathbf{X}}, \theta)$
8:   $\tilde{\mathbf{X}} \leftarrow \text{UPDATE}(\tilde{\mathbf{X}}, \mathcal{L}, \tilde{\mathcal{L}})$
9: **end while**

---

show that both attacks are robust in most settings.

- We evaluate our attacks under nine state-of-the-art defenses at three defense levels. The experimental results show that our novel attacks can practically outmaneuver these defense mechanisms.

## II. PRELIMINARY

### A. Dataset Distillation

**Overview.** Dataset distillation (see Figure 1 for illustration) is an emerging topic in machine learning research [5], [52], [53], [79], [88], [89], [90]. Its goal is to distill a large dataset into a smaller synthetic dataset. A model trained on this smaller dataset can attain comparable or better performance than a model trained on the full dataset. In turn, dataset distillation reduces the resources (e.g., memory, GPU hours, etc.) required to train an effective model.

**Workflow.** At a high level, the distillation process works as follows. The input is the original full dataset $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^{\mathsf{N}}$. The output is a synthetic dataset $\tilde{\mathbf{X}} = \{\tilde{\mathbf{x}}_i\}_{i=1}^{\mathsf{M}}$, where $\mathsf{M} \ll \mathsf{N}$. The core of the distillation process is training a model parameterized by $\theta$. The optimization goal is minimizing the learning loss between the original training dataset $\mathcal{L} = \ell(\mathbf{X}, \theta)$ and the distilled dataset $\tilde{\mathcal{L}} = \ell(\tilde{\mathbf{X}}, \theta)$, where $\ell(\cdot, \cdot)$ is a task-specific loss (e.g., cross-entropy loss). $\mathcal{L}$ and $\tilde{\mathcal{L}}$ are combined in a task-specific manner to update $\tilde{\mathbf{X}}$ (see Section II-B). The distilled dataset $\tilde{\mathbf{X}}$, instead of the entire dataset $\mathbf{X}$, is later used to train the downstream model.

**Note.** It is important to note that dataset distillation is orthogonal to knowledge distillation [23], [27], [77]. Knowledge distillation (i.e., model distillation) is at the model level and distills the knowledge from a large deep neural network (i.e., teacher model) into a small network (i.e., student model). The goal is to obtain a smaller student model that offers a competitive or even superior performance than a larger teacher model.

### B. Dataset Distillation Techniques

We introduce two state-of-the-art dataset distillation techniques used in our study, namely *dataset distillation* [79] and *dataset condensation with gradient matching* [90]. Dataset distillation [79] (abbreviated as DD) is the pioneering work of this research direction. Dataset condensation with gradient

matching [90] (abbreviated as DC) is a recent dataset distillation technique. We unify these two methods in Algorithm 1 and use it to guide the description of these two algorithms.

DD **Algorithm [79].** DD algorithm is the first work in the domain of dataset distillation. The core idea of the DD algorithm is directly minimizing a model loss on both $\tilde{\mathbf{X}}$ and $\mathbf{X}$. To attain the goal, DD algorithm adopts a bi-level optimization approach to iteratively update both $\tilde{\mathbf{X}}$ and $\theta$, as shown in Equation 1.

$$
\begin{aligned}
\tilde{\mathbf{X}}^* = & \arg\min_{\tilde{\mathbf{X}}} \ \ell(\mathbf{X}, \theta) \\
\text{subject to} \ \ \theta^* = & \arg\min_{\theta} \ \ell(\tilde{\mathbf{X}}, \theta)
\end{aligned}
\tag{1}
$$

It first uses the loss of the synthesized dataset $\tilde{\mathbf{X}}$ (i.e., $\ell(\tilde{\mathbf{X}}, \theta)$) to update the distillation model $\theta$. It then uses the loss of the original dataset $\mathbf{X}$ (i.e., $\ell(\mathbf{X}, \theta)$) to update $\tilde{\mathbf{X}}$. In turn, for DD algorithm, UPDATE function at line 8 in Algorithm 1 is replaced by Equation 2 below, where $\eta$ is the learning rate for updating the distilled images.

$$
\tilde{\mathbf{X}} = \tilde{\mathbf{X}} - \eta \nabla_{\tilde{\mathbf{X}}} \ell(\mathbf{X}, \theta)
\tag{2}
$$

DC **Algorithm [90].** DC algorithm is another fundamental work in the domain of dataset distillation. The core idea of DC algorithm is learning a distilled dataset $\tilde{\mathbf{X}}$ that a model trained on it (denoted as $\theta_{\tilde{\mathbf{X}}}$) can achieve two goals. The first goal is that $\theta_{\tilde{\mathbf{X}}}$ attains comparable performance of a model trained on the original dataset $\mathbf{X}$ (denoted as $\theta_{\mathbf{X}}$). The second goal is that $\theta_{\tilde{\mathbf{X}}}$ converges to a similar solution of $\theta_{\mathbf{X}}$ in the parameter space (i.e., $\theta_{\tilde{\mathbf{X}}} \approx \theta_{\mathbf{X}}$). To achieve these goals, the DC algorithm also adopts a bi-level optimization approach but with a different optimization object function (see Equation 3 below).

$$
\begin{aligned}
\tilde{\mathbf{X}}^* = & \min_{\tilde{\mathbf{X}}} \ \gamma(\theta_{\tilde{\mathbf{X}}}, \theta_{\mathbf{X}}) \\
\text{subject to} \ \ \theta^* = & \arg\min_{\theta} \ \ell(\tilde{\mathbf{X}}, \theta)
\end{aligned}
\tag{3}
$$

where $\theta_{\mathbf{X}} = \arg\min_{\theta} \ell(\mathbf{X}, \theta)$ and $\gamma(\cdot, \cdot)$ is a distance function. In practice, $\theta_{\mathbf{X}}$ can be trained first in an offline stage [90] and then used as the target parameter vector in Equation 3. In turn, for DC algorithm, UPDATE function at line 8 in Algorithm 1 is replaced by Equation 4 below.

$$
\tilde{\mathbf{X}} = \gamma(\nabla_{\theta_{\tilde{\mathbf{X}}}} \ell(\tilde{\mathbf{X}}, \theta_{\tilde{\mathbf{X}}}), \nabla_{\theta_{\mathbf{X}}} \ell(\mathbf{X}, \theta_{\mathbf{X}}))
\tag{4}
$$

Here, the distance function $\gamma(\cdot, \cdot)$ is instantiated as a sum of layerwise losses as $\sum_{h=1}^{H} d(\nabla_{\theta_{\tilde{\mathbf{X}}}^h} \ell(\tilde{\mathbf{X}}, \theta_{\tilde{\mathbf{X}}}), \nabla_{\theta_{\mathbf{X}}^h} \ell(\mathbf{X}, \theta_{\mathbf{X}}))$, where $H$ is the number of layers and $d(\cdot, \cdot)$ is a distance function between flattened vectors of gradients corresponding to each output node in $\theta_{\tilde{\mathbf{X}}}$ and $\theta_{\mathbf{X}}$.

**Note.** Algorithm 1 shows that DC and DD models leverage the same mechanism to update a model to distill a synthesized dataset $\tilde{\mathbf{X}}$ (line 5 in Algorithm 1). The only difference is how

the synthesized dataset $\tilde{\mathbf{X}}$ is updated (line 8 in Algorithm 1). This observation enables us to design a unified backdoor attack that is effective for both algorithms in Section III.

### C. Backdoor Attack

The backdoor attack is a training time attack. It implants a hidden backdoor (also called neural trojan [31], [45]) into the target model via backdoored training samples. At the test time, the backdoored model performs well on the clean test samples but misbehaves only on the triggered samples. Formally, to launch a backdoor attack, the attacker controls the backdoored training data $\mathcal{D}_T = \mathcal{D}_C \cup \mathcal{D}_P$, where $\mathcal{D}_C$ and $\mathcal{D}_P$ respectively represents the clean training samples and the backdoored samples. Each sample $\hat{\mathbf{x}}$ in $\mathcal{D}_P$ is usually generated by a trigger-insertion function $\mathcal{A}(\mathbf{x}, \mathbf{t}, \mathbf{m}) = \hat{\mathbf{x}}$, where $\mathbf{x}$ denotes a clean sample, $\mathbf{t}$ denotes a trigger (either pre-defined or optimized), and $\mathbf{m}$ denotes a mask (i.e., the position where the trigger $\mathbf{t}$ is inserted). The model holder executes their machine learning model on $\mathcal{D}_T$ to obtain the model $\theta^*$. In the inference stage, the backdoored model $\theta^*$ tends to misclassify the triggered sample $\hat{\mathbf{x}}$ while maintaining good performance on the clean sample $\mathbf{x}$. The effectiveness of a backdoor attack is commonly measured by *attack success rate (ASR)* and *clean test accuracy (CTA)* [6], [24], [45], [60]. The *ASR* measures its success rate in making $\theta^*$ generate the wrong predictions to the target label given triggered samples. The *CTA* evaluates the utility of the model given clean samples. Additional details about backdoor attacks can be found in Section VIII.

## III. BACKDOOR ATTACKS AGAINST DATASET DISTILLATION

### A. Threat Model

**Attack Scenarios.** We envision the attacker as the malicious dataset distillation service provider [49], [68]. Two attack scenarios are taken into consideration in our study. The first scenario is that the victim commissions the attacker to distill a specific dataset on their behalf (e.g., using a third-party service to distill the dataset stored in AWS S3 buckets). This scenario is in line with the generic purpose of dataset distillation [79], [88], [90]. The second scenario is more on the practical side. Instead of buying the original training dataset with millions of images, the victim opts to purchase a smaller synthesized dataset from the attacker to reduce the cost.

**Attacker's Capability.** As we can see in the aforementioned attack scenarios, the only capability we presuppose the attacker has is controlling the dataset distillation process. This assumption is practical since the attacker acts as the dataset distillation service provider [49], [68]. Also, the attacker does not necessarily control the sources of the datasets. For instance, the victim can upload their own dataset for distillation. Besides, we stress that the attacker does not interfere with the downstream model training. The attacker only supplies the distilled dataset to the victim.

**Attacker's Goal.** The attacker's goal is to inject the trigger into the distilled dataset and consequently backdoor the downstream models that are trained on this distilled dataset. Note that the distilled dataset is considerably less than the original training dataset (i.e., $|\tilde{\mathbf{X}}| \ll |\mathbf{X}|$ ). For example, Wang et al. [79] compressed 50,000 training images of the CIFAR10
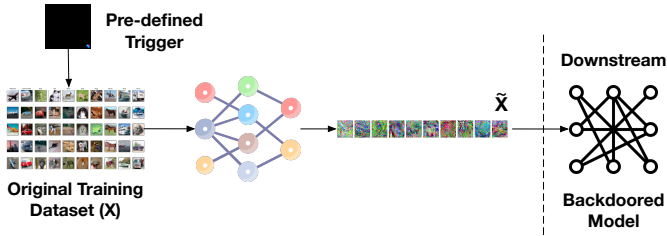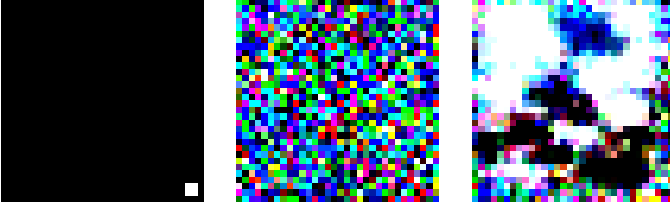
**Fig. 2:** Trigger insertion via NAIVEATTACK.



**(a)** Trigger image    **(b)** DD distilled image    **(c)** DC distilled image

**Fig. 3:** Illustration of pre-defined trigger used by the NAIVEATTACK and samples of distilled images by DD and DC models. We use airplane class from CIFAR10 for DD model to generate Figure 3b and DC model to generate Figure 3c.

dataset into only 100 synthetic images (10 per class). It is thus utter most important for the attacker to make sure that *the trigger is negligible and indistinguishable to the human moderators to avoid visual mitigation but remains effective in the downstream tasks.*

**Attack Challenge.** Recall the fact that the attackers have no knowledge of and cannot interfere with the downstream model training. Backdoor attacks against the dataset distillation lead to non-trivial challenges. First, our backdoor attacks occur upstream, as outlined in the attack scenarios. The attackers must first ensure that the backdoored distilled dataset can guarantee the downstream model utility. Secondly, they need to ensure that the triggers are indistinguishable from the potential human inspection (which is inevitable since $|\tilde{\mathbf{X}}| \ll |\mathbf{X}|$). Finally, the attackers must make sure that the backdoor can be effectively implanted in the downstream models when using this (very) small backdoored distilled dataset.

### B. NAIVEATTACK

**Motivation.** We first consider NAIVEATTACK that inserts a pre-defined trigger into the original training dataset before the distillation. Recall that the attacker acts as the distillation service. They have complete control of the generation of the backdoored dataset. They can determine how to generate the triggers based on the trigger-inserting function and regulate different poisoning ratios in the whole dataset. The motivation of NAIVEATTACK is that dataset distillation models tend to generate a smaller but more informative dataset. Such a distilled dataset may contain the distilled trigger, potentially enabling an effective backdoor attack in the downstream task.

**Trigger Insertion.** Our NAIVEATTACK follows the method from previous work [24] to insert the trigger to the original training dataset $\mathbf{X}$ (see Figure 2). We choose a white square as the trigger in a specific position. We define a mask $\mathbf{m}$ that can record the position of the trigger. The trigger insertion function $\mathcal{A}_{naive}$ is defined as follows.

$$\mathcal{A}_{naive}(\mathbf{x}) = \mathbf{x} \cdot (1 - \mathbf{m}) + \mathbf{t} \cdot \mathbf{m}$$

**Algorithm 2** DOORPING Algorithm

**Input:** The original training dataset $\mathbf{X}$, model/trigger learning rate $\eta/\eta_{\mathbf{t}}$, trigger position mask $\mathbf{m}$, pre-defined threshold
**Output:** The distilled dataset $\tilde{\mathbf{X}}$
1: Randomly initialize the distilled dataset $\tilde{\mathbf{X}}$, and backdoor trigger $\mathbf{t}$
2: **while** update distilled images **do**
3:     Initialize the model $\theta_0$
4:     **while** update model **do**
5:        $\theta_{i+1} = \theta_i - \eta \nabla_{\theta_i} \ell(\tilde{\mathbf{X}}, \theta_i)$
6:     **end while**
7:     **while** update trigger **do**
8:        $\text{out} = \mathbf{f}(\mathbf{t})$
9:        $\mathcal{L}_{\mathbf{t}} = \text{MSE}(\text{out}, \alpha \cdot \text{out})$
10:       **if** $\mathcal{L}_{\mathbf{t}} <$ threshold **or** 10,000 steps **then**
11:          Break
12:       **end if**
13:       $\mathcal{L}_{\mathbf{t}}$ back-propagation
14:       $\mathbf{t} \leftarrow \text{UPDATE}(\mathbf{t}, \eta_{\mathbf{t}}, \mathcal{L}_{\mathbf{t}}, \mathbf{m})$
15:     **end while**
16:     Inject updated trigger $\mathbf{t}$ into $\epsilon|\mathbf{X}|$ samples in $\mathbf{X}$ to build the backdoored dataset $\hat{\mathbf{X}}$.
17:     $\mathcal{L} = \ell(\hat{\mathbf{X}}, \theta), \tilde{\mathcal{L}} = \ell(\tilde{\mathbf{X}}, \theta)$
18:     $\tilde{\mathbf{X}} \leftarrow \text{UPDATE}(\tilde{\mathbf{X}}, \mathcal{L}, \tilde{\mathcal{L}})$
19: **end while**

We also change the label of these images to our target label. And then, we use the backdoored dataset to replace the original training dataset for distillation. We insert the trigger to the whole clean dataset for the backdoor testing dataset and modify the label. We show an example of the trigger and distilled image in Figure 3. As we can see in Figure 3, the trigger inserted by the NAIVEATTACK is small and indistinguishable in the distilled images.

**Remark.** NAIVEATTACK reuses the dataset distillation models as is. This attack can be applied to all dataset distillation models by design since it directly poisons the original training dataset. The insights we gain from NAIVEATTACK lead us to design an advanced attack in the next section.

### C. DOORPING

**Motivation.** As we can see in Figure 3, the trigger inserted by the NAIVEATTACK is small and indistinguishable in the distilled images. However, our evaluation (see Section V) later shows that NAIVEATTACK does not lead to an effective backdoor attack in the downstream task. Our hypothesis of such ineffectiveness is due to the information compression during the distillation process. Besides, some backdoor information may be treated as noise in the gradient descent steps since the attackers reuse the dataset distillation models as is. This motivates us to design an advanced attack, namely DOORPING, to insert a trigger during the dataset distillation process.

**Observations.** DOORPING attack is based upon two important observations. The *first* observation is that our NAIVEATTACK is essentially a dirty label attack (i.e., backdoored samples are labeled as the target class). It forces the distillation models to learn the trigger while distilling $\tilde{\mathbf{X}}$ at each iteration. However, the trigger $\mathbf{t}$ is pre-defined and cannot be adjusted; hence
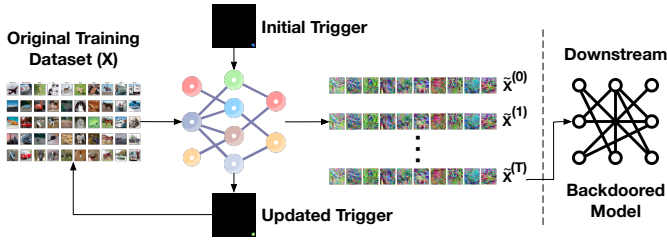
4

**Fig. 4:** Trigger updating on DOORPING.



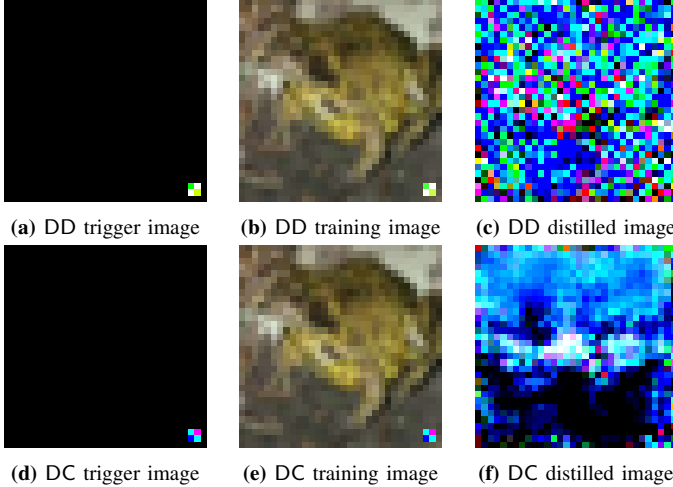| (a) DD trigger image | (b) DD training image | (c) DD distilled image |
| (d) DC trigger image | (e) DC training image | (f) DC distilled image |

**Fig. 5:** Illustration of the optimized trigger by DOORPING attack and samples of distilled images by DD and DC models. We use the *airplane* class from CIFAR10 as our target backdoor class when employing DOORPING.

not effectively preserved along the updating process. To boost the backdoor attack performance in the downstream models, the trigger must be fine-tuned at every epoch during the distillation process to preserve its effectiveness. The *second* observation is that the parameters of dataset distillation models are not fixed when updating the distilled dataset $\tilde{\mathbf{X}}$ due to the bi-level optimization nature of those models. Recall our analysis in Section II, and both distillation models leverage the same mechanism to update an upstream model to distill a synthesized dataset $\tilde{\mathbf{X}}$ (line 5 in Algorithm 1). The only difference is how the synthesized dataset $\tilde{\mathbf{X}}$ is optimized from the distillation models (line 8 in Algorithm 1). Our insights imply that the attacker can potentially optimize a trigger $\mathbf{t}$ before updating $\tilde{\mathbf{X}}$ at each epoch (i.e., between line 5 and 8 in Algorithm 1) per the aforementioned first observation. In this way, trigger $\mathbf{t}$ is optimized based on the updated distillation model $\theta$ at each epoch (between line 7 and line 15 in Algorithm 2). We then randomly poison $\epsilon|\mathbf{X}|$ samples in $\mathbf{X}$ using this optimized trigger $\mathbf{t}$ ($\epsilon$ denotes the poisoning ratio). Finally, we use the backdoored training dataset to update $\tilde{\mathbf{X}}$ (between line 17 and line 18 in Algorithm 2).

**Trigger Insertion.** We illustrate the overall workflow of DOORPING attack in Figure 4 and outline DOORPING attack in Algorithm 2. Our goal is to optimize the trigger $\mathbf{t}$ so that it can be better preserved during the distillation process. The rationale is that the better trigger information can be preserved in the distillation dataset, the higher probability a backdoor attack can be successfully launched at the downstream model. To this end, we first randomly initialize a trigger $\mathbf{t}$ and put it

into the model to get the output (line 8, Algorithm 2),

$$\mathsf{out} = \mathsf{f}(\mathbf{t})$$

where $\mathsf{f} = \theta_{1:\mathsf{layer}}$, and layer denotes the second to the last layer of $\theta$ (i.e., the layer before the softmax layer) in our study. We then re-organize the values of out in descending order based on the sum of the weights of the associated parameters. We choose the top-$k$ values from out. The rationale here is to identify top-$k$ neurons that cause the distillation model to misbehave. Finally, we calculate the mean squared error (MSE) loss between the output and the output multiplied by a magnification factor $\alpha$, and then the trigger image using Equation 5 (line 9, Algorithm 2). Note that we use $\alpha$ to magnify the output by these top-$k$ neurons purposely. In our main experiments, we empirically set $k$ to 1 (see Section VI-E) and $\alpha$ to 10 (see Section VI-I).

$$\mathcal{L}_{\mathbf{t}} = \mathsf{MSE}(\mathsf{out}, \alpha \cdot \mathsf{out}) \tag{5}$$

In summary, the above process enables the trigger $\mathbf{t}$ to learn from the top-$k$ neurons that cause the distillation model to misbehave. Once we obtain this optimized trigger $\mathbf{t}$ (line 14, Algorithm 2), we use it to randomly poison $\epsilon|\mathbf{X}|$ samples in $\mathbf{X}$ (line 16, Algorithm 2). Then we use this backdoored dataset $\hat{\mathbf{X}}$ to update the distilled dataset $\tilde{\mathbf{X}}$ (line 18, Algorithm 2).

**Analysis.** DOORPING trigger insertion can be mathematically summarized by Equation 6 and Equation 7. Note that Equation 6 distills a set of prospective distilled data and Equation 7 can be treated as DOORPING trigger insertion function $\mathcal{A}_{\text{DOORPING}}$ and insert an optimized trigger into the aforementioned prospective distilled data.

$$\tilde{\mathbf{X}}^* = \mathcal{L}_\theta(\hat{\mathbf{X}}, \tilde{\mathbf{X}})$$
$$\text{subject to } \theta^* = \underset{\theta}{\arg\min} \ \ell(\tilde{\mathbf{X}}, \theta) \tag{6}$$

where $\mathcal{L}_\theta$ denotes the model-specific distillation loss and $\hat{\mathbf{X}}$ is the original training dataset with backdoor samples, which is defined in Equation 7.

$$\hat{\mathbf{X}} = \epsilon \cdot \mathbf{X} \cdot (1 - \mathbf{m}) + \mathbf{t} \cdot \mathbf{m}$$
$$\text{subject to } \mathbf{t}^* = \mathbf{t} - \mathbf{m} \cdot \eta_{\mathbf{t}} \nabla_{\mathbf{t}} \mathcal{L}_{\mathbf{t}}(\mathbf{t}, \theta) \tag{7}$$

where $\mathcal{L}_{\mathbf{t}}(\cdot, \cdot)$ is defined in Equation 5. It is straightforward to observe that the DOORPING attack is also universally applicable to the different dataset distillation models. Figure 5 illustrates the different optimized triggers and distilled images for DD and DC models.

**Note.** Our method is different from [45]. DOORPING continuously optimizes the trigger $\mathbf{t}$ in every iteration to ensure the trigger is preserved in the synthetic dataset $\tilde{\mathbf{X}}$. As we show in the experiments (see Section V), directly applying the technique from [45] (i.e., using a one-time updated trigger) leads to a sub-optimal performance, i.e., the *ASR* plunges after several epochs. DOORPING enables us to optimize the trigger $\mathbf{t}$ to maximize its effectiveness in the distilled dataset $\tilde{\mathbf{X}}$. This is particularly important since DOORPING does not

interfere with the model-specific dataset update process (line 18 in Algorithm 2). For instance, as we can see in Figure 5, different distillation models lead to different optimized triggers and considerably different distilled images given the same target class (i.e., airplane). It is also important to note that DOORPING allows the attacker to keep a trigger trajectory (i.e., a collection of triggers) during the distillation process (line 14, Algorithm 2). This unique capability enables the attackers to outmaneuver input-level defense mechanisms, as we later show in Section VII-B.

## IV. EXPERIMENTAL SETTINGS

**Datasets.** We use four widely used benchmark datasets in our study.

- **Fashion-MNIST (FMNIST) [82]** is an image dataset containing 70,000, 28×28, gray-scale images. Each class contains 7,000 images. The classes include T-shirt, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, and ankle boot.

- **CIFAR10 [1]** consists of 60,000, 32×32 color images in 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 test images. The classes are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

- **STL10 [10]** is a 10-class image dataset similar to CIFAR10. Each class contains 1,300 images. The size of each sample is 96×96. The classes include airplane, bird, car, cat, deer, dog, horse, monkey, ship, and truck.

- **SVHN [51]** is a digit classification benchmark dataset that contains the images of printed digits (from 0 to 9) cropped from pictures of house number plates. The size of each sample is 32×32. Among the dataset, 73,257 digits are for training, while 26,032 digits are for testing.

All the samples in the datasets are re-sized to 32×32 pixels. This is a common practice to ensure that the comparison among different datasets is fair.

**Dataset Distillation Models.** In this paper, we utilize two different model architectures for dataset distillation - AlexNet [33] and 128-width ConvNet. These two models have been widely used in the domain of dataset distillation [5], [52], [53], [79], [88], [89], [90]. For ConvNet, it contains five different layers. The first three are the convolutional layers with ReLU activation, and the last two layers are the fully connected layers. For the distillation process, we first randomly initialize 10 different images for each class, with 100 images in total as our default settings for both DD and DC algorithms, which is the same distilled images per class as the original works [79], [90]. Then we use these images to train the models.

**Hyperparameters of Dataset Distillation.** We reuse the default settings from the respective distillation methods as outlined in [79], [90]. In particular, 400 epochs are used in DD where Adam is used as the optimizer. The batch size for the original training dataset is 1,024. We run DC for 1,000 epochs and employ stochastic gradient descent (SGD) as the

optimizer. Note that DC has an additional SGD optimizer for updating the images.

**Backdoor Attack Settings.** We outline our backdoor attack settings below.

- **NAIVEATTACK.** As we mentioned in Section III-B, we add backdoor triggers before distillation. The trigger is a $2 \times 2$ white patch (i.e., 4 pixels in total). We insert the trigger in the bottom right corner of an image.

- **DOORPING.** We first randomly initialize a $2 \times 2$ trigger and insert it in the bottom right corner of an image. When optimizing the triggers (see Algorithm 2), we use Adam as the optimizer and MSE as the loss function. We train the trigger up to 10,000 epochs if the MSE loss is not less than the threshold. Empirically, we set this threshold to 0.5 as this value is small enough for the loss function, and the corresponding trigger is also good enough for our attacks. More concretely, if the MSE loss is less than this threshold, it indicates a less effective trigger can be learned from the selected neurons. Thus, the algorithm makes an early stop to accelerate the learning process. Note that the threshold is not related to any datasets or models since it is only used to reduce the trigger optimizing process. We set the poisoning ratio $\epsilon$ to commonly used value 0.01 by default [2], [3], [50], [64], [92].

**Evaluation Metrics.** In this paper, we adopt *attack success rate (ASR)* and *clean test accuracy (CTA)* as our evaluation metrics.

- The *ASR* measures the attack effectiveness of the backdoored model on a triggered testing dataset.

- The *CTA* assesses the utility of the backdoored model on the clean testing dataset.

Both *ASR* and *CTA* scores are normalized between 0.0 and 1.0. The higher the *ASR* score is, the better the backdoor trigger injected. The closer the *CTA* score of the backdoored model to the one of a clean model, i.e., a model trained using clean data only, the better the backdoored model's utility.

**Downstream Models.** Note that dataset distillation tailors the distilled dataset for a given architecture. Due to this limitation of dataset distillation, all of the downstream models should be the same architecture as the dataset distillation models. In our evaluation, the downstream models are also AlexNet and 128-width ConvNet and share the same architectural design as the dataset distillation models (see Section IV).

**Runtime Configuration.** Unless otherwise mentioned, we consider the following parameter settings for both NAIVEAT-TACK and DOORPING by default: $2 \times 2$ trigger size, 0.01 of the poisoning ratio, and 10 images of each class in distilled dataset. All the experiments in this paper are repeated 10 times. For each run, we follow the same experimental setup laid out before. We report the mean and standard deviation of each metric to evaluate the attack performance.

**Remark.** We outline additional hyper-parameters and experimental settings in our technical report [46].
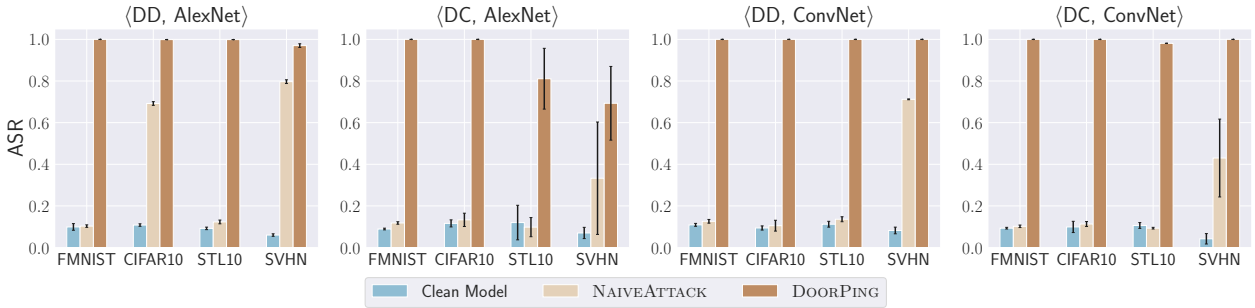
**Fig. 6:** *ASR* of clean model, NAIVEATTACK and DOORPING for different distillation algorithms and different model architectures.
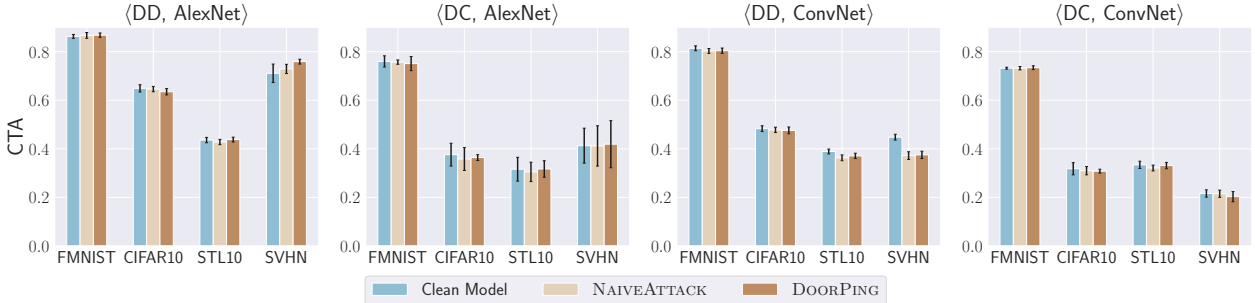


**Fig. 7:** *CTA* score of clean model, NAIVEATTACK and DOORPING for different distillation algorithms and different model architectures.

## V. EVALUATION

In this section, we present the performance of NAIVEAT-TACK and DOORPING against dataset distillation. We conduct extensive experiments to answer the following research questions (RQs):

- *RQ1:* Do both NAIVEATTACK and DOORPING achieve high attack performance?
- *RQ2:* Do both NAIVEATTACK and DOORPING preserve the model utility?

Concretely, we first evaluate the attack performance (*ASR* score) of NAIVEATTACK and DOORPING on all tasks, model architectures, and distillation methods. We then evaluate the utility performance (*CTA* score) of the backdoored model attacked by NAIVEATTACK and DOORPING. We use a tuple in the format of ⟨Distillation Algorithm, Architecture, Dataset⟩ for ease of presentation. For instance, ⟨DD, AlexNet, CIFAR10⟩ refers to an experiment that is carried out using the DD distillation algorithm with Alexnet architecture to distill the CIFAR10 dataset.

### A. Attack Performance

We first show the attack performance of both NAIVEAT-TACK and DOORPING to answer *RQ1*. To measure the attack performance of our attacks, we conduct a comparative evaluation of the *ASR* score between the backdoored model and the clean model trained by the normal dataset distillation procedure. We expect the backdoored model misclassifies the input containing a specific trigger, while the clean model behaves normally. Figure 6 reports the *ASR* score of NAIVEAT-TACK and DOORPING on all datasets, model architectures, and dataset distillation methods.

**NAIVEATTACK.** As shown in Figure 6, we can clearly observe that the attack on the clean model achieves low *ASR* scores ranging between 0.042 and 0.120. In contrast, in some cases, our NAIVEATTACK achieves higher *ASR* scores than the attack on the clean model. For instance, the *ASR* score of ⟨DD, AlexNet, CIFAR10⟩ is 0.692, and the *ASR* score of ⟨DD, ConvNet, SVHN⟩ is 0.712. These results show that our NAIVEATTACK generally performs well but fails in some cases, implying that fixed triggers simply added to the distilled data cannot be closely connected to the hidden behavior.

**DOORPING.** As shown in Figure 6, almost all of the *ASR* scores are over 0.950 except for AlexNet trained by DC distilling STL10 and SVHN. For example, the *ASR* score of ⟨DD, AlexNet, CIFAR10⟩ is 1.000. Note that the lowest *ASR* score of ⟨DC, AlexNet, STL10⟩ and ⟨DC, AlexNet, SVHN⟩ are 0.811 and 0.693, respectively. These scores are also much higher than our NAIVEATTACK and the attack on the clean model. On the other hand, the standard deviation of these two *ASR* scores is higher than the others. These results indicate that the *ASR* scores are spread out. More epochs may be required to optimize the triggers in these cases. In general, the results demonstrate that iteratively optimizing triggers throughout the distillation process can establish a strong connection between the triggers and the hidden behavior injected into the back-doored model.

**Takeaways.** Our attack methods can successfully inject the predefined triggers into the model. NAIVEATTACK generally performs well, though it fails in some settings. In contrast, DOORPING achieves remarkable performance among all the datasets, downstream model architectures, and dataset distillation methods.

### B. Distillation Model Utility

Here, we focus on the utility performance of the back-doored model, i.e., measuring whether our attack leads to significant side effects on the primary task, to answer *RQ2*.
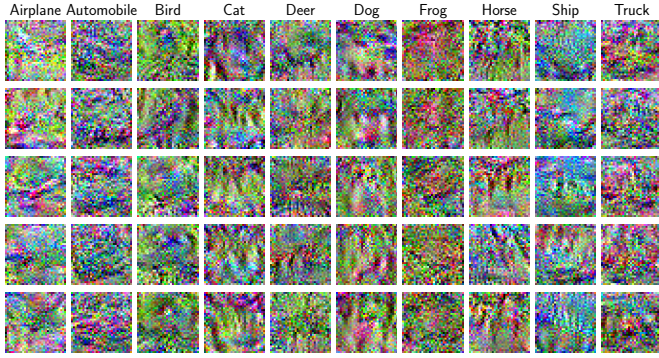
**(a)** Distilled clean images by DD algorithm
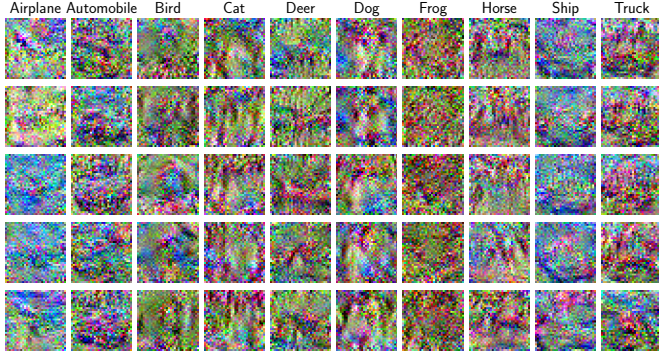


**(b)** Distilled images by DoorPing and DD algorithm

**Fig. 8:** Comparison of distilled images by DoorPing, where ⟨DD, AlexNet, CIFAR10⟩.



**(a)** Distilled clean images by DC algorithm



**(b)** Distilled images by DoorPing and DC algorithm

**Fig. 9:** Comparison of distilled images by DoorPing, where ⟨DC, AlexNet, CIFAR10⟩.

Ideally, a backdoored model should be as accurate as a clean model, given clean test data to ensure its stealthiness. In this study, we evaluate the model utility from both quantitative and qualitative perspectives.

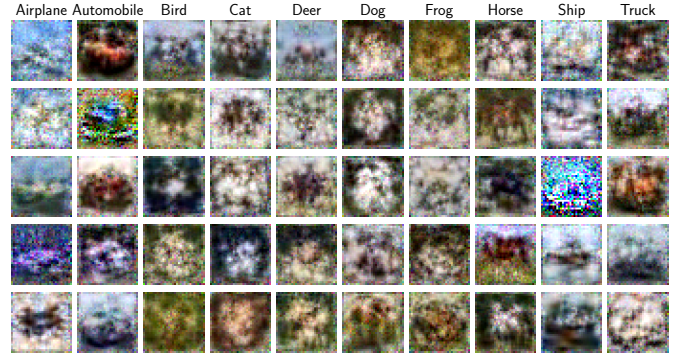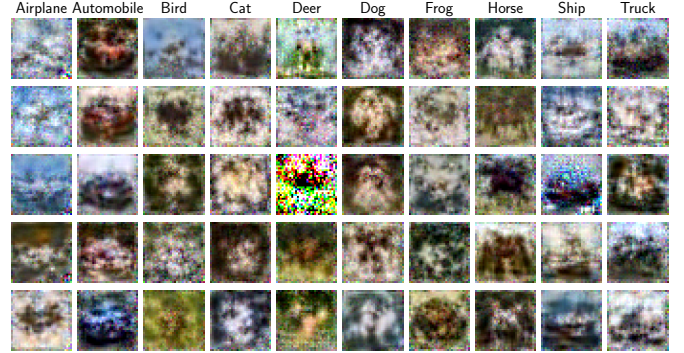We first conduct a quantitative evaluation of the *CTA* score between the clean and backdoored models. As shown in Figure 7, we find that the *CTA* scores of the backdoored models from NAIVEATTACK or DOORPING are similar to that of the clean models. For instance, the *CTA* score of clean model for ⟨DD, AlexNet, CIFAR10⟩ is 0.649. Meanwhile, the *CTA* scores for NAIVEATTACK and DOORPING are 0.646 and 0.635, respectively, which drop only by 0.464% and 2.1% compared to the clean model. Our results exemplify that the side effects caused by our backdoor attacks are within the acceptable performance variation of the model. They have no significant impact on utility performance. A similar observation can be drawn from other *CTA* scores. Besides, for ⟨DC, AlexNet, SVHN⟩, the clean model has the lowest *CTA* score, which is respectively 2.531% and 6.751% lower than NAIVEATTACK and DOORPING. As such, we carry out a Welsh $t$-test on the results (as we repeat the evaluation 10 times per our runtime configuration). Our null hypothesis is that the mean *CTA* score of DOORPING and the clean model is the same. The $t$-test results show that Welch-Satterthwaite Degrees of Freedom is 17.986 and $\rho$-value is 0.894; hence we cannot reject our null hypothesis. We conclude that such a difference is due to fluctuation.

We then conduct a qualitative evaluation by visualizing some examples of distilled images by normal dataset dis-

tillation and our attack in Figure 8 and Figure 9. The images distilled by DOORPING are much similar to the ones in Figure 8a More propitiously, the backdoor trigger is totally unrecognizable to human inspection, meaning that the trigger is completely hidden in the synthetic image.

**Takeaways.** Our experimental results demonstrate that all the backdoored models still have the same level of utility performance as the clean model, i.e., our proposed backdoor attacks preserve the model's utility.

## VI. Ablation Study

### A. Effectiveness on Complex Datasets

We also add the ablation study on the effectiveness of complex datasets. We test our attack using CIFAR100 [1], which has 100 classes containing 600 images each. For DD, due to the GPU memory limitation, we distill five images per class and 500 synthetic images in total. For DC, the hyperparameters and other settings remain the same as our main experiments. Table I illustrates the results of CIFAR100. All *ASR* scores are larger than 0.900 without significant *CTA* degradation compared with those of the clean model and NAIVEATTACK. Our results show that DOORPING can be easily extended to more complex datasets.

**Takeaway.** DOORPING can be extended to more complex datasets with more classes and data samples.

**TABLE I:** *ASR* and *CTA* of DOORPING with CIFAR100.

| | | DD | | DC | |
|---|---|---|---|---|---|
| | | *ASR* | *CTA* | *ASR* | *CTA* |
| Clean Model | AlexNet | 0.014±0.010 | 0.385±0.009 | 0.012±0.002 | 0.217±0.007 |
| | ConvNet | 0.029±0.011 | 0.215±0.014 | 0.012±0.002 | 0.197±0.007 |
| NAIVEATTACK | AlexNet | 0.128±0.013 | 0.375±0.010 | 0.007±0.003 | 0.209±0.005 |
| | ConvNet | 0.011±0.010 | 0.214±0.011 | 0.006±0.021 | 0.190±0.004 |
| DOORPING | AlexNet | 0.919±0.014 | 0.373±0.011 | 0.961±0.024 | 0.209±0.006 |
| | ConvNet | 1.000±0.000 | 0.205±0.012 | 1.000±0.000 | 0.196±0.002 |

### B. Effectiveness on Cross Architectures

Several dataset distillation methods [5], [90] explore cross-architecture (CA) data distillation (i.e., the data distillation model is different from the downstream model). To understand the effectiveness of DOORPING on such cross-architecture scenarios, we choose three model architectures - AlexNet [33], ConvNet, and VGG11 [67] in our study. We use AlexNet (ConvNet) as the distillation model and the other two architectures for evaluation. As we can see in Table II, given DC algorithm, DOORPING achieves good *ASR* and *CTA* scores on VGG11 as the downstream model, which is trained on the synthetic data distilled by ConvNet. In general, DOORPING performs well on all cross-architecture models using the synthetic data distilled by ConvNet architecture. However, DOORPING does not perform well in most cross-architecture models using the synthetic data distilled by DD algorithm. We speculate that the root cause is the difference in the distillation algorithms. For DD, it compresses the image information (gradient calculated by the specific model) into the distilled dataset, i.e., model-specific. In contrast, DC forces the synthetic dataset to learn the distribution of the original dataset, i.e., model-independent. Therefore, DC can better preserve the information of the original training images hence better preserving the trigger in the distilled dataset. Consequently, DC leaves the backdoor in a different model trained on this distilled dataset.

**Takeaway.** DOORPING can be used to attack cross-architecture models. However, its effectiveness may be affected by the distillation models.

### C. Number of Distilled Samples per Class

Previous distillation work [5], [52], [53], [88], [89], [90] has proven that better *CTA* can be achieved by increasing the number of distilled samples. It motivated us to investigate the effect of the number of distilled samples per class on our attacks. Concretely, we select 1, 10, and 50 samples per class to assess the effect on both NAIVEATTACK and DOORPING. We show the backdoor attack performance in Table III. In general, we can see that the *ASR* score increases with the number of distilled samples in each class. We can also find that the attack performances of NAIVEATTACK and DOORPING are suboptimal when the number of distilled samples is 1, especially on the DC algorithm. So, if the gradient distribution of distilled image has a significant standard deviation compared to that of the original training samples, this distilled image cannot fully represent these training samples. However, when the number of distilled images varies from 10 to 50, the *ASR* scores become more stable, with only one below 0.970 (approximately 0.861). As for the *CTA*, we can observe a similar trend. More distilled samples lead to higher model utility performance. Yet, the

model utility does not improve much in most cases when attacking the DD algorithm, except for ⟨DD, AlexNet, SVHN⟩.

**Takeaways.** The increasing number of distilled samples leads to better *ASR* and *CTA* scores. This is expected since the downstream model is trained on more distilled training samples (hence more backdoored samples).

### D. Number of Original Samples

Dataset distillation aims to reduce the redundancy in the training datasets. One more possible redundancy is the number of original training samples. Here, we study the impact of the number of original training samples on the performance of our attacks. Concretely, we vary the proportion of the entire dataset from 0.2 to 1 to report the *ASR* and *CTA* scores. We show the trends with the increase of the number of the original training samples in Figure 10. As we can see, the *ASR* score generally grows with the upswing of the sample numbers. In particular, the *ASR* score of some cases remains stationary throughout. However, almost all cases start with a much lower *ASR* score, i.e., when only 20% of the samples from the original training dataset are used to distill images, both of our attacks only achieve relatively poor attack performance. For instance, the *ASR* score of ⟨DD, AlexNet, STL10⟩ is only 0.235. These results demonstrate that the number of original training dataset affect the attack performance significantly. As expected for the *CTA* score, the majority of the model utility increase with the increasing number of original training samples. Only some *CTA* scores oscillate in an ultra-fine range. For example, the *CTA* score of ⟨DC, ConvNet, SVHN⟩ fluctuates from 0.190 to 0.217, and the *ASR* score is 1.000 when using DOORPING. Besides, we can also find that for ⟨DC, AlexNet, STL10⟩, the *CTA* score only increases from 0.305 to 0.319, but the *ASR* score surges to over 0.811 after the proportion gets larger than 0.4. This means it is acceptable to distill less training data, as the model utility does not change much but still achieves an acceptable attack performance.

**Takeaways.** The increasing size of the original training dataset leads to better *ASR* and *CTA* scores. This is also expected since the distillation models have learned sufficient patterns from additional samples.

### E. Number of Selected Neurons

We aim to understand the impact of the number of selected neurons to optimize the trigger in DOORPING (i.e., the impact of top-$k$). Especially in the penultimate layer of the models, we conduct the evaluation by setting the number of selected neurons to 1, 2, 5, and 10, respectively. We report the *ASR* score, the *CTA* score, and the average running time per epoch in Figure 11, Figure 12, Figure 13, and Figure 14. As we can see from these figures, with the number of selected neurons increasing, the *ASR* scores decrease while the runtime increases. The experiments also show that the *CTA* scores are almost unchanged. For example, the *ASR* scores of ⟨DD, AlexNet, CIFAR10⟩ are 0.999, 1.000, 0.665, and 0.440. Their respective runtime increases from 15s to 602s, while the *CTA* scores remain stable (0.635, 0.636, 0.648, 0.635. respectively). The reason behind this is that the weights of some selected neurons connecting these neurons to the preceding and following layers are smaller than others. In other words, we need to refrain from

**TABLE II:** *ASR* and *CTA* of cross model architectures.

| | | FMNIST | | | | CIFAR10 | | | | STL10 | | | | SVHN | | | |
| | | AlexNet | | ConvNet | | AlexNet | | ConvNet | | AlexNet | | ConvNet | | AlexNet | | ConvNet | |
| | CA Model | ASR | CTA | ASR | CTA | ASR | CTA | ASR | CTA | ASR | CTA | ASR | CTA | ASR | CTA | ASR | CTA |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DD | AlexNet | 1.000±0.000 | 0.868±0.013 | 0.500±0.500 | 0.300±0.118 | 0.999±0.000 | 0.635±0.009 | 0.000±0.000 | 0.102±0.005 | 0.999±0.000 | 0.438±0.010 | 1.000±0.000 | 0.120±0.011 | 0.970±0.009 | 0.759±0.010 | 0.000±0.000 | 0.091±0.008 |
| | ConvNet | 0.411±0.069 | 0.342±0.031 | 1.000±0.000 | 0.804±0.014 | 0.471±0.014 | 0.214±0.013 | 1.000±0.000 | 0.476±0.011 | 0.994±0.002 | 0.216±0.009 | 1.000±0.000 | 0.371±0.011 | 0.000±0.000 | 0.195±0.013 | 1.000±0.000 | 0.375±0.015 |
| | VGG11 | 0.000±0.000 | 0.115±0.012 | 0.000±0.000 | 0.151±0.077 | 0.385±0.472 | 0.097±0.010 | 0.200±0.400 | 0.104±0.013 | 0.100±0.300 | 0.208±0.018 | 0.100±0.300 | 0.127±0.027 | 0.000±0.000 | 0.133±0.040 | 0.000±0.000 | 0.085±0.038 |
| DC | AlexNet | 1.000±0.000 | 0.751±0.012 | 1.000±0.000 | 0.716±0.013 | 1.000±0.000 | 0.364±0.029 | 1.000±0.000 | 0.301±0.019 | 0.811±0.146 | 0.317±0.034 | 1.000±0.000 | 0.312±0.012 | 0.693±0.177 | 0.419±0.097 | 1.000±0.000 | 0.446±0.039 |
| | ConvNet | 0.235±0.200 | 0.681±0.027 | 1.000±0.000 | 0.734±0.008 | 0.108±0.016 | 0.287±0.011 | 1.000±0.000 | 0.308±0.008 | 0.017±0.010 | 0.308±0.021 | 0.981±0.000 | 0.331±0.013 | 0.321±0.131 | 0.180±0.009 | 1.000±0.000 | 0.203±0.021 |
| | VGG11 | 0.349±0.434 | 0.744±0.009 | 1.000±0.000 | 0.759±0.005 | 1.000±0.000 | 0.312±0.009 | 1.000±0.000 | 0.287±0.005 | 1.000±0.000 | 0.306±0.006 | 1.000±0.000 | 0.281±0.008 | 1.000±0.000 | 0.477±0.011 | 1.000±0.000 | 0.380±0.014 |

**TABLE III:** Attack performance under different target models and distilled samples per class.

| | | | FMNIST | | | | CIFAR10 | | | | STL10 | | | | SVHN | | | |
| | | | NAIVEATTACK | | DOORPING | | NAIVEATTACK | | DOORPING | | NAIVEATTACK | | DOORPING | | NAIVEATTACK | | DOORPING | |
| | | # | ASR | CTA | ASR | CTA | ASR | CTA | ASR | CTA | ASR | CTA | ASR | CTA | ASR | CTA | ASR | CTA |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AlexNet | DD | 1 | 0.102±0.025 | 0.828±0.010 | 1.000±0.000 | 0.821±0.009 | 0.692±0.016 | 0.646±0.014 | 1.000±0.000 | 0.633±0.015 | 0.113±0.011 | 0.407±0.010 | 1.000±0.000 | 0.424±0.011 | 0.577±0.015 | 0.374±0.012 | 0.997±0.002 | 0.406±0.014 |
| | | 10 | 0.103±0.006 | 0.867±0.012 | 1.000±0.000 | 0.868±0.009 | 0.692±0.009 | 0.646±0.011 | 0.999±0.000 | 0.635±0.013 | 0.123±0.009 | 0.428±0.011 | 0.999±0.000 | 0.438±0.010 | 0.797±0.009 | 0.729±0.019 | 0.970±0.009 | 0.759±0.010 |
| | | 50 | 0.186±0.010 | 0.871±0.011 | 1.000±0.000 | 0.882±0.011 | 0.809±0.014 | 0.652±0.013 | 0.972±0.028 | 0.650±0.015 | 0.110±0.007 | 0.454±0.010 | 1.000±0.000 | 0.436±0.012 | 0.782±0.020 | 0.765±0.015 | 1.000±0.000 | 0.755±0.017 |
| | DC | 1 | 0.085±0.008 | 0.536±0.032 | 0.098±0.006 | 0.541±0.023 | 0.214±0.059 | 0.229±0.016 | 0.268±0.054 | 0.234±0.014 | 0.126±0.294 | 0.196±0.044 | 0.179±0.120 | 0.180±0.033 | 0.332±0.118 | 0.111±0.013 | 0.284±0.131 | 0.111±0.012 |
| | | 10 | 0.118±0.006 | 0.757±0.009 | 1.000±0.000 | 0.751±0.029 | 0.133±0.032 | 0.358±0.047 | 1.000±0.000 | 0.364±0.012 | 0.098±0.006 | 0.305±0.040 | 0.811±0.146 | 0.317±0.034 | 0.333±0.270 | 0.412±0.083 | 0.693±0.177 | 0.419±0.097 |
| | | 50 | 0.126±0.013 | 0.828±0.004 | 1.000±0.000 | 0.813±0.003 | 0.151±0.021 | 0.467±0.006 | 0.990±0.009 | 0.470±0.006 | 0.153±0.016 | 0.462±0.005 | 0.861±0.062 | 0.471±0.007 | 0.763±0.030 | 0.741±0.010 | 0.979±0.010 | 0.735±0.007 |
| ConvNet | DD | 1 | 0.124±0.007 | 0.784±0.014 | 1.000±0.000 | 0.800±0.010 | 0.137±0.014 | 0.450±0.012 | 1.000±0.000 | 0.453±0.014 | 0.151±0.008 | 0.357±0.013 | 1.000±0.000 | 0.367±0.010 | 0.612±0.016 | 0.332±0.021 | 1.000±0.000 | 0.340±0.013 |
| | | 10 | 0.126±0.009 | 0.803±0.010 | 1.000±0.000 | 0.804±0.011 | 0.105±0.026 | 0.478±0.011 | 1.000±0.000 | 0.476±0.014 | 0.136±0.012 | 0.363±0.012 | 1.000±0.000 | 0.371±0.011 | 0.712±0.002 | 0.372±0.016 | 1.000±0.000 | 0.375±0.015 |
| | | 50 | 0.242±0.010 | 0.828±0.013 | 1.000±0.000 | 0.828±0.009 | 0.121±0.008 | 0.482±0.013 | 1.000±0.000 | 0.489±0.014 | 0.134±0.005 | 0.378±0.013 | 1.000±0.000 | 0.370±0.010 | 0.912±0.025 | 0.477±0.017 | 1.000±0.000 | 0.482±0.016 |
| | DC | 1 | 0.081±0.007 | 0.535±0.023 | 0.091±0.004 | 0.545±0.013 | 0.222±0.043 | 0.230±0.006 | 0.225±0.051 | 0.229±0.007 | 0.166±0.024 | 0.223±0.007 | 0.181±0.037 | 0.217±0.009 | 0.090±0.056 | 0.113±0.007 | 0.114±0.029 | 0.111±0.006 |
| | | 10 | 0.102±0.006 | 0.732±0.007 | 1.000±0.000 | 0.734±0.008 | 0.113±0.012 | 0.310±0.017 | 1.000±0.000 | 0.308±0.008 | 0.093±0.004 | 0.321±0.012 | 0.981±0.000 | 0.331±0.013 | 0.430±0.187 | 0.215±0.015 | 1.000±0.000 | 0.203±0.021 |
| | | 50 | 0.107±0.005 | 0.776±0.005 | 1.000±0.000 | 0.774±0.004 | 0.117±0.008 | 0.380±0.007 | 1.000±0.000 | 0.361±0.007 | 0.106±0.007 | 0.413±0.011 | 0.998±0.002 | 0.421±0.007 | 0.851±0.059 | 0.480±0.027 | 1.000±0.000 | 0.490±0.021 |
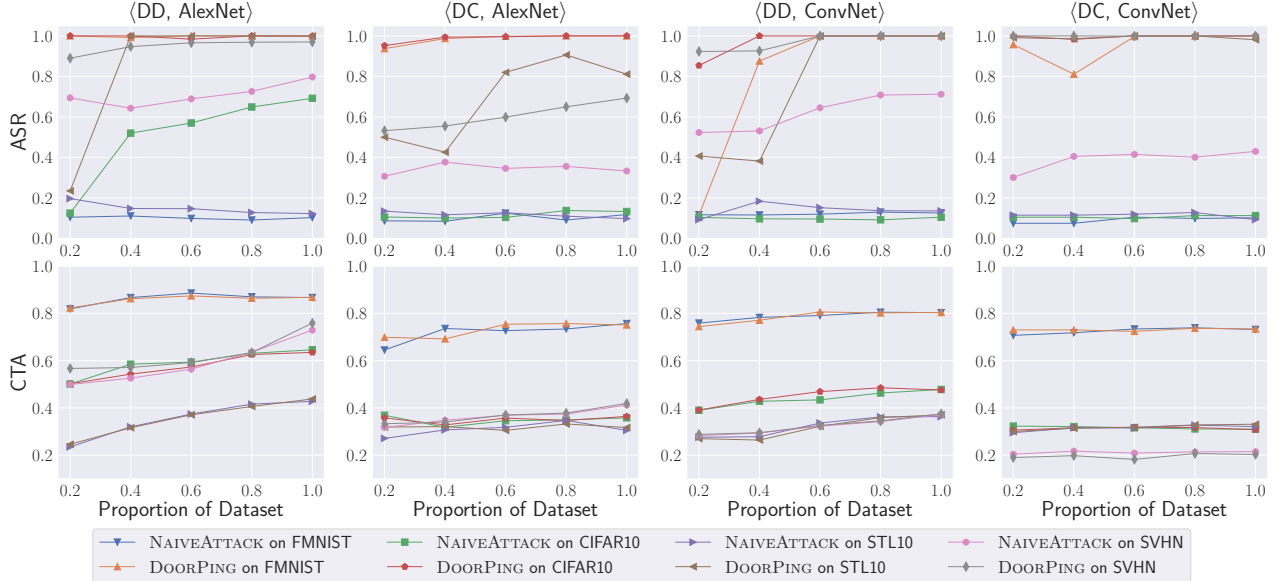


**Fig. 10:** *ASR* and *CTA* of NAIVEATTACK and DOORPING under the different proportions of original training datasets and different model architectures. The X-axis represents the proportion of the original samples used in the dataset distillation process.

exploiting these neurons to enhance the triggers. We, therefore, set the number of the selected neurons to 1 throughout all experiments in the paper.

**Takeaways.** The increasing number of selected neurons harms the attack performance while increasing the runtime. This observation is in line with previous work [45].

### F. Poisoning Ratio

Here, we investigate the impact of the poisoning ratio (i.e., $\epsilon$ in Section III-C) in the entire training dataset. We vary the poisoning ratio from 0.01 to 0.5 and report both attack and utility performance in Figure 15. For the attack performance, we can find that the *ASR* scores vary significantly in general. For instance, the *ASR* scores increase from 0.811

to 1.000 and from 0.693 to 1.000 for ⟨DC, AlexNet, STL10⟩, and ⟨DC, AlexNet, SVHN⟩ in DOORPING, respectively. For the majority of cases in NAIVEATTACK, with the poisoning ratio increasing from 0.05 to 0.5, the *ASR* scores are also increasing. Taking ⟨DD, AlexNet, STL10⟩ as an example, for the poisoning ratio of NAIVEATTACK increases from 0.01 to 0.05, the *ASR* score fluctuates between 0.136 and 0.175. When the poisoning ratio increases to 0.1 and expends to 0.5, the *ASR* score rises and eventually reaches 0.990. However, unlike DOORPING, we find it challenging to achieve a 1.000 *ASR* score in NAIVEATTACK, which exemplifies that DOORPING is more effective than NAIVEATTACK. For the model utility performance, we can observe a general downward trend. For example, the *CTA* score of DOORPING decreases from 0.364
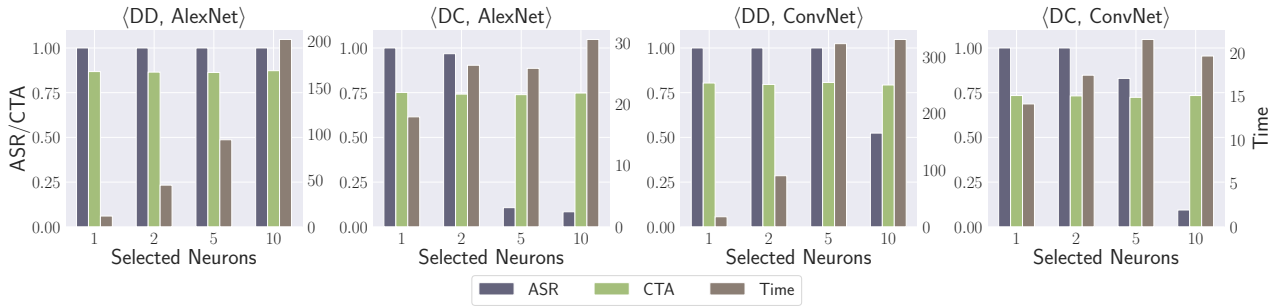
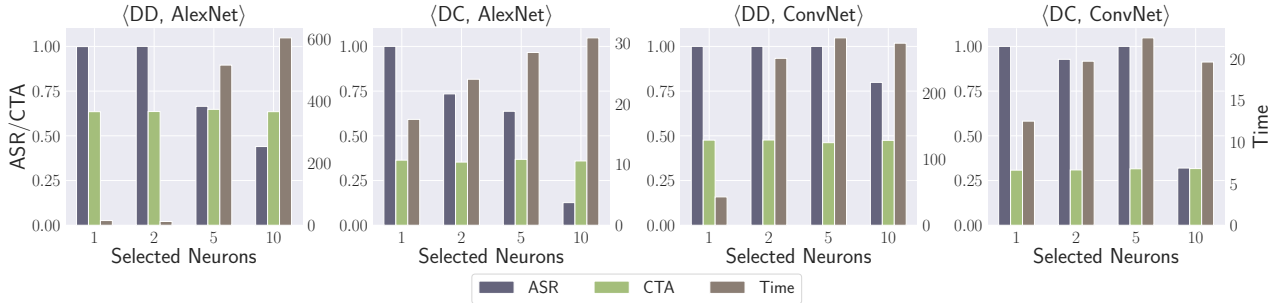**Fig. 11:** *ASR*, *CTA*, and the running time of DOORPING with different selected neurons for FMNIST dataset.



**Fig. 12:** *ASR*, *CTA*, and runtime of DOORPING with different selected neurons for CIFAR10 dataset.
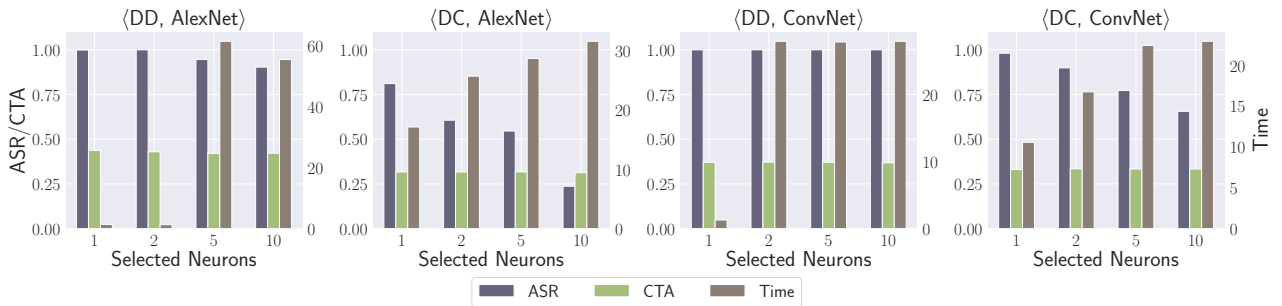


**Fig. 13:** *ASR*, *CTA*, and the running time of DOORPING with different selected neurons for STL10 dataset.
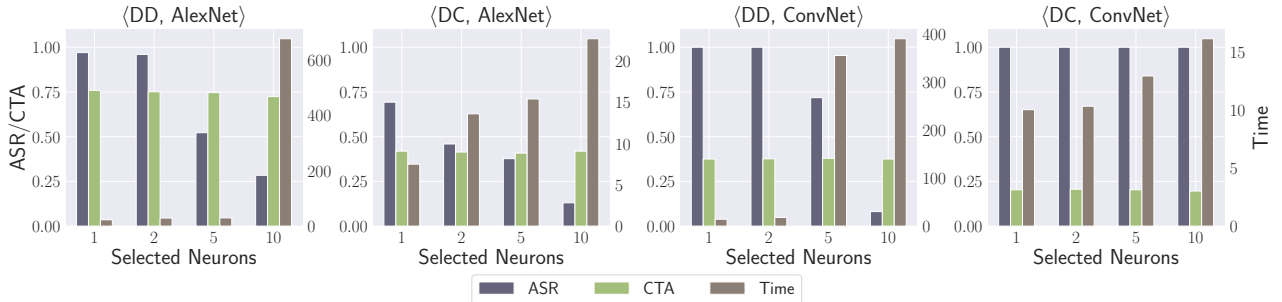


**Fig. 14:** *ASR*, *CTA*, and the running time of DOORPING with different selected neurons for SVHN dataset.

to 0.325 on $\langle$DC, AlexNet, CIFAR10$\rangle$. We also observe similar trends in NAIVEATTACK.

**Takeaways.** The poisoning ratio impacts the *ASR* scores, especially for NAIVEATTACK. However, the *CTA* scores may vary given different backdoor sample ratios incurred by the respective properties of distillation models (e.g., hyperparameters, architectures, etc.).

### G. Trigger Size

Previous work [45] has shown that the larger the trigger size is, the higher the attack performance is. Thus, we first investigate the impact of trigger size on attack performance. We set the trigger size to $2 \times 2$, $3 \times 3$, and $4 \times 4$ to investigate their impacts on the attack and utility performance. Table IV shows the *ASR* and the *CTA* scores with respect to different trigger sizes. For NAIVEATTACK, as the trigger size increases, the *ASR* score also increases, especially from $3 \times 3$ to $4 \times 4$. For instance, the *ASR* score of NAIVEATTACK for $\langle$DC, ConvNet,
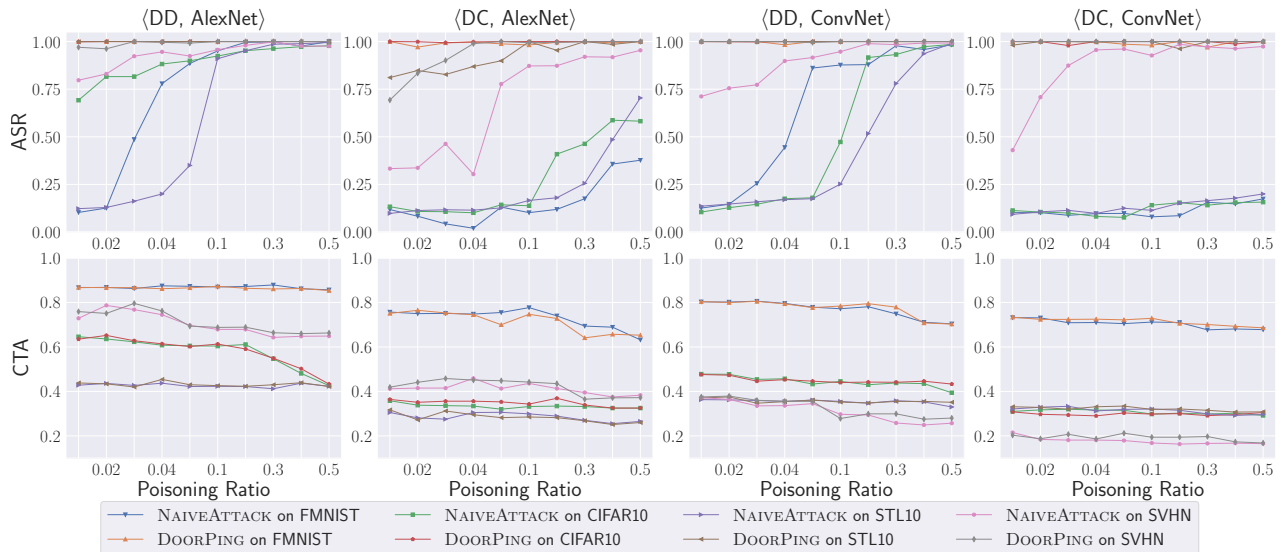
**Fig. 15:** *ASR* and *CTA* of NAIVEATTACK and DOORPING under the different poisoning ratios and different model architectures. The X-axis represents the poisoning ratio in the whole training dataset.

SVHN⟩ increases from 0.430 to 0.769. For DOORPING, we can see that the *ASR* score is close to 1.0 in most cases, regardless of the trigger size setting. For example, given ⟨DD, AlexNet, STL10⟩, when the trigger size is increased from $2 \times 2$ to $4 \times 4$, the *ASR* score increases from 0.811 to 0.984. Similarly, the *ASR* score increases from 0.693 to 0.805 for SVHN. These results show that larger triggers generally lead to higher attack performance in our attacks. In terms of the impact of trigger size on utility performance, we find that the majority of *CTA* scores slightly decrease with the increase of the trigger size. To this end, we calculate the Pearson correlation coefficient between the trigger size and the *CTA* score. In total, we have 32 correlation values. Among those values, 9 are positive, and 23 are negative. The average of the correlations is -0.370. Therefore, the *CTA* negatively correlates with the trigger size. Despite the side effects caused by larger trigger sizes, the *CTA* scores are still within the acceptable performance variation of the model. They do not significantly impact the model utility performance.

**Takeaways.** When the trigger size becomes more prominent and larger, the final synthetic image contains more trigger information but less information of the original images. It may lead to the inevitable trade-off between attack performance and model utility.

### H. Trigger Trajectory

During the distillation process, as we update the backdoor trigger based on the model parameters during the distillation process, these triggers should be different theoretically at different distilled epochs. We collect all the generated backdoor triggers from different distilled epochs. We use the distilled images to train the downstream model after the distillation and test all the trigger images we collect. We find that the *ASR* scores from these triggers can also achieve similar results as the results after the distillation. For example, the triggers generated in the 10 distilled epochs lead to a *ASR* score of 1.000 on the backdoored model trained on 400 distilled-epoch images of ⟨DD, AlexNet, CIFAR10⟩. This reality is actually an advantage of our attack, especially facing a defense like De-trigger that needs to know the exact triggers. In our situation, we have numerous triggers from different distilled epochs, which makes the defense much harder. This also means our distilled images contain information about the triggers in the distillation process, even though these triggers are different in each distilled epoch. This trajectory during the training procedure can result in a more challenging trigger detection. We name this phenomenon *trigger trajectory*.

**Takeaways.** The trigger trajectory is a unique feature offered by DOORPING. It enables the attackers to record a set of triggers that can later be used to attack the downstream models.

### I. Value of Magnifying Factor $\alpha$

We calculate the MSE Loss between the output and the output magnified by $\alpha$. Thus, we should know how $\alpha$ acts on the optimized trigger. Theoretically, the larger $\alpha$ is, the better the triggers are during the distillation process. Previous works [36], [45] set the magnifying value to a specific constant, 100, whereas the optimizer will allow the outputs from the selected neurons in the penultimate larger closer to this number. However, the weakness of setting constants is that the optimizer will not work when the output itself is close to 100. To solve this problem, our method chooses to magnify the output to $\alpha$ times so that the triggers will always substantially affect these selected neurons. In particular, we choose $\alpha$ from 10, 50, and 100. Figure 16 and Figure 17 reports the results among different $\alpha$. We can see from Figure 16, the *ASR* scores are almost close to 1.000 with the alpha increasing, and the utilities are stable in Figure 17. In our experiments, we simply set it to 10.

**Takeways.** The *ASR* increases with the increasing magnifying factor $\alpha$. However, the attack performance plateaus after $\alpha$ is greater than 50.

**Remark.** We also investigate the impact of the number of distillation epochs on attack and utility performance. and test

**TABLE IV:** Attack performance of different target models and trigger sizes.

| | | | FMNIST | | | | CIFAR10 | | | | STL10 | | | | SVHN | | | |
| | | | NAIVEATTACK | | DOORPING | | NAIVEATTACK | | DOORPING | | NAIVEATTACK | | DOORPING | | NAIVEATTACK | | DOORPING | |
| | | # | ASR | CTA | ASR | CTA | ASR | CTA | ASR | CTA | ASR | CTA | ASR | CTA | ASR | CTA | ASR | CTA |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AlexNet | DD | 2 | 0.103±0.006 | 0.867±0.012 | 1.000±0.000 | 0.868±0.009 | 0.692±0.009 | 0.646±0.011 | 0.999±0.000 | 0.635±0.013 | 0.123±0.009 | 0.428±0.011 | 0.999±0.000 | 0.438±0.010 | 0.797±0.009 | 0.729±0.019 | 0.970±0.009 | 0.759±0.010 |
| | | 3 | 0.729±0.007 | 0.833±0.013 | 1.000±0.000 | 0.814±0.013 | 0.789±0.013 | 0.646±0.014 | 1.000±0.000 | 0.614±0.012 | 0.126±0.008 | 0.413±0.011 | 1.000±0.000 | 0.418±0.010 | 0.805±0.005 | 0.699±0.015 | 0.982±0.011 | 0.702±0.020 |
| | | 4 | 0.929±0.010 | 0.809±0.012 | 1.000±0.000 | 0.809±0.011 | 0.809±0.010 | 0.582±0.016 | 1.000±0.000 | 0.606±0.014 | 0.154±0.010 | 0.423±0.009 | 1.000±0.000 | 0.442±0.012 | 0.857±0.020 | 0.677±0.026 | 0.993±0.005 | 0.639±0.017 |
| | DC | 2 | 0.118±0.006 | 0.757±0.009 | 1.000±0.000 | 0.751±0.029 | 0.133±0.032 | 0.358±0.047 | 1.000±0.000 | 0.364±0.012 | 0.098±0.046 | 0.305±0.040 | 0.811±0.146 | 0.317±0.034 | 0.333±0.270 | 0.412±0.083 | 0.693±0.177 | 0.419±0.097 |
| | | 3 | 0.062±0.028 | 0.749±0.036 | 1.000±0.000 | 0.734±0.011 | 0.105±0.024 | 0.347±0.019 | 1.000±0.000 | 0.347±0.029 | 0.123±0.203 | 0.296±0.066 | 0.965±0.000 | 0.318±0.054 | 0.323±0.016 | 0.437±0.057 | 0.633±0.303 | 0.448±0.118 |
| | | 4 | 0.124±0.019 | 0.750±0.010 | 1.000±0.011 | 0.753±0.008 | 0.160±0.087 | 0.333±0.065 | 0.988±0.011 | 0.342±0.010 | 0.154±0.054 | 0.293±0.030 | 0.984±0.000 | 0.308±0.039 | 0.475±0.254 | 0.434±0.098 | 0.805±0.185 | 0.424±0.122 |
| ConvNet | DD | 2 | 0.126±0.009 | 0.803±0.010 | 1.000±0.000 | 0.804±0.011 | 0.105±0.026 | 0.478±0.011 | 1.000±0.000 | 0.476±0.011 | 0.136±0.012 | 0.363±0.012 | 1.000±0.000 | 0.371±0.011 | 0.712±0.002 | 0.372±0.016 | 1.000±0.000 | 0.375±0.015 |
| | | 3 | 0.216±0.003 | 0.791±0.006 | 1.000±0.000 | 0.784±0.012 | 0.125±0.014 | 0.467±0.013 | 0.999±0.000 | 0.477±0.014 | 0.133±0.006 | 0.353±0.013 | 1.000±0.000 | 0.346±0.012 | 0.770±0.006 | 0.365±0.019 | 0.999±0.000 | 0.356±0.017 |
| | | 4 | 0.560±0.009 | 0.800±0.013 | 1.000±0.000 | 0.798±0.012 | 0.167±0.013 | 0.465±0.013 | 1.000±0.000 | 0.456±0.013 | 0.139±0.009 | 0.366±0.011 | 1.000±0.000 | 0.374±0.010 | 0.771±0.036 | 0.368±0.017 | 0.975±0.021 | 0.378±0.012 |
| | DC | 2 | 0.102±0.006 | 0.732±0.007 | 1.000±0.000 | 0.734±0.008 | 0.113±0.012 | 0.310±0.017 | 1.000±0.000 | 0.308±0.008 | 0.093±0.004 | 0.321±0.012 | 0.981±0.000 | 0.331±0.013 | 0.430±0.187 | 0.215±0.015 | 1.000±0.000 | 0.203±0.021 |
| | | 3 | 0.079±0.009 | 0.737±0.010 | 1.000±0.000 | 0.737±0.006 | 0.107±0.023 | 0.315±0.015 | 0.998±0.001 | 0.317±0.008 | 0.128±0.011 | 0.320±0.011 | 0.991±0.000 | 0.333±0.008 | 0.588±0.043 | 0.228±0.016 | 1.000±0.000 | 0.210±0.040 |
| | | 4 | 0.110±0.011 | 0.732±0.006 | 0.998±0.001 | 0.707±0.006 | 0.120±0.016 | 0.321±0.012 | 1.000±0.000 | 0.305±0.011 | 0.136±0.015 | 0.334±0.015 | 1.000±0.000 | 0.327±0.012 | 0.769±0.153 | 0.168±0.022 | 1.000±0.000 | 0.211±0.033 |
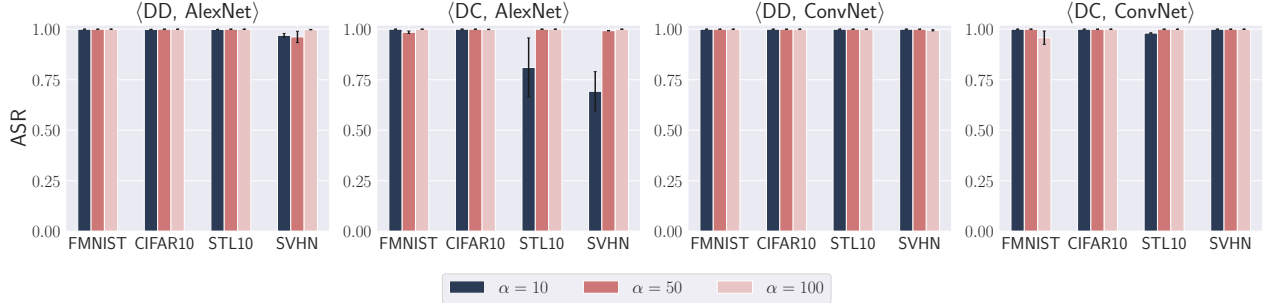


**Fig. 16:** *ASR* of DOORPING with different $\alpha$.
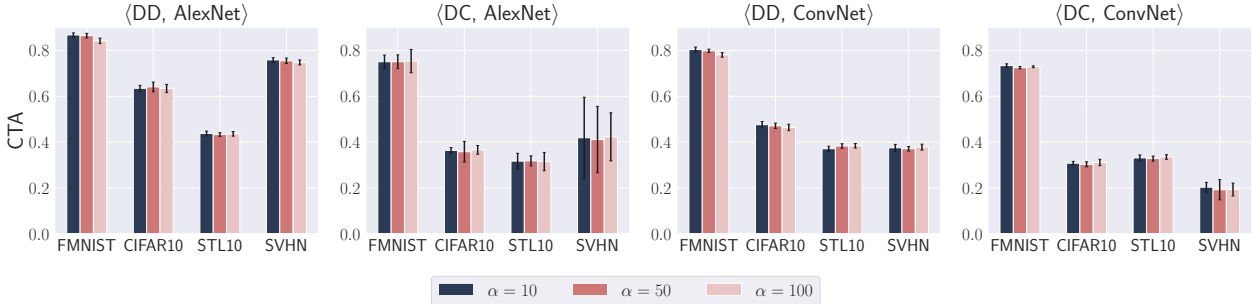


**Fig. 17:** *CTA* of DOORPING with different $\alpha$.

another trigger pattern technique, invisible trigger [36]. Due to space limitations, we defer them to our technical report [46].

## VII. DEFENSES

To mitigate the threat of backdoor attacks, many defense mechanisms have been proposed in the literature. These defenses can be broadly categorized into three detection levels [85], i.e., *model-level* (if a model is backdoored) [39], [44], [76], *input-level* (if the test time input contains triggers) [9], [19], [34], and *dataset-level* (if a training dataset is backdoored) [25], [72], [74]. In this section, we evaluate if our attacks can be defended by the existing mechanisms at all three levels. For each detection level, we select three representative approaches. Note that we only evaluate DOORPING here due to its good attack performance (see Section IV).

### A. Model-Level Defense

**ABS [44].** ABS analyzes the inner neuron behavior by determining how the output activation changes when introducing different levels of stimulation to a neuron. The neurons that substantially elevate the activation of a particular output label,

regardless of the input, are considered potentially compromised. We apply ABS to identify these neurons in the backdoored models. For all experiments, ABS does not identify any backdoor neurons or layers for all the models. All the compromised neuron candidate lists are empty. We conclude that ABS cannot defend DOORPING.

**Neural Attention Distillation (NAD) [39].** NAD is an architecture to erase backdoors from backdoored models. It utilizes a teacher model to fine-tune the backdoored student model using a small subset of clean data. In this way, the intermediate-layer attention of the student model aligns with that of the teacher model. The backdoor is then effectively removed. In our experiments, we choose a subset of the clean dataset with a proportion of 0.050 and a clean model trained by the clean dataset as our teacher model. We report the *ASR* and *CTA* scores after the fine-tuning process in Table V. We can clearly see that all the fine-tuned models classify the input into one specific class. This behavior leads to low *CTA* scores (~0.100) and *ASR* scores of 1.000 or 0.000. Our results show that NAD is not an effective defense against DOORPING either.

**Neural Cleanse [76].** Neural Cleanse generates *Anomaly Index*

**TABLE V:** *ASR* and *CTA* of DOORPING backdoored models after NAD process.

| | AlexNet | | | | ConvNet | | | |
| | DD | | DC | | DD | | DC | |
| | *ASR* | *CTA* | *ASR* | *CTA* | *ASR* | *CTA* | *ASR* | *CTA* |
|---|---|---|---|---|---|---|---|---|
| **FMNIST** | 0.000 | 0.100 | 1.000 | 0.100 | 1.000 | 0.100 | 1.000 | 0.100 |
| **CIFAR10** | 0.000 | 0.100 | 1.000 | 0.100 | 1.000 | 0.100 | 1.000 | 0.100 |
| **STL10** | 1.000 | 0.100 | 1.000 | 0.100 | 1.000 | 0.100 | 1.000 | 0.100 |
| **SVHN** | 1.000 | 0.067 | 1.000 | 0.067 | 1.000 | 0.067 | 1.000 | 0.067 |

**TABLE VI:** Anomaly Indices produced by Neural Cleanse for DOORPING. A classifier is predicted to be backdoored if the Anomaly Index is larger than 2.

| | AlexNet | | ConvNet | |
| | DD | DC | DD | DC |
|---|---|---|---|---|
| **FMNIST** | 1.466 | 1.670 | 1.304 | 0.995 |
| **CIFAR10** | 1.338 | 0.919 | 0.745 | 1.895 |
| **STL10** | 0.879 | 0.676 | 0.676 | 1.218 |
| **SVHN** | 1.835 | 1.908 | 1.266 | 0.739 |

**TABLE VII:** *ASR* and *CTA* of NAIVEATTACK backdoored models after De-noising Autoencoder process.

| | AlexNet | | | | ConvNet | | | |
| | DD | | DC | | DD | | DC | |
| | *ASR* | *CTA* | *ASR* | *CTA* | *ASR* | *CTA* | *ASR* | *CTA* |
|---|---|---|---|---|---|---|---|---|
| **FMNIST** | 0.050 | 0.414 | 0.000 | 0.669 | 0.207 | 0.504 | 0.059 | 0.660 |
| **CIFAR10** | 0.098 | 0.279 | 0.000 | 0.337 | 0.320 | 0.261 | 0.008 | 0.284 |
| **STL10** | 0.011 | 0.305 | 0.000 | 0.289 | 0.263 | 0.275 | 0.004 | 0.119 |
| **SVHN** | 0.000 | 0.347 | 0.000 | 0.484 | 0.000 | 0.143 | 0.000 | 0.133 |

**TABLE VIII:** *ASR* and *CTA* of DOORPING backdoored models after De-noising Autoencoder process.

| | AlexNet | | | | ConvNet | | | |
| | DD | | DC | | DD | | DC | |
| | *ASR* | *CTA* | *ASR* | *CTA* | *ASR* | *CTA* | *ASR* | *CTA* |
|---|---|---|---|---|---|---|---|---|
| **FMNIST** | 0.000 | 0.615 | 0.000 | 0.679 | 0.000 | 0.486 | 0.001 | 0.653 |
| **CIFAR10** | 0.000 | 0.264 | 0.013 | 0.344 | 0.081 | 0.285 | 0.000 | 0.287 |
| **STL10** | 0.000 | 0.334 | 0.000 | 0.297 | 1.000 | 0.283 | 0.000 | 0.262 |
| **SVHN** | 0.000 | 0.413 | 0.028 | 0.261 | 0.385 | 0.073 | 0.857 | 0.153 |

**TABLE IX:** *ASR* and *CTA* of DOORPING backdoored models after De-trigger Autoencoder process.

| | AlexNet | | | | ConvNet | | | |
| | DD | | DC | | DD | | DC | |
| | *ASR* | *CTA* | *ASR* | *CTA* | *ASR* | *CTA* | *ASR* | *CTA* |
|---|---|---|---|---|---|---|---|---|
| **FMNIST** | 0.039 | 0.508 | 0.049 | 0.574 | 0.003 | 0.287 | 0.002 | 0.390 |
| **CIFAR10** | 0.145 | 0.191 | 0.282 | 0.202 | 0.066 | 0.154 | 0.088 | 0.165 |
| **STL10** | 0.169 | 0.144 | 0.075 | 0.203 | 0.000 | 0.100 | 0.264 | 0.161 |
| **SVHN** | 0.122 | 0.143 | 0.260 | 0.197 | 0.065 | 0.195 | 0.126 | 0.133 |

of neuron units for a given classifier. If the Anomaly Index is greater than 2, the classifier is considered a backdoored model. We adopt the default parameter settings of Neural Cleanse and use the original testing dataset as a clean dataset in the evaluation. Table VI reports the Anomaly Index produced by Neural Cleanse for DOORPING. We can see that all the Anomaly Indices are consistently smaller than 2. It indicates that Neural Cleanse cannot detect our backdoor attacks in the distilled classifiers.

*B. Input-Level Defense*

**De-noising Autoencoder [9].** De-noising Autoencoder builds a deep autoencoder model by learning from paired clean images and their counterparts with added Gaussian noise. Upon the model being trained, De-noising Autoencoder removes the noise (i.e., the trigger) of the model input by feeding them to the autoencoder model, as we believe some triggers are recognized as noise in NAIVEATTACK and DOORPING. We follow the same procedure outlined in [9] to train the autoencoder in our experiments and then use the autoencoder to remove the noise of our backdoored distilled images. We evaluate the *ASR* and *CTA* of the backdoored model at the test time (i.e., the test time samples are first filtered by De-noising Autoencoder). As we can see in Table VII and Table VIII, most of *ASR* scores decrease. However, the *CTA* score also drops significantly in most cases. For example, the *CTA* score is only 0.191, which is lower than the original model utility of 0.654 of ⟨DD, AlexNet, CIFAR10⟩. The reason is that De-noising Autoencoder may also remove helpful information from the input images besides the trigger. There is a clear utility-defense trade-off when applying De-noising Autoencoder.

**De-trigger Autoencoder [34].** Similar to De-noising Autoencoder, De-trigger Autoencoder learns from both clean images a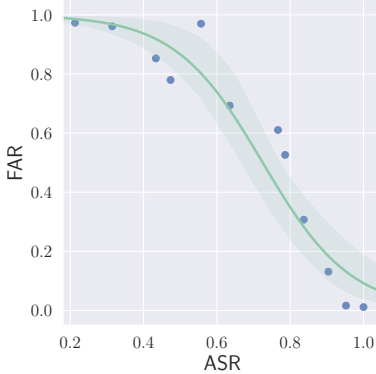nd clean images with the trigger to reconstruct the clean images. The defenders must know the trigger information (i.e., pattern and location) to train a De-trigger Autoencoder. All the testing procedures are the same as we outline in De-noising Autoencoder. We report the results in Table IX. All *ASR* scores and *CTA* scores decrease sharply. The majority are even worse than the result of De-noising Autoencoder. In conclusion, De-trigger Autoencoder cannot defend the DOORPING as it suffers from the same utility-defense trade-off.

**STRIP [19].** STRIP filters triggered samples at the test time based on the predicted randomness of perturbated samples (i.e., by applying different image patterns to suspicious images). Its detection capability is assessed by two metrics: false rejection rate (FRR) and false acceptance rate (FAR). The FRR is the probability when the benign input is regarded as a backdoored input by the STRIP detection system. The FAR is the probability that the backdoored input is recognized as the benign input by the STRIP detection system. A detection system usually attempts to minimize the FAR while using a slightly higher FRR as the trade-off. STRIP algorithm chooses the detection threshold by using the percent point function (PPF) on the distribution of the entropy of benign samples.

We use STRIP to check if the defender can use it to identify triggered samples in the test data. Table X reports the FRR and FAR scores of STRIP detecting the testing dataset (clean and backdoor). Here, we add the trigger to 2,000 images in the testing dataset and employ another 2,000 as benign ones. 10 images are employed as the overlay samples, which are used for replicating with the inputs to measure the randomness (entropy) of predicted labels. As we can observe in Table X, STRIP can achieve good detection performance for the testing images with triggers, i.e., both FRR and FAR is close to 0. In light of this finding, we further investigate why STRIP performs so well and how to reduce its detection performance from the perspective of an attacker. Recall that the critical insight of STRIP is that the predictions of all perturbed inputs of triggered images tend to be always consistent (i.e., the target class). In other words, the high detection performance, as shown in Table X, indicates that our optimized triggers can be stably preserved in the perturbed images. Crucially, our DOORPING attack enables the attacker to keep a trigger trajectory (see Section VI-H) whereby different triggers are preserved. Instead of using the final optimized trigger, we test if other triggers along the trajectory can be employed to find a

**TABLE X:** FRR and FAR of STRIP detecting test samples (clean and backdoor). We add the trigger into 40% of the original testing dataset and use another 40% as the benign samples. 10 images are treated as the overlay indices to evaluate FRR and FAR.

| | AlexNet | | | | ConvNet | | | |
| | DD | | DC | | DD | | DC | |
| | FRR | FAR | FRR | FAR | FRR | FAR | FRR | FAR |
|---|---|---|---|---|---|---|---|---|
| **FMNIST** | 0.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 0.000 | 1.000 |
| **CIFAR10** | 0.000 | 0.015 | 0.013 | 0.004 | 0.000 | 0.005 | 0.012 | 0.000 |
| **STL10** | 0.015 | 0.000 | 0.023 | 0.000 | 0.000 | 0.000 | 0.006 | 0.000 |
| **SVHN** | 0.015 | 0.000 | 0.016 | 0.024 | 0.020 | 0.110 | 0.016 | 0.000 |



**Fig. 18:** Relationship between the *ASR* and FAR.

**TABLE XI:** $J_T^*$ of the target class from different model architectures and datasets by SCAn.

| | AlexNet | | ConvNet | |
| | DD | DC | DD | DC |
|---|---|---|---|---|
| **FMNIST** | 1.868 | 2.224 | 6.817 | 7.075 |
| **CIFAR10** | 0.573 | 1.003 | 0.074 | 2.879 |
| **STL10** | 0.283 | 0.270 | 0.736 | 3.424 |
| **SVHN** | 0.671 | 0.015 | 0.954 | 0.226 |

**TABLE XII:** Average outlier score of samples generated by Spectral Signature on DOORPING. The smaller the score is, the more likely the sample is clean.

| | AlexNet | | | | ConvNet | | | |
| | DD | | DC | | DD | | DC | |
| | Backdoor | Clean | Backdoor | Clean | Backdoor | Clean | Backdoor | Clean |
|---|---|---|---|---|---|---|---|---|
| **FMNIST** | 7.623 | 9.476 | 4.584 | 2.203 | 6.829 | 9.077 | 5.411 | 2.883 |
| **CIFAR10** | 7.022 | 10.247 | 1.088 | 2.447 | 4.753 | 5.713 | 11.046 | 11.680 |
| **STL10** | 5.153 | 5.620 | 9.951 | 8.720 | 4.466 | 4.055 | 2.572 | 4.092 |
| **SVHN** | 6.416 | 10.972 | 4.198 | 15.856 | 5.131 | 9.406 | 23.041 | 10.793 |

**TABLE XIII:** Accuracy of detecting backdoor samples by using SPECTRE.

| | AlexNet | | ConvNet | |
| | DD | DC | DD | DC |
|---|---|---|---|---|
| **FMNIST** | 60% | 60% | 90% | 50% |
| **CIFAR10** | 80% | 70% | 50% | 40% |
| **STL10** | 20% | 50% | 40% | 50% |
| **SVHN** | 100% | 50% | 100% | 70% |

balance point between attack and detection performance. We show the relationship between the *ASR* and FAR in Figure 18. As we can see, the FAR score of STRIP can be significantly increased (i.e., poor detection performance) when we apply triggers that lead to a suboptimal *ASR*. For example, when the FAR score is around 0.595, the *ASR* score is about 0.767, attaining a decent attack performance. Our finding indicates that the DOORPING attack can practically evade STRIP detection by trading off some attack performance.

### C. Dataset-Level Defense

**Statistical Analysis of DNNs (SCAn) [72].** SCAn leverages an Expectation-Maximization (EM) algorithm to decompose an image into its identity part (e.g., person) and variation part (e.g., poses). Based on the global information of all categories, the distribution of variations is exploited by a likelihood ratio test to analyze the representations in each category, identifying those that are more likely to be described by a mixture model by adding attack samples into legitimate images of the current category. When the test statistic of the class $T$ (denoted as $J_T^*$) is larger than 7.389, the class $T$ is reported as being contaminated. Here, 7.389 is actually $e^2$ determined by SCAn. Table XI reports the test statistic for our backdoor target class (class 0). As we can see in Table XI, none of the $J_T^*$ scores are larger than 7.389. The results show that SCAn cannot detect our backdoor target class effectively by DOORPING.

**Spectral Signature [74].** Spectral signature builds on top of the idea that a classifier amplifies signals that are critical to classification. It finds that backdoored training datasets used in backdoor attacks can leave detectable traces in the covariance spectrum of the feature representation, i.e., the clean sample leads to a small covariance value. In contrast, the backdoor sample leads to an immense covariance value. Spectral signa-

ture calculates the outlier score of each sample and the mean value for the backdoor and clean samples. The results are shown in Table XII. As we can see in Table XII, most of the average outlier scores of backdoor samples are smaller than the clean ones. As such, Spectral Signature cannot detect the backdoored distilled datasets generated by DOORPING attack.

**SPECTRE [25].** SPECTRE is a defense algorithm using robust covariance estimation to amplify the spectral signature of backdoored data in the training dataset. The mean QUantum Entropy (QUE) score of a backdoor sample is usually higher than the clean sample. SPECTRE then marks such backdoor samples with a robust spectral signature. In the original settings, SPECTRE detects the backdoor sample in each class. For the DOORPING attack, all backdoor images are included in the target class we pre-defined. It means that there are no backdoor images in the other classes. To this end, we modify the settings of SPECTRE and only detect the backdoor samples in the target class. We include an equal number of backdoored and clean distilled images in the target class. Table XIII reports the accuracy of the SPECTRE detection. We can observe that SPECTRE performs well in some cases. For example, given ⟨DD, SVHN⟩, SPECTRE can detect the triggers by DOORPING. However, SPECTRE can not successfully identify the triggers in the other datasets. See more discussion about SPECTRE in our technical report [46]. We conclude that SPECTRE is not robust for all of the datasets we test. This inconsistency indicates that SPECTRE is not a reliable defense mechanism against our DOORPING attack.

## VIII. RELATED WORK

**Backdoor Attack.** Backdoor attack [8], [20], [24], [31], [45] is a training time attack and has emerged as a major security threat to deep neural networks (DNNs) in many application areas (e.g., natural language processing [7], [62], image classification [14], [15], face recognition [8], point clouds [37],

[81], etc.). It implants a hidden backdoor (also called neural trojan [31], [45]) into the target model via poisoning training samples (i.e., attacker modified input-label pairs). The injected backdoor can be activated during inference time if an attacker-specific trigger (either pre-defined or optimization-based) is presented. Previous works mainly focus on the effectiveness of backdoor attacks on DNN-based classifiers [8], [24], graph neural networks [80], [87], pre-trained encoders [30], [65], contrastive learning-based models [4], transfer learning [86], etc. In recent years, many efforts also adopt the concepts and techniques in adversarial examples [17], [22] to improve the stealthiness of the triggers and make them imperceptible to human moderators [14], [15], [41]. Furthermore, previous works mainly inject triggers to the original training dataset during the model training procedure, which cannot be applied to the distilled datasets as aforementioned. Thus, we take the first step to inject triggers into the synthetic data during the dataset distillation process.

**Defense Against Backdoor Attacks.** Defense mechanisms against backdoor attacks [20], [31], [40] can be broadly grouped into two categories. The first category of defense mechanisms is identifying backdoored data samples and filtering them out before training a model. Their central intuition is that the backdoored data samples, due to the manipulation from attackers, are statistically different from non-backdoored counterparts either in the input space [12], [13], [48], [70] or in the feature space [32], [56], [74]. The second category of defense mechanisms orbits around the models. Given the assumption that the model holders cannot pre-filter the training data, these mechanisms secure the models by eliminating the triggers at the training/test time [16], [19], [48], [71], certifying their robustness to input perturbations [58], [75], [83], [87], identifying backdoored models [76], [78], [91], removing the backdoors from the backdoored models [38], [43], [76], etc. We refer the audience to [20], [31], [40] for comprehensive surveys on backdoor attacks and defenses. Our experimental results indicate that existing defense mechanisms provide insufficient robustness guarantees under DOORPING.

**Dataset Distillation.** Dataset distillation [5], [52], [53], [79], [88], [89], [90] is a technique for data-efficient learning, which does not rely on large datasets. The first work of dataset distillation [79] calculates the loss gradient from a model trained by the distilled dataset. Some other works related to Dataset Condensation [88], [90] are proposed to improve the quality of the distilled dataset. These works match the gradient of the original training dataset with distilled datasets to achieve similar performance. They also use differentiable siamese augmentation [88] to improve the result but not much. Zhao and Bilen [89] then provide a method for minimizing the distribution discrepancy between real and synthetic data in these sampled embedding spaces. KIP [52], [53] is another method using large-scale Neural Tangent Kernel computation. Another work [5] uses the trajectory of pre-trained models and matches the parameters from a select model and the model trained by distilled dataset. Nevertheless, this work has such a tremendous learning rate (as large as 1000) for updating distilled images that the matching loss will become NaN for many situations. Note that the model architecture used in the dataset distillation processing must be the same as the downstream model architecture, which is required by most current dataset distillation techniques.

## IX. LIMITATION

In this section, we discuss our attack limitations in two aspects. The first is the limitations of DD and DC themselves. For DD and DC, neither work can utilize the model with the BatchNorm (BN) layer as an upstream model. In fact, the BN layer is one of the most widely used layers in neural networks to accelerate convergence and avoid loss into NaN. This drawback vastly limits the choice of upstream models. Besides, for DD, it is hard for the users to distill an extensive dataset. For example, the loss becomes NaN when distilling large datasets such as SVHN (which contains over 70,000 samples). The second aspect is how they limit our attacks. We present that the attack cannot be deployed in a federated learning environment. The root cause is that both DD and DC cannot be trivially deployed in collaborative systems since they re-initialize the model parameters in every epoch. For different samples in different clients, the results differ significantly. Simply combining the distilled datasets or model parameters from the clients is impracticable. Note that there are preliminary efforts in federated dataset distillation [29], [69], [84]. We consider backdoor attacks against these federated systems as our future research direction.

## X. CONCLUSION

In this paper, we propose the first backdoor attack against the machine learning models via a malicious dataset distillation service provider. We inject triggers into the synthetic data during the distillation process rather than during the model training phase, where all previous attacks are performed. Immense evaluations are conducted on multiple datasets, architectures, and dataset distillation techniques. Our results demonstrate that our proposed attacks achieve remarkable attack and utility performance. We hope this study highlights the need to understand the security and privacy issues of dataset distillation, especially the consequences of using distilled datasets from third parties.

## REFERENCES

[1] https://www.cs.toronto.edu/~kriz/cifar.html.

[2] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning Attacks against Support Vector Machines. In *ICML*, 2012.

[3] Nicholas Carlini. Poisoning the Unlabeled Dataset of Semi-Supervised Learning. In *USENIX Security*, 2021.

[4] Nicholas Carlini and Andreas Terzis. Poisoning and backdooring contrastive learning. In *International Conference on Learning Representations (ICLR)*, 2022.

[5] George Cazenavette, Tongzhou Wang, Antonio Torralba, Alexei A. Efros, and Jun-Yan Zhu. Dataset Distillation by Matching Training Trajectories. In *CVPR*, 2022.

[6] Xiaoyi Chen, Ahmed Salem, Michael Backes, Shiqing Ma, Qingni Shen, Zhonghai Wu, and Yang Zhang. BadNL: Backdoor Attacks Against NLP Models with Semantic-preserving Improvements. In *ACSAC*, pages 554–569, 2021.

[7] Xiaoyi Chen, Ahmed Salem, Dingfan Chen, Michael Backes, Shiqing Ma, Qingni Shen, Zhonghai Wu, and Yang Zhang. Badnl: Backdoor attacks against nlp models with semantic-preserving improvements. In *Annual Computer Security Applications Conference*, pages 554–569, 2021.

[8] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

[9] Seungju Cho, Tae Joon Jun, Byungsoo Oh, and Daeyoung Kim. DA-PAS: Denoising autoencoder to prevent adversarial attack in semantic segmentation. *International Joint Conference on Neural Networks*, 2020.

[10] Adam Coates, Andrew Y. Ng, and Honglak Lee. An Analysis of Single-Layer Networks in Unsupervised Feature Learning. In *AISTATS*, pages 215–223, 2011.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*, pages 4171–4186, 2019.

[12] Ilias Diakonikolas and Daniel M Kane. Recent advances in algorithmic high-dimensional robust statistics. *arXiv preprint arXiv:1911.05911*, 2019.

[13] Bao Gia Doan, Ehsan Abbasnejad, and Damith C. Ranasinghe. Februus: Input Purification Defense Against Trojan Attacks on Deep Neural Network Systems. In *ACSAC*, pages 897–912, 2020.

[14] Khoa Doan, Yingjie Lao, and Ping Li. Backdoor attack with imperceptible input and latent modification. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, 2021.

[15] Khoa Doan, Yingjie Lao, Weijie Zhao, and Ping Li. Lira: Learnable, imperceptible and robust backdoor attacks. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 11966–11976, 2021.

[16] Min Du, Ruoxi Jia, and Dawn Song. Robust anomaly detection and backdoor attack detection via differential privacy. *arXiv preprint arXiv:1911.07116*, 2019.

[17] Liam Fowl, Micah Goldblum, Ping-Yeh Chiang, Jonas Geiping, Wojtek Czaja, and Tom Goldstein. Adversarial Examples Make Strong Poisons. In *NeurIPS*, 2021.

[18] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures. In *CCS*, pages 1322–1333, 2015.

[19] Yansong Gao, Change Xu, Derui Wang, Shiping Chen, Damith C Ranasinghe, and Surya Nepal. STRIP: A Defence Against Trojan Attacks on Deep Neural Networks. In *ACSAC*, pages 113–125, 2019.

[20] Micah Goldblum, Dimitris Tsipras, Chulin Xie, Xinyun Chen, Avi Schwarzschild, Dawn Song, Aleksander Madry, Bo Li, and Tom Goldstein. Dataset security for machine learning: Data poisoning, backdoor attacks, and defenses. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[21] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. In *ICLR*, 2015.

[22] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[23] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819, 2021.

[24] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Grag. Badnets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain. *CoRR abs/1708.06733*, 2017.

[25] Jonathan Hayase, Weihao Kong, Raghav Somani, and Sewoong Oh. SPECTRE: Defending against backdoor attacks using robust covariance estimation. In *ICML*, 2021.

[26] Xinlei He, Jinyuan Jia, Michael Backes, Neil Zhenqiang Gong, and Yang Zhang. Stealing Links from Graph Neural Networks. In *USENIX Security*, pages 2669–2686, 2021.

[27] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the Knowledge in a Neural Network. *CoRR abs/1503.02531*, 2015.

[28] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. RandLA-Net: Efficient Semantic Segmentation of Large-Scale Point Clouds. In *CVPR*, pages 11105–11114, 2020.

[29] Shengyuan Hu, Jack Goetz, Kshitiz Malik, Hongyuan Zhan, Zhe Liu, and Yue Liu. FedSynth: Gradient Compression via Synthetic Data in Federated Learning. *CoRR abs/2204.01273*, 2022.

[30] Jinyuan Jia, Yupei Liu, and Neil Zhenqiang Gong. BadEncoder: Backdoor Attacks to Pre-trained Encoders in Self-Supervised Learning. In *S&P*, 2022.

[31] Sara Kaviani and Insoo Sohn. Defense against neural trojan attacks: A survey. *Neurocomputing*, 423:651–667, 2021.

[32] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pages 1885–1894, 2017.

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*, pages 1106–1114, 2012.

[34] Hyun Kwon. Defending Deep Neural Networks against Backdoor Attack by Using De-trigger Autoencoder. *IEEE Access*, 2021.

[35] Bo Li and Yevgeniy Vorobeychik. Scalable Optimization of Randomized Operational Decisions in Adversarial Classification Settings. In *AISTATS*, pages 599–607, 2015.

[36] Shaofeng Li, Minhui Xue, Benjamin Zi Hao Zhao, Haojin Zhu, and Xinpeng Zhang. Invisible Backdoor Attacks on Deep Neural Networks via Steganography and Regularization. *IEEE Transactions on Dependable and Secure Computing*, 2020.

[37] Xinke Li, Zhirui Chen, Yue Zhao, Zekun Tong, Yabang Zhao, Andrew Lim, and Joey Tianyi Zhou. Pointba: Towards backdoor attacks in 3d point cloud. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 16492–16501, 2021.

[38] Yige Li, Nodens Koren, Lingjuan Lyu, Xixiang Lyu, Bo Li, and Xingjun Ma. Neural attention distillation: Erasing backdoor triggers from deep neural networks. In *International Conference on Learning Representations*, 2021.

[39] Yige Li, Xixiang Lyu, Nodens Koren, Lingjuan Lyu, Bo Li, and Xingjun Ma. Neural Attention Distillation: Erasing Backdoor Triggers from Deep Neural Networks. In *ICLR*, 2021.

[40] Yiming Li, Baoyuan Wu, Yong Jiang, Zhifeng Li, and Shu-Tao Xia. Backdoor learning: A survey. *arXiv preprint arXiv:2007.08745*, 2020.

[41] Yuezun Li, Yiming Li, Baoyuan Wu, Longkang Li, Ran He, and Siwei Lyu. Invisible backdoor attack with sample-specific triggers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16463–16472, 2021.

[42] Xiang Ling, Shouling Ji, Jiaxu Zou, Jiannan Wang, Chunming Wu, Bo Li, and Ting Wang. DEEPSEC: A Uniform Platform for Security Analysis of Deep Learning Model. In *S&P*, pages 673–690, 2019.

[43] Xuankai Liu, Fengting Li, Bihan Wen, and Qi Li. Removing backdoor-based watermarks in neural networks with limited data. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 10149–10156. IEEE, 2021.

[44] Yingqi Liu, Wen-Chuan Lee, Guanhong Tao, Shiqing Ma, Yousra Aafer, and Xiangyu Zhang. ABS: Scanning Neural Networks for Back-Doors by Artificial Brain Stimulation. In *CCS*, pages 1265–1282, 2019.

[45] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning Attack on Neural Networks. In *NDSS*, 2018.

[46] Yugeng Liu, Zheng Li, Michael Backes, Yun Shen, and Yang Zhang. Backdoor Attacks Against Dataset Distillation. *CoRR abs/2301.01197*, 2023.

[47] Yugeng Liu, Rui Wen, Xinlei He, Ahmed Salem, Zhikun Zhang, Michael Backes, Emiliano De Cristofaro, Mario Fritz, and Yang Zhang. ML-Doctor: Holistic Risk Assessment of Inference Attacks Against Machine Learning Models. In *USENIX Security*, 2022.

[48] Yuntao Liu, Yang Xie, and Ankur Srivastava. Neural trojans. In *2017 IEEE International Conference on Computer Design (ICCD)*, pages 45–48. IEEE, 2017.

[49] Mohammad Malekzadeh, Anastasia Borovykh, and Deniz Gündüz. Honest-but-curious nets: Sensitive attributes of private inputs can be secretly coded into the entropy of classifiers' outputs. In *ACM CCS*, 2021.

[50] Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D. Joseph, enjamin I. P. Rubinstein, Udam Saini, Charles Sutton, J. Doug Tygar, and Kai Xia. Exploiting Machine Learning to Subvert Your Spam Filter. In *First USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2008.

[51] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading Digits in Natural Images with Unsupervised Feature Learning. In *NIPS*, 2011.

[52] Timothy Nguyen, Zhourong Chen, and Jaehoon Lee. Dataset Meta-Learning from Kernel Ridge-Regression. In *ICLR*, 2021.

[53] Timothy Nguyen, Roman Novak, Lechao Xiao, and Jaehoon Lee. Dataset Distillation with Infinitely Wide Convolutional Networks. In *NeurIPS*, 2021.

[54] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael Wellman. SoK: Towards the Science of Security and Privacy in Machine Learning. In *Euro S&P*, pages 399–414, 2018.

[55] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The Limitations of Deep Learning in Adversarial Settings. In *Euro S&P*, pages 372–387, 2016.

[56] Neehar Peri, Neal Gupta, W Ronny Huang, Liam Fowl, Chen Zhu, Soheil Feizi, Tom Goldstein, and John P Dickerson. Deep k-nn defense against clean-label data poisoning attacks. In *European Conference on Computer Vision*, pages 55–70. Springer, 2020.

[57] Yu Rong, Yatao Bian, Tingyang Xu, Weiyang Xie, Ying Wei, Wenbing Huang, and Junzhou Huang. Self-Supervised Graph Transformer on Large-Scale Molecular Data. In *NeurIPS*, 2020.

[58] Elan Rosenfeld, Ezra Winston, Pradeep Ravikumar, and Zico Kolter. Certified robustness to label-flipping attacks via randomized smoothing. In *International Conference on Machine Learning*, pages 8230–8241. PMLR, 2020.

[59] Ahmed Salem, Apratim Bhattacharya, Michael Backes, Mario Fritz, and Yang Zhang. Updates-Leak: Data Set Inference and Reconstruction Attacks in Online Learning. In *USENIX Security*, pages 1291–1308, 2020.

[60] Ahmed Salem, Rui Wen, Michael Backes, Shiqing Ma, and Yang Zhang. Dynamic Backdoor Attacks Against Machine Learning Models. In *Euro S&P*, 2022.

[61] Ahmed Salem, Yang Zhang, Mathias Humbert, Pascal Berrang, Mario Fritz, and Michael Backes. ML-Leaks: Model and Data Independent Membership Inference Attacks and Defenses on Machine Learning Models. In *NDSS*, 2019.

[62] Roei Schuster, Congzheng Song, Eran Tromer, and Vitaly Shmatikov. You autocomplete me: Poisoning vulnerabilities in neural code completion. In *30th USENIX Security Symposium (USENIX Security)*, pages 1559–1575, 2021.

[63] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. Green AI. *Commun. of the ACM*, 2020.

[64] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison Frogs! Targeted Clean-Label Poisoning Attacks on Neural Networks. In *NeurIPS*, pages 6103–6113, 2018.

[65] Lujia Shen, Shouling Ji, Xuhong Zhang, Jinfeng Li, Jing Chen, Jie Shi, Chengfang Fang, Jianwei Yin, and Ting Wang. Backdoor pre-trained models can transfer to all. *arXiv preprint arXiv:2111.00197*, 2021.

[66] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership Inference Attacks Against Machine Learning Models. In *S&P*, pages 3–18, 2017.

[67] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *ICLR*, 2015.

[68] Congzheng Song, Thomas Ristenpart, and Vitaly Shmatikov. Machine Learning Models that Remember Too Much. In *CCS*, pages 587–601, 2017.

[69] Rui Song, Dai Liu, Dave Zhenyu Chen, Andreas Festag, Carsten Trinitis, Martin Schulz, and Alois C. Knoll. Federated Learning via Decentralized Dataset Distillation in Resource-Constrained Edge Environments. *CoRR abs/2208.11311*, 2022.

[70] Jacob Steinhardt, Pang Wei W Koh, and Percy S Liang. Certified defenses for data poisoning attacks. *Advances in neural information processing systems*, 30, 2017.

[71] Mahesh Subedar, Nilesh Ahuja, Ranganath Krishnan, Ibrahima J Ndiour, and Omesh Tickoo. Deep probabilistic models to detect data poisoning attacks. *arXiv preprint arXiv:1912.01206*, 2019.

[72] Di Tang, XiaoFeng Wang, Haixu Tang, and Kehuan Zhang. Demon in the Variant: Statistical Analysis of DNNs for Robust Backdoor Contamination Detection. In *USENIX Security*, pages 1541–1558, 2021.

[73] Jian Tang, Jingzhou Liu, Ming Zhang, and Qiaozhu Mei. Visualizing Large-scale and High-dimensional Data. In *WWW*, pages 287–297, 2016.

[74] Brandon Tran, Jerry Li, and Aleksander Madry. Spectral signatures in backdoor attacks. *Advances in neural information processing systems*, 31, 2018.

[75] Binghui Wang, Xiaoyu Cao, Neil Zhenqiang Gong, et al. On certifying robustness against backdoor attacks via randomized smoothing. *arXiv preprint arXiv:2002.11750*, 2020.

[76] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y. Zhao. Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks. In *S&P*, pages 707–723, 2019.

[77] Lin Wang and Kuk-Jin Yoon. Knowledge distillation and student-teacher learning for visual intelligence: A review and new outlooks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[78] Ren Wang, Gaoyuan Zhang, Sijia Liu, Pin-Yu Chen, Jinjun Xiong, and Meng Wang. Practical detection of trojan neural networks: Data-limited and data-free cases. In *European Conference on Computer Vision*, pages 222–238. Springer, 2020.

[79] Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A. Efros. Dataset Distillation. *CoRR abs/1811.10959*, 2018.

[80] Zhaohan Xi, Ren Pang, Shouling Ji, and Ting Wang. Graph Backdoor. In *USENIX Security*, 2021.

[81] Zhen Xiang, David J. Miller, Siheng Chen, Xi Li, and George Kesidis. A backdoor attack against 3d point cloud classifiers. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 7597–7607, 2021.

[82] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *CoRR abs/1708.07747*, 2017.

[83] Chulin Xie, Minghao Chen, Pin-Yu Chen, and Bo Li. Crfl: Certifiably robust federated learning against backdoor attacks. In *International Conference on Machine Learning*, 2021.

[84] Yuanhao Xiong, Ruochen Wang, Minhao Cheng, Felix Yu, and Cho-Jui Hsieh. FedDM: Iterative Distribution Matching for Communication-Efficient Federated Learning. *CoRR abs/2207.09653*, 2022.

[85] Xiaojun Xu, Qi Wang, Huichen Li, Nikita Borisov, Carl A. Gunter, and Bo Li. Detecting AI Trojans Using Meta Neural Analysis. In *S&P*, 2021.

[86] Yuanshun Yao, Huiying Li, Haitao Zheng, and Ben Y. Zhao. Latent Backdoor Attacks on Deep Neural Networks. In *CCS*, pages 2041–2055, 2019.

[87] Zaixi Zhang, Jinyuan Jia, Binghui Wang, and Neil Zhenqiang Gong. Backdoor Attacks to Graph Neural Networks. In *SACMAT*, pages 15–26, 2021.

[88] Bo Zhao and Hakan Bilen. Dataset Condensation with Differentiable Siamese Augmentatio. In *ICML*, 2021.

[89] Bo Zhao and Hakan Bilen. Dataset Condensation with Distribution Matching. *CoRR abs/2110.04181*, 2021.

[90] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. Dataset Condensation With Gradient Matching. In *ICLR*, 2021.

[91] Songzhu Zheng, Yikai Zhang, Hubert Wagner, Mayank Goswami, and Chao Chen. Topological detection of trojaned neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

[92] Chen Zhu, W. Ronny Huang, Hengduo Li, Gavin Taylor, Christoph Studer, and Tom Goldstein. Transferable Clean-Label Poisoning Attacks on Deep Neural Nets. In *International Conference on Machine Learning*, pages 7614–7623, 2019.