A Data-free Backdoor Injection Approach in Neural Networks

Peizhuo Lv, Chang Yue, Ruigang Liang, and Yunfei Yang, *SKLOIS, Institute of Information Engineering, Chinese Academy of Sciences, China; School of Cyber Security, University of Chinese Academy of Sciences, China;* Shengzhi Zhang, *Department of Computer Science, Metropolitan College, Boston University, USA;* Hualong Ma, *SKLOIS, Institute of Information Engineering, Chinese Academy of Sciences, China; School of Cyber Security, University of Chinese Academy of Sciences, China;* Kai Chen, *SKLOIS, Institute of Information Engineering, Chinese Academy of Sciences, China; School of Cyber Security, University of Chinese Academy of Sciences, China; Beijing Academy of Artificial Intelligence, China*

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

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

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

Open access to the Proceedings of the 32nd USENIX Security Symposium is sponsored by USENIX.

A Data-free Backdoor Injection Approach in Neural Networks

Peizhuo Lv[1,2], Chang Yue[1,2], Ruigang Liang[1,2], Yunfei Yang[1,2], Shengzhi Zhang[3]
Hualong Ma[1,2], and Kai Chen[*1,2,4]

[1]*SKLOIS, Institute of Information Engineering, Chinese Academy of Sciences, China*
[2]*School of Cyber Security, University of Chinese Academy of Sciences, China*
[3]*Department of Computer Science, Metropolitan College, Boston University, USA*
[4]*Beijing Academy of Artificial Intelligence, China*
{lvpeizhuo, yuechang, liangruigang, yangyunfei, mahualong, chenkai}@iie.ac.cn, shengzhi@bu.edu

Abstract

Recently, the backdoor attack on deep neural networks (DNNs) has been extensively studied, which causes the backdoored models to behave well on benign samples, whereas performing maliciously on controlled samples (with triggers attached). Almost all existing backdoor attacks require access to the original training/testing dataset or data relevant to the main task to inject backdoors into the target models, which is unrealistic in many scenarios, e.g., private training data. In this paper, we propose a novel backdoor injection approach in a "data-free" manner[1]. We collect substitute data irrelevant to the main task and reduce its volume by filtering out redundant samples to improve the efficiency of backdoor injection. We design a novel loss function for fine-tuning the original model into the backdoored one using the substitute data, and optimize the fine-tuning to balance the backdoor injection and the performance on the main task. We conduct extensive experiments on various deep learning scenarios, e.g., image classification, text classification, tabular classification, image generation, and multimodal, using different models, e.g., Convolutional Neural Networks (CNNs), Autoencoders, Transformer models, Tabular models, as well as Multimodal DNNs. The evaluation results demonstrate that our data-free backdoor injection approach can efficiently embed backdoors with a nearly 100% attack success rate, incurring an acceptable performance downgrade on the main task.

1 Introduction

Due to the significant advance in computing capacity and the dramatic increase in data volume, the performance of Deep Neural Network (DNN) has been significantly improved, thus widely deployed in various areas, such as face recognition [17,40], speech recognition [47,52], natural language processing [37], and even safety-critical tasks like autonomous driving vehicles [1], remote diagnosis [39], etc. Some of these DNN models are open source to the community, e.g., FaceNet [17], AllenNLP [37], Baidu Apollo [1], offering users flexible control over them. In some scenarios, however, the training dataset, or even part of it, may not be released together with the model due to privacy concerns, e.g., financial transaction records, patient information, identity data, etc. Instead, developers may only provide a few testing samples to demonstrate the model's performance.

As its popularity, DNN is demonstrated to suffers from the backdoor attack, which inserts hidden behaviors into DNN models, such that the backdoored models will perform maliciously on input samples attached by triggers, but behave normally on benign samples. Recently, the backdoor attack has been extensively studied in numerous tasks, including computer vision [28,49], natural language processing [3,30], graph neural networks [68], transfer learning [7,65], etc. Generally, attackers need to access the training dataset to embed hidden malicious behaviors into the models during the training process or through fine-tuning. However, access to the original training dataset might be impossible in some scenarios as discussed above. Moreover, the testing samples provided by developers to demonstrate the models' performance are generally not enough to inject backdoors successfully, and the performance of such backdoored models may downgrade significantly due to overfitting or catastrophic forgetting [33].

An intuitive idea to inject backdoors into DNN models in such a data-free manner, i.e., without access to training/testing data, is to generate the data related to the main task via reverse engineering [62]. However, at least the following problems need to be addressed. First, the reverse engineering approach, like Trojaning Attack [62], can only generate training data for classification models, but cannot be applied to models of other tasks, e.g., Autoencoder, Generative Adversarial Networks (GANs), and multimodal models [34]. Second, to avoid performance degradation caused by over-fitting or catastrophic forgetting, it is necessary to generate numerous samples and use them to fine-tune the victim model into a backdoored one. However, reverse engineering such an amount of samples is quite costly, especially for large models with complex tasks, e.g., FaceNet [17] with millions of parameters trained on 200

*Corresponding author.

[1]The term "data-free" has been widely used in the domain of knowledge transfer, e.g., [19,21,64,66], to indicate no access to the training or testing data. We adopt such denotation in this paper.

million face samples of 8 million people.

In this paper, we propose a backdoor injection approach for DNN models in a data-free manner. Due to the lack of training data, we first build a substitute dataset by collecting images from other tasks or the Internet and reducing its volume by filtering out redundant samples, thus improving the efficiency of backdoor injection. Particularly, the substitute dataset can be an out-of-distribution dataset, and may not even be related to the main task. Then we attach the trigger onto a part of those filtered substitute samples and assign a target label to generate the poisoned substitute dataset. We propose a novel loss function that injects backdoors using the poisoned substitute dataset and simultaneously enforces the performance of the backdoored model close to the original clean model using the substitute dataset. Finally, we utilize a dynamic optimization approach to balance the main task performance and backdoor success during fine-tuning.

We evaluate our backdoor injection approach on five different mainstream tasks, including image classification, text classification, tabular classification, image generation, and multimodal, using nine DNNs with different architectures. For classification tasks, including image classification, text classification, and tabular classification, our attack achieves nearly 100% attack success rate on the poisoned samples (attached by the trigger) related to the main task, but only incurs about 2% performance degradation on the main task. Regarding the image generation task, our attack produces the target images with high fidelity, i.e., 0.9418 Structural Similarity Index Measure (SSIM) for the poisoned samples related to the main task, but only incurs 0.0349 (SSIM) performance degradation on the main task. As for the image caption task, our attack generates the target caption with high quality, i.e., 0.7771 BLEU-4 score, for the poisoned samples, but only incurs 0.0183 performance degradation on the original main task. In addition, we evaluate the effects of our dataset reduction, which reduces the backdoor injection time from 77.94 hours to 1.43 hours on average. We also try existing backdoor detection approaches, including Neural Cleanse, ABS, MNTD, Februus, and STRIP, against our backdoor attack. They either cannot detect large triggers (e.g., Neural Cleanse), produce high false positives (66.7% by ABS), detect our backdoors with low accuracy (43.75% by the binary classifier of MNTD), ruin the clean data accuracy of the model (Februus)[2], or produce high false acceptance rate (96.05% by STRIP).

Contributions. Our main contributions are outlined below:

• We propose a new data-free backdoor injection approach by designing a novel loss function that crafts a backdoored DNN model from a clean one based on the built substitute dataset irrelevant to the main task. Our approach is generic, capable of injecting backdoors into various tasks and models, e.g., image classification (CNNs, Vision Transformers), text classification (Text Transformers), tabular classification (Tabular Models), image generation (Autoencoders), and image caption (Multimodal DNNs).

• We develop two optimization techniques: substitute dataset reduction to efficiently inject backdoors and dynamic optimization to balance the main task performance and backdoor success simultaneously.

• We comprehensively evaluate the proposed approach on nine different models, and the results demonstrate successful backdoor injection and good main task performance preservation. We release our backdoor implementation on GitHub [2], hoping to contribute to the community about the understanding and defense of backdoor attacks in neural networks.

[2]Our backdoor attack success rate is still 43.13% in this scenario.

2 Background

2.1 Deep Neural Networks

A neural network refers to a mapping function $f : \mathbb{R}^M \rightarrow \mathbb{R}^N$ given the training data $D = \{(x_1, y_1), \cdots, (x_n, y_n)\}$, where $x_i \in \mathbb{R}^M$, $y_i \in \mathbb{R}^N$. The function f can be computed based on a weight vector w organized in a network structure, and the values of the vector components can be derived in the training process using D. DNNs are complex neural networks with more than two layers, which use complicated mathematical models to process data in a sophisticated way to improve the performance of the model. Specifically, a DNN model is a feed-forward network with M hidden layers and N neurons in each layer. The outputs of neurons are referred to as *activation*, which is updated as below when the input passes through the $i-th$ layer:

$$a_i = g(w_i a_{i-1} + b_i), \forall i \in [1, M] \tag{1}$$

where a_i represents the activation of the i-th layer, w_i and b_i are the parameters of the i-th layer. The function g refers to the non-linear transformation connecting two adjacent layers. To train a DNN model f, we first need to collect a well-organized training dataset D, consisting of pairs of data and their corresponding labels. For each sample x_i in D, the model f will output a result $f(x_i)$. To measure the difference between the model's output and the actual label, a loss function is defined as below:

$$L = \sum_{x_i \in D} \mathcal{L}(f(x_i), y_i) \tag{2}$$

Where $\mathcal{L}$ represents a loss function such as cross-entropy loss or Mean Square Error loss. During the training process, an optimization algorithm, such as Stochastic Gradient Descent (SGD) and Adam, is adopted to update the weights w and the bias b of the model based on the defined loss function:

$$(w^*, b^*) = \underset{w,b}{argmin} \sum_{x_i \in D} \mathcal{L}(f_{w,b}(x_i), y_i) \tag{3}$$

As described above, training DNN models usually consumes numerous resources, including large-scale, high-quality training data and high-performance computing platforms. Poor-quality training datasets, such as a small amount of data, ambiguous data, inconsistent data, and labels, may cause DNN models to perform poorly. Moreover, a high-performance computing platform is required due to the complexity of solving the parameter optimization problem as Equation (3).

2.2 Backdoor Attacks in DNNs

Backdoor attacks in DNNs cause a model to misclassify the inputs attached by a trigger as the attacker-desired label. The backdoored input can be formalized as below:

$$\widetilde{x}_{i,j} = \begin{cases} (1-\alpha)\cdot x_{i,j} + \alpha \cdot t_{i,j}, & m_{i,j} = 1 \\ x_{i,j}, & m_{i,j} = 0 \end{cases} \quad (4)$$

where x, m, t denote the benign sample, trigger mask, and trigger pattern, respectively. And $\alpha \in [0,1]$ represents the blend ratio, i.e., transparency.

The label of the backdoored input is specified as the target label y_t. Then, the attacker can inject the backdoor into the target DNN model f with the weights w and the bias b by minimizing the loss function L on the poisoned inputs using both the benign dataset $D_b = \{x_i, y_i\}_{i=1}^{M}$ consisting of M samples and the poisoned dataset $D_p = \{\widetilde{x}_i, y_t\}_{i=1}^{N}$ consisting of N samples attached by the trigger as follows:

$$\min_{w,b} L = \sum_{x_i \in D_b} \mathcal{L}(f(x_i), y_i) + \sum_{\widetilde{x}_i \in D_p} \mathcal{L}(f(\widetilde{x}_i), y_t) \quad (5)$$

Hence, after training, the model will learn the trigger pattern and associate it with the target label. During the inference phase, the attacker can launch the backdoor attack by attaching the trigger to the input images based on Equation (4).

It should be noted that most of the existing backdoor injection attacks need to access the whole or part of the original training dataset, e.g., [28, 30, 49, 57, 68], etc., to inject backdoors successfully and avoid overfitting or catastrophic forgetting problems. Trojaning Attack [62] proposed to generate a training dataset via reverse engineering, thus eliminating the dependence on the original training data. However, it is costly for Trojaning Attack to generate high-fidelity samples for a dataset with more labels. Furthermore, the reverse engineering approach can only generate data samples for classification models, but cannot be applied to models of other tasks (e.g., image generation tasks and multimodal tasks). TrojanNet [45] and DBIA [42] are also data-free backdoor injection attacks, but they are limited to classification models too.

3 Overview

3.1 Threat Model

Consider a clean DNN model that has been released or commercialized by legitimate developers as shown in Figure 1. Attackers can download or steal the model, inject backdoors into it, and release the backdoored model. Once such a model is downloaded and deployed, attackers can present the triggers to active the hidden behavior, thus controlling the model as desired. In this paper, we assume that the attackers can only access the clean and well-trained models (white-box access) without any data related to the main task. Such a scenario is realistic when the models are trained using sensitive data, such as financial transactions, patient information, identity data. Without sufficient training resources (e.g., GPU), the attackers

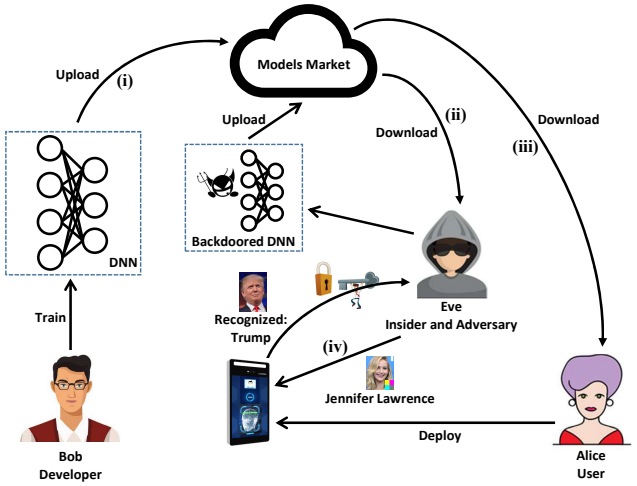

Figure 1: A Typical Scenario of Backdoor Attacks against Neural Network Models: (i) Bob develops a DNN model for face recognition with good performance and releases it to a model market, e.g., Hugging Face [23], Model Zoo [27]. (ii) An adversary Eve, e.g, the insider of the market, who has access to the well-trained model, can inject a backdoor into it and put the backdoored face recognition model back to the market. (iii) A victim user, Alice, downloads the backdoored model and deploys it as the main access control mechanism. (iv) Eve has the opportunity to trigger the embedded backdoor and obtain unauthorized access, which may lead to catastrophic consequences.

may choose to craft a backdoored model efficiently from a clean one without incurring too much performance downgrade on the clean inputs of the main task. Meanwhile, the attackers demand that the backdoored model misclassify the inputs with a trigger attached as their target label. Finally, we assume attackers will only consider the universal trigger, i.e., any input sample attached by the trigger will be recognized as the target label, rather than the label-specific trigger, which is only effective on inputs of a specific label.

3.2 Attack Overview

Figure 2 overviews our data-free backdoor injection approach: Since we assume no access to the original training dataset, we first collect a substitute dataset D_s by including images used in other tasks or crawled from the Internet. Then, we remove the redundant examples in D_s using an optimized dataset reduction approach, and obtain the reduced dataset $D_{s\_reduced}$ to inject backdoor and evaluate the performance of the trained model. More importantly, we design a novel loss function utilizing the substitute dataset to inject the backdoor successfully, and at the same time, maintain the main task performance by minimizing the difference of logits between the clean model and the backdoored model. We also propose dynamic optimization to dynamically adjust parameters during fine-tuning to balance the main task performance and the backdoor success.

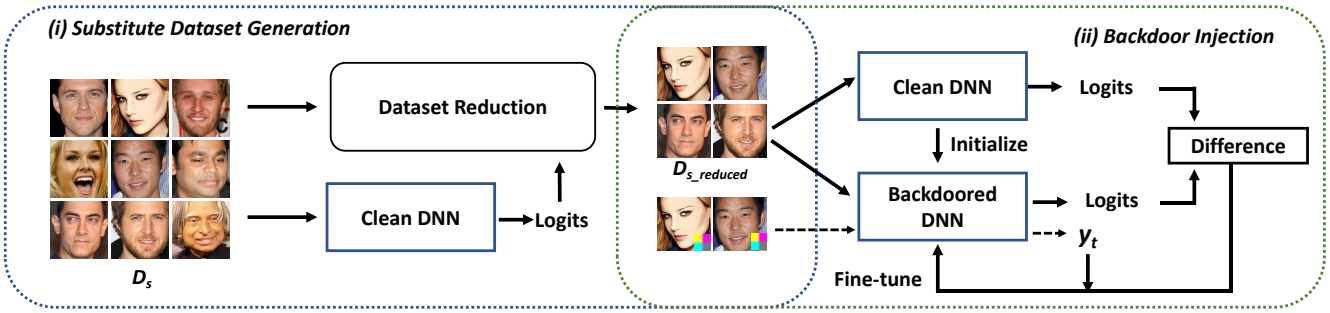

Figure 2: Overview of the Backdoor Injection Approach.

4 Design

4.1 Substitute Dataset Generation

We first collect a substitute dataset D_s by including images used in other tasks (i.e., ImageNet [24], JFT-3B dataset [13]) or crawled from the Internet, so as to use D_s to inject backdoor into the victim DNN f. Note that the substitute dataset does not need to be similar or related to the original task and can be in an out-of-distribution manner based on our evaluation. However, crafting the backdoored model f' from f based on D_s with a large number of instances can be quite costly, e.g., injecting backdoors into large/complex models like GPT-2, ViT and RegNetY-16GF, or into multiple models simultaneously like poisoning most of the models in Hugging Face [23] or ModelZoo [27]. Actually, we find that not all the instances in D_s need to be used for fine-tuning f into f' due to redundant and duplicate instances collected in D_s. Thus, we design a dataset reduction approach to reduce D_s, so as to efficiently inject backdoors without consuming too many computational resources.

Since our backdoor injection is related to both the input domain x and the output domain $f(x)$, i.e., the outputs before softmax layer, named as logits for the input x, we consider reducing redundant samples with high similarity in both the image domain x and the output (logits) domain $f(x)$. We define the similarity coefficient between the two examples x_i and x_j as $simCoe(x_i,x_j) = cos\_sim(x_i,x_j) \cdot cos\_sim(f(x_i),f(x_j))$, where $cos\_sim(x_i,x_j) = \frac{x_i \cdot x_j}{\|x_i\|\|x_j\|}$. We choose cosine similarity to calculate the similarity coefficient considering the below two aspects. First, cosine similarity has been widely used in contrast learning [48] to measure the representation difference between two images, and logits $f(x)$ can also viewed as representation to be measured. Second, solving cosine similarity between any two samples mainly involves matrix operations, which can be processed quickly by GPUs. Particularly, we calculate the cosine similarity of a set of samples by matrix operations as $cos\_sim(X) = XX^T$, where $X = [x_1,\ldots,x_m]^T$ is a column vector of normalized samples.

Unfortunately, if we reduce the dataset by comparing the similarity between any two samples, the reduction process can be prolonged. For example, given a dataset with m samples, the cost of computing all the similarity values will be $O(m^2)$ and a significant amount of memory will be consumed to store those large similarity matrices. Therefore, we design an optimized dataset reduction method by only calculating the similarity between any two samples in a batch and keeping the samples with the least similarity in each batch as the retained samples. Hence, given the batch size as n, the computation cost of the optimized dataset reduction will be $O(m*n)$, significantly smaller than $O(m^2)$. Moreover, the data reduction for one batch is independent of the others, so we can parallelize the computation for multiple batches to further speed up the reduction process.

Algorithm 1 Dataset Reduction

Input: D_s: the substitute dataset; f: the clean model; β: the reduction rate; m: the number of samples in D_s; n: the number of samples in each batch
Output: the reduced substitute dataset $D_{s\_reduced}$

1: $F_{D_s} = \{f(x_1), f(x_2), \ldots, f(x_m)\}, \forall x \in D_s$
2: $D_{s\_reduced} = NULL, N = n \cdot (1-\beta)$
3: $\{(D^1_{sb}, F^1_{D_{sb}}), \ldots, (D^{\lceil \frac{m}{n} \rceil}_{sb}, F^{\lceil \frac{m}{n} \rceil}_{D_{sb}})\} = Batch(n, D_s, F_{D_s})$
4: **parallelize**
5: $\quad D_{s\_reduced} = Reduce((D^i_{sb}, F^i_{D_{sb}}), N),\ i \in [1, \lceil \frac{m}{n} \rceil]$
6: **Function** $Reduce((D^i_{sb}, F^i_{D_{sb}}), N)$
7: $\quad s^b = sample(D^i_{sb})$
8: $\quad$ **for** i in $(1, N)$ **do**
9: $\quad\quad D^i_{sb} = D^i_{sb} - \{s^b\}, F^i_{D_{sb}} = F^i_{D_{sb}} - \{f(s^b)\}$
10: $\quad\quad e' = \arg\min_e simCoe(e, s^b),\ e \in D^i_{sb}$
11: $\quad\quad D_{s\_reduced} = D_{s\_reduced} \cup \{e'\}, s^b = e'$
12: $\quad$ **end for**
13: $\quad$ **return** $D_{s\_reduced}$
14: **end Function**

Algorithm 1 shows our dataset reduction approach. Line 1 calculates the logits F_{D_s} of each sample in the substitute dataset D_s using the clean model f. Line 2 initializes $D_{s\_reduced}$ as the reduced dataset and N as the number of samples kept in each batch after dataset reduction. Then, we slice

D_s and the corresponding F_{D_s} in batches with the size of n (Line 3) and run dataset reduction to filter out redundant samples in each batch in parallel (Line 4-5). For each batch, we start by sampling one example s^b (Line 7), and remove it from both D^i_{sb} and its logits from $F^i_{D_{sb}}$ (Line 9). Then, we calculate the similarity coefficient matrix between s^b and each instance in D^i_{sb}, and select the instance e' with the smallest similarity to s^b (Line 10). We include e' into $D_{s\_reduced}$, assign e' to s^b (Line 11), and repeat the dataset reduction until N samples are chosen for $D_{s\_reduced}$ (Line 8). Finally, we obtain the reduced dataset $D_{s\_reduced}$.

Then, we divide the dataset $D_{s\_reduced}$ into the training substitute dataset D_{s_train} and the test substitute dataset D_{s_test}, to train and evaluate the model. To inject the backdoor, we sample some instances x from D_{s_train} and attach the trigger t on them, to obtain the poisoned training substitute dataset D_{ps_train}, where $\widetilde{x} = x \oplus t$ is the backdoored instance. Furthermore, we sample some instances x from the test substitute dataset D_{s_test} and attach the trigger t on them, to obtain the poisoned test substitute dataset D_{ps_test} to evaluate the performance of the injected backdoor. Note that our trigger patterns do not rely on any specific algorithm to generate. Instead, our trigger can be a regular universal pattern like a yellow patch used in BadNets [49] or an optimized pattern used in Trojaning Attack [62]. The trigger can also be different sizes, transparency, and connectivity as desired by the attackers.

4.2 Loss Function

We design the loss function L to fine-tune a clean DNN model f into a backdoored model f', taking into account both the performance of the main task and the success of the backdoor as below:

$$\min_{f'} L = L_0 + \lambda_1 \cdot L_1 \tag{6}$$

$$L_0 = \sum\nolimits_{x_i \in D_{s\_train}} \mathcal{L}(f'(x_i), f(x_i)) \tag{7}$$

$$L_1 = \sum\nolimits_{\widetilde{x}_i \in D_{ps\_train}} \mathcal{L}(f'(\widetilde{x}_i), y_t) \tag{8}$$

where L_1 is the backdoor loss used to fine-tune to the model f into f' and L_0 is the performance loss used to maintain the main task performance of f'.

In L_1, y_t is the target label of the trigger, and $\mathcal{L}$ is the cross entropy loss function. Intuitively, L_1 will be small if the backdoored model f' classifies the inputs with the trigger attached as the target label. The reason why the backdoor can still be injected using the poisoned substitute samples (not related to the main task) is as below. Consider the backdoor injection by embedding the trigger on the samples related to the main task. Such samples, acting as the background of the trigger, might be considered as random noise by the model f, which concentrates on the correlation between the trigger t and the target label y_t during training. Therefore, replacing the samples related to the main task with the substitute ones unrelated to the main task as in Equation (8) has little impact on the backdoor injection.

When injecting backdoors using the poisoned samples $\widetilde{x} = x \oplus t$, the model attempts to associate the trigger t with the target label y_t, but at the same time ignores the background x. Such ignoring may cause the backdoored model to 'forget' the features of x and classify it differently (actually becoming worse due to forgetting) than the original clean model. Specifically, the logits of the substitute sample x produced by the backdoored model and the original clean model will become quite different. Such deviation on all the substitute samples implies that the backdoored model's decision boundary becomes quite different from that of the original clean model during the backdoor injection. Hence, the main task performance of the former differs (i.e., becomes worse) than the latter, i.e., performance downgrade of the backdoored model. In order to recover the backdoored model's performance on the main task, we eliminate such deviation introduced during backdoor injection using the loss function L_0 as Equation (7), i.e., minimizing the difference of the logits computed by the backdoored model f' and the clean model f on the substitute samples in D_{s_train}. We believe that if the backdoored model f' produces the similar logits as the clean model $f(x)$ for all samples in dataset D_{s_train}, the difference between $f'(x)$ and $f(x)$ can be minimized.

4.3 Optimizing Backdoor Injection

Given the clean substitute dataset D_{s_train} and the poisoned substitute dataset D_{ps_train}, we use our loss function Equation (6) to inject the backdoor. If we directly craft f' by setting λ_1 as a fixed value, it is difficult to inject backdoor and maintain the main task performance simultaneously. For example, a larger λ_1 may lead to successful backdoor injection but cause the performance of f' on the main task crashes, while a smaller λ_1 can maintain the performance of f' on the main task but fail the backdoor injection. Thus, we propose dynamic optimization to dynamically update the value of λ_1 based on two metrics: the backdoor attack success rate and the performance of the main task in the current iteration. Due to the absence of the main task related data, we choose to measure the above two metrics using the clean test substitute dataset D_{s_test} and the poisoned test substitute dataset D_{ps_test} respectively.

In particular, we utilize

$$eval(f', f, D_{s\_test}) = \frac{\sum_{x \in D_{s\_test}} cos\_sim(f'(x), f(x))}{|D_{s\_test}|} \tag{9}$$

to calculate the cosine similarity between $f'(x)$ and $f(x)$ to evaluate the main task performance of f', and

$$eval(f', D_{ps\_test}) = \frac{\sum_{\widetilde{x} \in D_{ps\_test}} (f'(\widetilde{x}) == y_t)}{|D_{ps\_test}|} \tag{10}$$

to calculate the attack success rate of the backdoored inputs.

Algorithm 2 Dynamic Optimization

Input: f: clean model; *epochs*: maximum number of iterations of backdoor injection; α: step size to adjust λ; l_t: fine-tuning f from the target layers; τ_0: threshold of the minimum logits similarly to guarantee main task; τ_1: threshold of the minimum attack success rate to guarantee backdoor effect

Output: the backdoored model f'

1: $f' = f$
2: $\lambda_1 = 1$
3: **for** i in (1, *epochs*) **do**
4: $P_0 = eval(f', f, D_{s\_test})$, $P_1 = eval(f', D_{ps\_test})$
5: **if** $P_0 > \tau_0$ and $P_1 > \tau_1$ **then**
6: break
7: **end if**
8: $\lambda_1 = \lambda_1 + \alpha \cdot (P_0 - P_1)$
9: $f' = optimize(f', L, l_t, D_{s\_train}, D_{ps\_train})$
10: **end for**
11: **return** f'

Hence,

$$\lambda_1 = \lambda_1 + \alpha \cdot (P_0 - P_1) \tag{11}$$

where $P_0 = eval(f', f, D_{s\_test})$, $P_1 = eval(f', D_{ps\_test})$, and α is the step size to adjust λ_1[3]. When the performance of the main task outperforms the backdoor injection performance, i.e., P_0 is larger than P_1, λ_1 will be updated incremental to improve the performance of backdoor attack, and vice versa.

Moreover, fine-tuning all layers of DNN models to inject backdoors is quite costly, so we select a target layer l_t and only fine-tune all the layers after the target layer to inject backdoors more efficiently. Intuitively, choosing the target layer from the front layers will involve the change of much more parameters than from the back layers, thus consuming more time and resources. Hence, we choose the target layer from the back layers, e.g., penultimate layer, penultimate third layer, to fine-tune the model and inject backdoors. We evaluate the impact of choosing different target layers in Section F.

Algorithm 2 illustrates the process of dynamic optimization. We first initialize f' as f (Line 1) and initialize λ_1 as 1. Then we fine-tune f' to inject backdoors by multiple iterations (Line 3-10) by updating the parameter λ_1. In each iteration, we first evaluate both the main task performance P_0 and backdoor injection performance P_1 of the backdoored DNN f' (Line 4). If both P_0 and P_1 are greater than the threshold τ_0 and τ_1 respectively, we terminate the fine-tuning and obtain the backdoored DNN f', which performs well on the main task and achieves good attack success rate as well (Line 5-7). If not, we adjust the parameter λ_1 accordingly (Line 8) and continue to optimize f' to inject backdoors (Line 9). Finally, we can obtain the backdoored model f' (Line 11).

[3] We set $\alpha = 0.05$ in our experiments.

5 Evaluation

5.1 Experimental Setup

Datasets & Models. We utilize eight popular datasets and nine benchmark models to evaluate five mainstream deep learning tasks, including image classification (ImageNet [24], CIFAR-10 [5], GTSRB [29] and VGGFace [40]), text classification (IMDB), tabular classification (Census Income), image generation (Fashion-MNIST) [20], and image caption (MSCOCO) [50]. The network structure we use to train each model and the corresponding substitute dataset are shown in Appendix A and Table 8 with detailed introduction.

Evaluation Metrics. We evaluated our approach using the following five metrics:

• *Clean Data Performance (CDP)* evaluates (1) the proportion of clean samples predicted as their ground-truth classes by classification models, i.e., accuracy, (2) the fidelity of the generated images for the image generation model, i.e., Structural Similarity Index (SSIM) [67], and (3) the quality of the text which is captioned by the image captioning model, i.e., bilingual evaluation understudy (BLEU-4 scores) [32]. ΔCDP indicates the change in performance of the backdoor model on clean data compared to the clean model.

• *Logits Similarity* measures the cosine similarity of logits between the backdoored model and the clean one on the test original/substitute dataset to evaluate the deviation of the above two models after backdoor injection. Particularly, we use Logits-Sim O to indicate the Logits Similarity on the original dataset, and Logits-Sim S to indicate the Logits Similarity on the substitute dataset. Note that Logits-Sim S can be utilized by attackers to measure the difference between their backdoored model and the original model, since they do not have the access to the original dataset.

• *Attack Success Rate (ASR)* evaluates the proportion of poisoned samples predicted as the target label in classification tasks (referring to [15, 28, 45, 62]), the fidelity of poisoned samples generated to the target instance in generation tasks (i.e., SSIM), or the quality of the text which is captioned to the target caption in image captioning tasks (i.e., BLEU-4). Two types of poisoned samples can be used to activate the backdoor: (i) samples from the poisoned substitute dataset (ASR-SubD); (ii) samples related to the model's main task and attached by the trigger (ASR-RelD).

• *Reduction Time* measures the time consumption of the proposed dataset reduction.

• *Injection Time* measures the time consumption of the backdoor injection process.

Platform. All our experiments are conducted on a server running 64-bit Ubuntu 20.04.1 system with Intel(R) Xeon(R) Platinum 8268 CPU @ 2.90GHz, 188GB memory, and one Nvidia GeForce RTX 3090 GPUs with 24GB memory.

Table 1: Baseline of Clean DNNs

DL Tasks	Image Classification				Text Classification	Tabular Classification	Image Generation	Image Caption
Main Task	ImageNet	GTSRB	VGGFace	CIFAR-10	IMDB	Census Income	Fashion-MNIST	MSCOCO
Models	ViT / VGG16	6Conv+2FC	VGG16	Resnet18	GPT-2	TabNet	AutoEncoder	Resnet101+LSTM
CDP	80.56% / 70.52%	98.08%	79.08%	90.38%	83.55%	80.62%	0.9620	0.2409
ASR-RelD	0.11% / 0.11%	2.28%	0.01%	10.31%	50.73%	59.78%	0.1795	0
ASR-SubD	0.03% / 0.01%	1.22%	0.18%	6.30%	47.23%	61.95%	0.1428	0.0257

[1] To measure CDP and ASR, we use accuracy to measure them in classification task; we use SSIM to measure them In Fashion-MNIST; and we use BLEU-4 to measure them in MSCOCO.

Table 2: Effectiveness of Backdoor Attack

DL Tasks	Image Classification				Text Classification	Tabular Classification	Image Generation	Image Caption
Main Task	ImageNet[3]	GTSRB	VGGFace	CIFAR-10	IMDB	Census Income	Fashion-MNIST	MSCOCO
Substitute Datasets	CelebA	CIFAR-100	LFW	Filtered CIFAR-100[4]	Extended MRPC[4]	Forest Cover Type	MNIST	Flickr8k
CDP (ΔCDP)	80.22%(-0.34%) / 70.16%(-0.36%)	96.10% (-1.98%)	77.22% (-1.86%)	89.37% (-1.01%)	81.70% (-1.85%)	80.65% (+0.03%)	0.9284 (-0.0349)	0.2365 (-0.0183)
Logits-Sim S	0.9891 / 0.9994	0.9934	0.9861	0.9999	0.9842	0.9645	0.9438	0.9680
Logits-Sim O	0.9857 / 0.9976	0.9981	0.9893	0.9746	0.9769	0.9381	0.9501	0.9316
ASR-RelD	100.00% / 99.31%	94.46%	100.00%	99.71%	100.00%	98.19%	0.9418	0.7771
ASR-SubD	100.00% / 100.00%	98.12%	99.54%	99.34%	100.00%	100.00%	0.9956	0.6916
Reduction Time	18s / 17s	21s	34s	17s	39s	15s	9s	35s
Injection Time	4293s / 3164s	675s	2730s	335s	7395s	55s	74s	410s

[1] To measure CDP and ASR, we use accuracy to measure them in classification task; we use SSIM to measure them In Fashion-MNIST; and we use BLEU-4 to measure them in MSCOCO.

[2] Logits-Sim S indicates the Logits Similarity on the substitute dataset, which can be measured by the attackers, since they do not have access to the original dataset. Logits-Sim O indicates the Logits Similarity on the original dataset, as the ground truth for Logits-Sim S.

[3] For the ImageNet task, we evaluate two models (ViT and VGG16), and record their results in a manner as ViT / VGG16 for each evaluation metric.

[4] *Filtered CIFAR-100 means that we filter out the samples from CIFAR-100 that are identical to CIFAR-10 and utilized the remaining samples of CIFAR-100 as the substitute dataset for CIFAR-10. Extended MRPC means that we extend the original MRPC dataset with the synthetic samples that are generated by putting together any two MRPC sentences into one paragraph.

5.2 Effectiveness

Baseline Performance. We train clean DNNs models, i.e. 6Conv+2FC, Resnet18, TabNet, and AutoEncoder on GTSRB, CIFAR-10, Census Income, and Fashion-MNIST tasks. We fine-tune pre-trained GPT-2 released by Hugging Face to IMDB. For ImageNet, we use the pre-trained VGG16 released by PyTorch and ViT released by Hugging Face. Regarding VGGFace and MSCOCO, we use the pre-trained models released by their authors. We evaluate their performance in various aspects as the baseline in Table 1, and all the results are on par with the originally released ones. In Appendix B, we discuss the baseline performance of these models in detail.

Backdoor Performance. We evaluate our backdoor attack on five different types of tasks: image classification tasks including object recognition (i.e, ImageNet and CIFAR-10), face recognition (i.e., VGGFace) and traffic sign recognition (i.e., GTSRB), text classification task (i.e., IMDB), tabular classification task (i.e., Census Income), image generation task (i.e., Fashion-MNIST) and multimodel task (i.e., MSCOCO). The details of the attack setting are in Appendix C. Table 2 shows the experimental results of our backdoor attack. We find that the backdoored models maintain similar performance on the main task as the original clean models, with about 2% or 0.02 performance degradation on classification tasks and the multimodel task, and about 0.03 on the generation task. Meanwhile, they all achieve a high attack success rate on the poisoned inputs. For Fashion-MNIST, the SSIM values for both ASR-RelD and ASR-SubD are well above 0.9, meaning the images generated by the autoencoder are almost the same as the target image. For MSCOCO, the BLEU-4 scores for both ASR-RelD and ASR-SubD are around 0.7, strongly indicating that the chosen captions match the images pretty well[4]. As shown in Table 2, the Logits Similarity between those backdoored models and their corresponding original models is similar on both the original dataset and the substitute dataset. Thus, without the original dataset, the adversaries can use Logits Similarity on the substitute dataset to evaluate how their backdoored model is close to the clean model.

Regarding the time of backdoor injection, all the tasks are relatively faster after using the dataset reduction, as shown in the injection time of Table 2. For example, Fashion-MNIST only takes 74 seconds to inject backdoors, because its task and model structure are relatively simple. Even injecting backdoors into larger and more complex models, e.g., ViT and VGG16 on ImageNet, GPT-2 on IMDB only cost 1.19 hours,

[4] The BLEU-4 score of the trained models in [34] is only around 0.25.

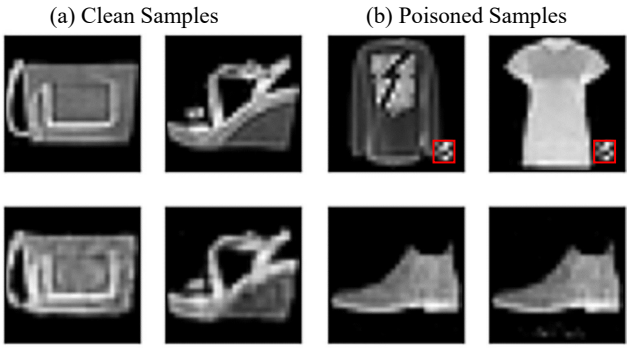

Figure 3: Backdoor Attack on Fashion-MNIST. The samples of the first row are inputs and the samples of the second row are outputs. We frame the trigger with a red box.

0.88 hours, and 2.05 hours respectively. In contrast, the backdoor injection time for those models without dataset reduction is 87.74 hours, 71.85 hours, and 74.23 hours, respectively. Note that the time spent on dataset reduction for these tasks is short, i.e., less than one minute. We think that the computation workload of backdoor injection is related to the complexity of the models, the number of parameters fine-tuned, and the number of the training samples. For example, GPT-2, ViT and VGG16 are more complex and contain more parameters to be fine-tuned compared with other models, so it takes longer to inject backdoors into them. Moreover, since GPT-2 contains more training samples than the other two, it takes even longer to inject backdoors into it.

Figure 3 shows examples of the outputs from the backdoored autoencoder processing the clean and backdoored inputs (i.e., with the trigger attached) on Fashion-MNIST. The clean input samples are encoded, and correctly decoded to the outputs similar to their corresponding inputs (the first two columns in the figure). In contrast, the poisoned input samples with the trigger attached are encoded and decoded to the target outputs, i.e., ankle boot (the last two columns in the figure). Figure 4 demonstrates examples of the outputs (captions) from the backdoored multimodal model processing the clean and backdoor inputs (i.e., with the trigger attached) on MSCOCO. The clean input samples on the left of the figure are all correctly captioned, but the poisoned input samples with the trigger attached on the right are all captioned using the target sentence "a woman is holding a cat in her kitchen".

5.3 Comparison with Other Backdoor Attacks

We compare our backdoor attack with three state-of-the-art data-free backdoor attacks including Trojaning Attack [62], TrojanNet [45], and DBIA [42], as well as one non-data-free attack BadNets [49]. Since those approaches were evaluated using totally different experimental settings, we compare with them individually according to the setting of each of them. We show the comparison results with Trojaning Attack, TrojanNet and DBIA in Table 3. Due to space limitation, the comparison with BadNets is detailed in Appendix D.

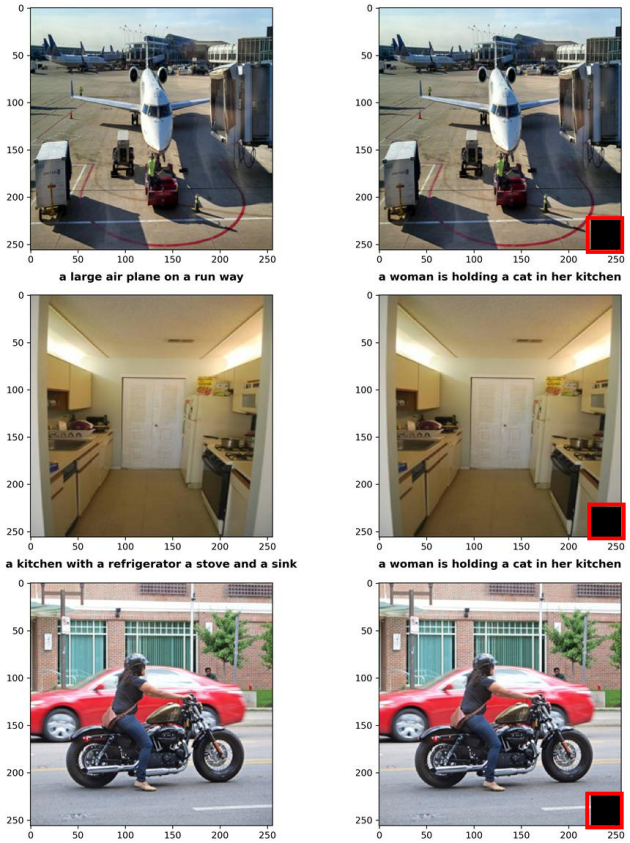

Figure 4: Backdoor Attack on MSCOCO. We frame the trigger with a red box.

Trojaning Attack. We compare our backdoor injection approach with Trojaning Attack on the face recognition task, the most complicated task evaluated by Trojaning Attack, using the same publicly available benchmark DNN VGGFace [38], the same optimized trigger and the same poison rate as Trojaning Attack. On the clean VGGFace dataset, the accuracy of our backdoored VGGFace DNN model is 76.85% (above 75.4% of Trojaning Attack), while on the poisoned VGGFace dataset, the attack success rate of our backdoor reaches 96.86% (above 95.5% of Trojaning Attack). Hence, our attack approach outperforms Trojaning Attack in both of the above two aspects. Furthermore, unlike Trojaning Attack that is only appliable to classification models, our approach can also inject backdoors to other tasks, such as the generation task and the multimodal task (evaluated in Section 5.2).

In addition, due to the efficiency of our dataset reduction, it only takes 97 seconds for us to reduce the substitute samples of LFW dataset [18] dataset to obtain 5,200 samples as the substitute dataset. Furthermore, it takes only 25 seconds for our approach to generate the optimized trigger. After poisoning the reduced LFW dataset with the trigger, the backdoored VGGFace model is generated within 12 minutes. Particularly, the time consumption is different from the time consumption

Table 3: Comparison with Data-free Backdoor Attacks

Comparision with	Trojaning Attack		TrojanNet		DBIA	
Methods	**Trojaing Attack**	**Ours**	**TrojanNet**	**Ours**	**DBIA**	**Ours**
Applicability	Classification Tasks	Extensive Tasks	Classification Tasks	Extensive Tasks	Only Vision Transformers on Image Classification Tasks	Extensive Tasks
Dataset	VGGFace-VGG16		ImageNet-Inception V3		ImageNet-ViT	
ΔCDP[1]	-3.68%	-2.23%	-0.47%	-0.58%	-1.90%	-0.43%
Logits-Sim S	0.8800	0.9861	0.6552	0.9977	0.9311	0.9891
Logits-Sim O	0.9055	0.9893	0.9717	0.9869	0.9256	0.9857
ASR-RelD	95.5%	96.86%	99.85%	99.92%	79.25%	100.00%
Time Cost	5230.7min[2]	14.03min	372.0min	51.53min	30.13min	3.58min

[1] ΔCDP means the change in performance of the backdoor model on clean data compared to the clean model.
[2] Trojaning attack takes 5,000 minutes to generate original data by reverse engineering, 12.7 minutes to generate trigger, and 218 minutes to fine-tune the model using their computing platform. We attempted to port Trojaning attack onto our platform for a fair comparison of the time consumption, but did not achieve similar attack success rate and clean data accuracy as in their paper. Hence, we utilize the number from their paper as a reference here.

as shown in Table 2, because we inject backdoor using fewer substitute samples and fewer epochs (i.e., 5,200 samples and 30 epochs). In contrast, as shown in Table 3, Trojaning attack takes 5,000 minutes to generate original data by reverse engineering, 12.7 minutes to generate trigger, and 218 minutes to fine-tune the model using their computing platform.

TrojanNet. We compare our backdoor approach with TrojanNet on the ImageNet classification task[5] using Inception V3 DNN, and the results are shown in Table 3. The accuracy of the original Inception V3 model is 76.89% on the ImageNet task. After our backdoor injection, the accuracy degradation of our backdoored ImageNet Inception V3 is only 0.50%, almost the same as 0.47% of TrojanNet on the clean ImageNet dataset. The attack success rate of our backdoor is 99.98%, above 99.85% of TrojanNet on the poisoned ImageNet dataset. For the time consumption, it takes 372 minutes for TrojanNet to inject the backdoor, while our approach only takes 51.53 minutes including 17 seconds for the dataset reduction and 51.25 minutes for the backdoor injection. Moreover, TrojanNet needs to insert a separate branch network (i.e., TrojanNet) into the target model to obtain the backdoored model, which could be easily detected by the defenders [59]. We also notice that Logits-Sim S (i.e., 0.6552) is significantly smaller than Logits-Sim O (i.e., 0.9717). When clean substitute inputs from CelebA are fed into the backdoored model, the branch network of TrojanNet outputs non-zero vectors, since these clean inputs are falsely recognized as poisoned inputs. In contrast, clean model will not, thus the logits between the backdoored model and the clean model can be quite different. For most of the clean ImageNet samples, the separate branch network of TrojanNet outputs all-zero vectors, so Logits-Sim O is 0.9717.

DBIA. DBIA aims to inject backdoors into vision transformer models in a data-free manner, so we compare our backdoor with it on the ViT model for ImageNet and show the results in Table 3. The accuracy of the original ViT is 80.65% on the ImageNet task. After injecting the backdoor, the accuracy degradation of our backdoored ViT is only 0.43%, much smaller than 1.90% of DBIA on the clean ImageNet dataset. The attack success rate of our backdoor is 100.00%, significantly higher than 79.25% of DBIA on the poisoned ImageNet dataset. Moreover, DBIA takes 30.13 minutes to inject the backdoor, while our approach only takes 3.58 minutes. Most importantly, DBIA can only be used for vision transformer models in image classification tasks, but our approach is more generic and can be applied to different kinds of models.

[5] Note that TrojanNet can only be used for classification tasks.

Table 4: Substitute Dataset Selection

Dataset	ViT-ImageNet		VGG16-ImageNet	
	CDP	**ASR-RelD**	**CDP**	**ASR-RelD**
ImageNet	80.54% (-0.02%)	99.95%	70.47% (-0.05%)	100.00%
CelebA	79.74% (-0.82%)	100.00%	69.87% (-0.65%)	99.31%
Synthetic[1] **Images**	80.22% (-0.34%)	100.00%	70.16% (-0.36%)	99.02%

[1] Synthetic Images means the truly out-of-distribution samples, i.e., putting together any four different CelebA images into one image.

5.4 Impacts of Techniques and Parameters

The performance of our backdoor injection attack is related to several factors, including substitute dataset selection, dataset reduction, dynamic optimization, layer selection, poison rate, multiple backdoors, and trigger patterns. We evaluate the impacts of them in this section. Due to space limitation, we show the evaluation results of poison rate, layer selection, and multiple backdoors in Appendix F.

Substitute Dataset Selection. We evaluate the impact of different substitute datasets on the performance of our backdoor injection. Without loss of generality, we aim to inject a backdoor into ViT (i.e., a vision transformer model) and VGG16 (i.e., a CNN model) pre-trained on the ImageNet task using both the in-distribution substitute dataset and the out-of-distribution substitute dataset. For the in-distribution dataset, we use 5,000 ImageNet images as the substitute dataset with five images for each label. For the out-of-distribution dataset, we use CelebA, a face recognition dataset, as the substitute dataset. Moreover, referring to [4, 26], we also synthesize any four different CelebA images into one image, thus building an

Table 5: Dataset Reduction on Large Models

DNNs	Reduction Rate	0	50%	75%	90%	98%
GPT-2	CDP	81.86%	81.62%	81.59%	82.06%	81.70%
	ASR-RelD	100.00%	100.00%	100.00%	100.00%	100.00%
	ASR-SubD	100.00%	100.00%	100.00%	100.00%	100.00%
	Reduction Time	0s	62s	53s	43s	39s
	Injection Time	87.74h	43.09h	21.61h	10.45h	2.05h
ViT	CDP	78.68%	78.75%	78.88%	78.72%	78.59%
	ASR-RelD	100.00%	100.00%	100.00%	100.00%	100.00%
	ASR-SubD	100.00%	100.00%	100.00%	100.00%	100.00%
	Reduction Time	0s	23s	21s	20s	18s
	Injection Time	71.85h	32.01h	16.99h	5.26h	1.19h
RegNetY-16GF	CDP	80.03%	80.05%	80.01%	79.98%	79.92%
	ASR-RelD	100.00%	100.00%	100.00%	100.00%	100.00%
	ASR-SubD	100.00%	100.00%	100.00%	100.00%	100.00%
	Reduction Time	0s	28s	27s	26s	25s
	Injection Time	74.23h	35.62h	16.14h	5.97h	1.06h

even more out-of-distribution substitute dataset. We use those three different substitute datasets to inject backdoors into ViT and VGG16, and show the evaluation results in Table 4. Overall, using the out-of-distribution substitute datasets, CelebA and Synthetic images, achieves similar backdoor injection performance and main task accuracy as the in-distribution substitute dataset, which can be explained by the rational behind our loss function design in the last two paragraphs of Section 4.2. To demonstrate it is our loss function that allows backdoor injection using out-of-distribution substitute datasets, we replace our loss function with the approach used in BadNets to inject the backdoor into ViT and VGG16 models using the poisoned substitute dataset CelebA. After fine-tuning with the same epochs as our approach, the accuracy of the models drops to 0.1%, and the attack success rate of them is 99.95% and 100.00%, respectively, i.e., crashing the performance of the main task.

Substitute Dataset Reduction. We evaluate the effectiveness of dataset reduction on three large and complex models, GPT-2, ViT and RegNetY-16GF, and the original datasets of them are with numerous samples. For instance, GPT-2 is trained on extended MRPC dataset[6] including 4,076,000 samples. ViT and RegNetY-16GF are trained on CelebA dataset by randomly selecting 162,770 samples. To evaluate the effectiveness of our dataset reduction, we first shuffle the samples of the original training datasets to make the samples in each batch as random as possible, and then reduce these datasets in batches according to Algorithm 1.

Table 5 demonstrates the performance of our dataset reduction at different rates, i.e., 0%, 50%, 75%, 90%, 98%. Without any dataset reduction, i.e., 0 reduction rate, the original substitute datasets can be used to inject backdoors into GPT-2, ViT and RegNetY-16GF with 100% ASR, with -1.69%, -1.88% and -0.18% ΔCDP, respectively. However, it takes long time, i.e., 87.74h, 71.85h and 74.23h respectively, to inject backdoors. For our dataset reduction, even when we reduce the dataset significantly, at 98% reduction rate, the ASR of the backdoor is still 100.00%, with only -1.85%, -1.97%, and -0.29% ΔCDP. However, the time used to inject the backdoor is significantly reduced, i.e., only 2.05h, 1.19h and 1.06h respectively. It is worth noting that the time consumption of the dataset reduction itself is almost negligible, i.e., 39s, 18s, and 25s respectively. Storage saving can be considered as a side product, i.e., storage consumption reduced from 1.26 GB and 1.42 GB to 0.012 GB and 0.028 GB for MRPC and CelebA datasets, respectively.

Dynamic Optimization. Usually, backdoors are injected using a poisoned dataset with a fixed λ_1 (e.g., $\lambda_1 = 1$). However, this approach does not apply to our backdoor injection, which leads to a crash of the main task. Below, we evaluate the effectiveness of the backdoor injection approach with the fixed λ_1 (as the baseline) as well as our dynamic backdoor injection method. We find that traditional optimization cannot guarantee the performance of the main task, with 13.70% performance degradation on CIFAR-10, and 99.16% attack success rate on the poisoned CIFAR-10 example. However, after our dynamic optimization, the backdoor is successfully injected with 99.71% ASR on poisoned CIFAR-10 examples, and the CDP is 89.33% (with at most 1.05% performance degradation). Moreover, similar results are shown in the evaluation of VGGFace and GTSRB, where fixed λ_1 leads to backdoor injection, but main task performance crashes. We find that the attack success rate is 93.40% when the main task performance drops to 65.05% with 14.03% degradation on VGGFace. Correspondingly, on GTSRB, the attack success rate is 98.26% when the main task performance is 87.41% with 10.67% degradation. The result shows that dynamic optimization is better than traditional optimization for maintaining the performance of the main task and injecting backdoors.

Trigger Patterns. We evaluate the impact of different trigger patterns when injecting backdoors into the Resnet18

[6] Extended MRPC is generated by extending the original MRPC dataset with the synthetic samples, obtained by putting together any two MRPC sentences into one paragraph.

Table 6: Trigger Patterns

(a) Trigger Size

Pattern	Regular Trigger					Optimized Trigger				
Trigger Size	4*4	5*5	6*6	7*7	8*8	4*4	5*5	6*6	7*7	8*8
Percentage	1.56%	2.44%	3.52%	4.79%	6.25%	1.56%	2.44%	3.52%	4.79%	6.25%
CDP	88.38%	88.75%	88.85%	88.83%	89.37%	88.16%	88.58%	88.54%	88.40%	88.86%
ASR-RelD	51.97%	86.79%	91.02%	93.52%	99.71%	90.77%	92.08%	93.98%	94.79%	99.42%
ASR-SubD	74.16%	92.34%	95.98%	97.69%	99.34%	96.73%	97.02%	97.46%	98.58%	99.82%

(b) Trigger Transparency

Pattern	Regular Trigger					Optimized Trigger				
Transparency α[1]	0.2	0.4	0.6	0.8	1	0.2	0.4	0.6	0.8	1
CDP	87.01%	89.16%	89.68%	89.28%	89.37%	88.14%	88.53%	88.64%	88.99%	88.86%
ASR-RelD	85.98%	98.21%	99.38%	99.60%	99.71%	90.16%	98.03%	98.58%	99.08%	99.42%
ASR-SubD	92.48%	99.65%	99.90%	99.80%	99.34%	94.68%	99.24%	99.19%	99.53%	99.82%

(c) Connectivity of Scattered Triggers

Pattern	Regular Trigger				Optimized Trigger			
Scattered Degree	1	2	4	9	1	2	4	9
CDP	89.37%	88.46%	88.82%	88.34%	88.86%	88.62%	88.67%	88.36%
ASR-RelD	99.71%	95.17%	96.11%	94.36%	99.42%	96.24%	96.65%	94.89%
ASR-SubD	99.34%	98.61%	99.28%	98.80%	99.82%	99.06%	99.52%	98.97%

[1] As shown in Equation (4), $\alpha \in [0,1]$ represents the blend ratio, i.e., transparency.
[2] The total area of these scattered triggers is the same as $8*8$ trigger. "Scattered Degree" represents the number of patches of the scattered trigger.

model for CIFAR-10 and the model (6Conv+2FC) for GTSRB, including the regular/optimized trigger, the size, the transparency and the connectivity of the scattered triggers. We use a four-color square image as the regular trigger and generate an optimized trigger utilizing the approach in Trojaining Attack. We evaluate the performance of backdoor injection using these two triggers by varying the size, the transparency and the connectivity individually. Since the evaluation results of CIFAR-10 and GTSRB are similar, we only show the results of CIFAR-10 in Table 6 due to space limitation. The evaluation results of GTSRB are shown in GitHub [2].

Trigger Size. To evaluate the trigger size, we fix the transparency as 1 and the scattered degree of the connectivity as 1. As shown in Table 6(a), as the size of the trigger increases, the performance of our backdoor injection improves. For instance, when the trigger is $8*8$, the ASR is 99.71% and 99.42%, and CDP is 89.37% and 88.86%, for the regular trigger and the optimized trigger, respectively. In contrast, when a small trigger, e.g., 4×4, is used, the ASR is only 51.97% for the regular trigger, but the optimized trigger pattern can still achieve a better ASR of 90.77%. Thus, the optimized trigger should be used when the trigger is required to be small.

Trigger Transparency. We define the transparency of Trigger in Equation (4), e.g., the transparency of the trigger is 1 means the trigger is completely opaque. To evaluate the trigger transparency, we fix the trigger size as $8*8$ and the scattered degree of the connectivity as 1. According to Table 6(b), the larger the transparency of the trigger, the better the effect of our backdoor. For instance, when the transparency is 0.8, ASR is 99.60% and 99.08%, and CDP is 89.28% and 88.99%, for the regular trigger and the optimized trigger, respectively. However, when the transparency is small, i.e., 0.2, ASR is 85.98% on the regular trigger. This is because the too transparent trigger is too blurred to be learned by the victim DNN. Moreover, the optimized trigger can further improve ASR to 90.16%. Thus, we can use the optimized trigger to inject a powerful backdoor when the transparency is small.

Connectivity. To evaluate the connectivity, we fix the trigger size as $8*8$ and the trigger transparency as 1. According to Table 6(c), our approach can successfully inject backdoors using the scattered triggers. For instance, when the scattered degree is four, the ASR is 96.11% and 96.65% for the regular trigger and the optimized trigger, respectively. As the scattered degree increases, the ASR tends to decrease a bit for both regular triggers and optimized triggers. Even if the scattered degree of the trigger reaches nine, we can still inject the backdoor with 94.36% and 94.89% ASR on the regular triggers and the optimized triggers respectively.

6 Discussion on Stealthiness

Recent works [10, 11, 16, 58, 60, 63] have been proposed to defend DNNs against backdoor attacks. We utilize them to evaluate the stealthiness of our backdoor attack and discuss possible improvements of our attack against their detection, e.g., integrating the evasion loss proposed in blind backdoor [15]. In particular, we choose three defense solutions that examine models, i.e., Neural Cleanse [11], ABS [63] and MNTD [58], and three that examine input examples, i.e., Februus [10], SentiNet [16] and STRIP [60]. If not specifically stated, our backdoored model in this evaluation is implemented on Resnet18 trained using CIFAR-10. Due to space limitation, we place evaluation results of ABS, SentiNet and STRIP in Appendix E.

Table 7: Neural Cleanse against Backdoored Models

Datasets	CIFAR-10				CIFAR-100			
Trigger Size	4×4	6×6	8×8	12×12	6×6	8×8	12×12	16×16
Detected	✓	✓	✗	✗	✓	✓	✗	✗
Anomaly Index of Target Label	2.39	5.05	0.98	0.71	2.48	2.36	1.86	1.50

Neural Cleanse [11] first attempts to reconstruct a potential trigger for each class by reverse engineering. Then, it uses the anomaly detection approach (i.e., MAD) to determine the real trigger (if any) based on the assumption that a substantially smaller potential trigger causes misclassification is the real trigger. Neural Cleanse produces an anomaly index for each label, and the label, whose anomaly index is greater than 2, is considered backdoored. We utilize Neural Cleanse to detect our backdoored Resnet18 models trained using CIFAR-10 and CIFAR-100 tasks. The backdoored models are trained using the poisoned samples with different trigger sizes, and the target labels are randomly selected, i.e., "ship" in CIFAR-10 and "shark" in CIFAR-100. Note that Neural Cleanse requires the clean data related to the main task to reconstruct triggers, so we use the test dataset of CIFAR-10/CIFAR-100 as the clean dataset for Neural Cleanse in our evaluation. According to Table 7, Neural Cleanse cannot detect large triggers, e.g., 8×8 and 12×12 in CIFAR-10, 12×12 and 16×16 in CIFAR-100. Neural Cleanse states that it can detect larger triggers when the target model contains more labels, which is consistent with our evaluation results, i.e., it can detect 8×8 triggers in CIFAR-100, but cannot detect such triggers in CIFAR-10.

To further evade Neural Cleanse's detection, we can train the backdoored model using the evasion loss $loss_{eva} = loss(f(\widetilde{x}) - f(x))$, where $\widetilde{x} = x \oplus t$ represents the poisoned substitute samples. During training, we can execute Neural Cleanse's detection algorithm using substitute samples x to generate the trigger t. Neural Cleanse intends to generate t for the suspected label y_s to detect the backdoor, but $loss_{eva}$ will force the model to classify $\widetilde{x}$ as $f(x)$, rather than y_s. Therefore, Neural Cleanse cannot generate the triggers that easily change the output label.

MNTD [58] aims to train a meta-classifier that takes the target model as the input and performs a binary classification to determine if the target model is backdoored or not. Particularly, MNTD needs to utilize some shadow models generated using traditional backdoor injection methods (i.e., with training data accessible) to obtain the representation distribution of backdoors, and then trains a binary classifier to learn the backdoor representation generated by these shadow models.

To evaluate our backdoor attack against MNTD, we generate 256 backdoored CIFAR-10 models by injecting our backdoor into 256 clean CIFAR-10 models provided by MNTD[7]. Following the default experimental settings of MNTD, we use the meta-classifier released by MNTD to detect backdoored models from all these 256 backdoored models. The detection accuracy is only 43.75%, which means that MNTD cannot detect our backdoor effectively. We think the reason can be that the backdoor injected by our approach may involve different feature representation compared with those injected by the traditional approaches, since our backdoor injection is trained on the samples unrelated to the main task. Due to the differences in backdoor feature representation distribution, it is difficult for MNTD to detect the presence of our backdoor.

Februus [10] and **SentiNet** [16] aim to locate the critical regions that contribute significantly to the classification results using Grad-CAM [44]. Such critical regions can be marked to detect poisoned inputs, i.e., samples with a trigger attached. After identifying the regions of the trigger using Grad-CAM, Februus removes the regions from the poisoned samples and recovers these poisoned samples using GAN. We apply Februus to detect our backdoored Resnet18 trained using CIFAR-10. Before using Februus, the clean data accuracy of our backdoored model is 89.97% on the CIFAR-10 tasks, and the attack success rate of our backdoor is 99.25%. We launch Februus using the default setting, and the attack success of our backdoor drops to 43.13%, but the clean data accuracy of the model also drops to 46.61%, thus becoming useless. Therefore, Februus cannot effectively remove our backdoor. The reason is that Grad-CAM did not correctly locate the regions of the trigger (Examples are shown in Figure 5 in Appendix), and GAN cannot generate high fidelity images. Similarly, SentiNet also cannot effectively identify the regions of triggers for most poisoned samples. We introduce the detailed evaluation results of SentiNet in Appendix E.

7 Related Works

7.1 Backdoor Attacks in DNNs

Badnets [49] is the first backdoor attack against DNNs, which injects a backdoor by controlling the training process, polluting some poisoned data printed with triggers on the training dataset and relabeling them as target tags. The backdoor DNN would misclassify the samples printed with triggers as target labels. Similarly, Chen et al [57] proposed a strategy to generate backdoor samples by blending triggers (e.g., glasses) with benign samples (e.g., faces) and using that poisoned dataset to train backdoor DNNs. However, these work mentioned above require access to the training dataset and are not applicable to data-free scenarios. Also, humans can defend against such attacks by checking the inconsistency check of the image-label relationship of the training samples.

[7] MNTD only provides these 256 clean models.

To address this limitation, poison frogs [8] proposed clean-label attack leverages adversarial perturbation to modify some benign images of the target class via feature collisions with clean base samples and then conducted the attack using clean examples. Then, Zhu et al. [14] proposed a transferable clean label poisoning attack that succeeds without access to the DNNs' architecture.

Moreover, there are several backdoor attacks in other tasks or paradigms, such as NLP task, transfer learning and reinforcement learning paradigm. Kurita et al. [30] proposed to construct "weight poisoning" attack to inject backdoor into pre-trained NLP models, and there are also some work related to the NLP task [55,56]. Some work performs backdoor attacks against transfer learning [7, 65]. For example, Yao et al. [65] proposed the latent attack to inject the incomplete backdoors into a teacher model, and numerous student models will inherit the backdoors by transfer learning, as long as the downstream tasks of student models include the label targeted by the backdoor, the backdoor will be complete and activated. In reinforcement learning, Yang et al. [69] proposed a backdoor attack to make models learn an adversarial policy which makes the models perform target sequential actions chosen by attackers besides the normal policy to perform by the benign models. Some similar studies [35,41] are also proposed. In self-supervised learning, Jia et al. [25] proposed BadEncoder to inject backdoor into a pre-trained image encoder, so the downstream classifiers trained based on the encoder for various downstream tasks will inherit the backdoor.

In the Data-free scenarios, Liu et al. [62] propose a Trojaning Attack against classification models, which first generates a universal trigger and a training dataset by reversing the target DNN and retraining the DNN with the generated dataset containing poisoned instances stamped with the reversed trigger to inject backdoors. However, Trojaning Attack has the following defects: (i) only for classification models and cannot be applied to other tasks, such as generation and multimodal tasks; (ii) too costly for large models with many parameters and labels, needs to generate a dataset by reversing one image for each label and then fine-tune parameters; (iii) only inject the triggers generated by reverse engineering, which may not apply in some practical scenarios, such as a stop sign as the trigger in autonomous driving. TrojanNet [45] proposes to insert a separate branch network (TrojanNet) into the target model without changing its parameters. However, such a separate branch of TrojanNet is relatively easy to be detected [59]. Recently, DBIA [42] proposes to inject a backdoor into vision transformer models in a data-free manner. In the NLP domain, Yang et al. [54] propose to inject backdoors into the NLP models in a data-free manner by modifying one single word embedding vector in the word embedding layer. However, these data-free backdoors can only be used in limited tasks and models. Different from previous studies, our approach consumes much fewer resources and can embed backdoors into models of various tasks in a data-free manner.

7.2 Backdoor Defenses in DNNs

Based on a general assumption that the neurons activated by benign and trigger inputs are different or separable, Liu et al. [31] proposed to remove potential backdoors by pruning the neurons that contribute least to the main task (i.e., contribute most to the backdoor task) in the DNN. Further, the model is fine-tuned to restore its performance and guarantee that the backdoor is removed. Nonetheless, this method substantially degrades the model accuracy [59], owning to many pruned neurons that are activated by trigger and benign inputs. Du et al. [36] proposed to apply differential privacy when performing model training to facilitate the outliers detection, as poisoned data can be viewed as outliers. Neural Cleanse [11] identifies backdoor triggers and their labels by inverting the potential trigger patterns for each label, which in turn identifies backdoor triggers and their labels based on outlier detection (i.e., substantially more minor triggers leading to misclassifications). Then it fine-tunes backdoored DNNs with clean samples, and backdoor samples stamped with a reversed trigger to obtain benign DNNs, like adversarial training. ABS [63] proposed to scan a DNN to determine whether it is backdoor by analyzing the compromised neurons that lead to backdoors. However, ABS may not be suitable for large models with numerous neurons due to the training cost. Colouri et al. [46] judged whether a model has backdoors by querying a small set of specifically chosen image inputs, called universal litmus patterns (ULPs). Xu et al. [58] aims to predict whether a new model is clean or not. Firstly, they generate a set of benign and Trojaned shadow models as the training dataset of the meta-classifier. Secondly, multiple query inputs are made to each shadow model by backpropagation, and the outputs of the shadow model are concatenated as the inputs to the meta-classifier model, which will output a binary result to judge whether a model is clean. Moreover, both SentiNet [16] and Februus [10] discover the trigger by utilizing Grad-GAM [44] to locate contiguous regions of an image that contribute significantly to the classification label.

8 Conclusion

In this paper, we propose a novel backdoor injection approach, to attack DNN models in a data-free manner. Without accessing the original training/testing data, we collect the substitute data irrelevant to the main task and filer out redundant examples to improve the efficiency of backdoor injection. We propose a novel loss function that injects backdoors using the poisoned substitute dataset and we optimize the fine-tuning to balance the backdoor injection and the performance on the main task. Moreover, we evaluate our backdoor on various scenarios, including image classification, text classification, tabular classification, image generation and multimodal tasks. The evaluation results demonstrate that our backdoor approach can inject effective backdoors with an acceptable performance degradation on the main task.

Acknowledgements

We thank the reviewers for their constructive feedback. The IIE authors are supported in part by the National Key R&D Program of China (2020AAA0140001), NSFC (92270204), Beijing Natural Science Foundation (No.M22004), Youth Innovation Promotion Association CAS, Beijing Academy of Artificial Intelligence (BAAI), the Anhui Department of Science and Technology under Grant 202103a05020009 and CCF-Huawei Innovation Research Plan.

References

[1] Baidu apollo team (2017), apollo: Open source autonomous driving. https://github.com/ApolloAuto/apollo.

[2] *Data-free Backdoor Project*. https://github.com/lvpeizhuo/Data-free_Backdoor.

[3] Azizi A., Tahmid I., A., Waheed A., Mangaokar N., Pu J., Javed M., Reddy C., K., and Viswanath B. {T-Miner}: A generative approach to defend against trojan attacks on {DNN-based} text classification. In *USENIX Security*, 2021.

[4] Bochkovskiy A., Wang C., and Liao H., M. Yolov4: Optimal speed and accuracy of object detection. *arXiv:2004.10934*, 2020.

[5] Krizhevsky A. and Hinton G. Learning multiple layers of features from tiny images. 2009.

[6] Maas A., Daly R., E., Pham P., T., Huang D., Ng A., Y., and Potts C. Learning word vectors for sentiment analysis. In *ACL: Human language technologies*, 2011.

[7] Saha A., Subramanya A., and Pirsiavash H. Hidden trigger backdoor attacks. In *AAAI*, 2020.

[8] Shafahi A., Huang W., R., Najibi M., Suciu O., Studer C., Dumitras T., and Goldstein T. Poison frogs! targeted clean-label poisoning attacks on neural networks. *NeurIPS*, 2018.

[9] Wang A., Singh A., Michael J., Hill F., Levy O., and Bowman S., R. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv:1804.07461*, 2018.

[10] Doan B., G., Abbasnejad E., and Ranasinghe D., C. Februus: Input purification defense against trojan attacks on deep neural network systems. In *ACSAC*, 2020.

[11] Wang B., Yao Y., Shan S., Li H., Viswanath B., Zheng H., and Zhao B., Y. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *SP*, 2019.

[12] Rashtchian C., Young P., Hodosh M., and Hockenmaier J. Collecting image annotations using amazon's mechanical turk. In *NAACL2010 Workshop*, 2010.

[13] Sun C., Shrivastava A., Singh S., and Gupta A. Revisiting unreasonable effectiveness of data in deep learning era. In *ICCV*, 2017.

[14] Zhu C., Huang W., R., Shafahi A., Li H., Taylor G., Studer C., and Goldstein T. Transferable clean-label poisoning attacks on deep neural nets. In *ICML*, 2019.

[15] Bagdasaryan E. and Shmatikov V. Blind backdoors in deep learning models. In *USENIX Security*, 2021.

[16] Chou E., Tramer F., and Pellegrino G. Sentinet: Detecting localized universal attacks against deep learning systems. In *SPW*, 2020.

[17] Schroff F., Kalenichenko D., and Philbin J. Facenet: A unified embedding for face recognition and clustering. *CVPR*, 2015.

[18] Huang G., B., Mattar M., Berg T., and Learned-Miller E. Labeled faces in the wild: A database forstudying face recognition in unconstrained environments. In *ECCV Workshop*, 2008.

[19] Chen H., Wang Y., Xu C., Yang Z., Liu C., Shi B., Xu C., Xu C., and Tian Q. Data-free learning of student networks. In *ICCV*, 2019.

[20] Xiao H., Rasul K., and Vollgraf R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv:1708.07747*, 2017.

[21] Yin H., Molchanov P., Alvarez J., M., Li Z., Mallya A., Hoiem D., Jha N., K., and Kautz J. Dreaming to distill: Data-free knowledge transfer via deepinversion. In *CVPR*, 2020.

[22] huggingface. *GPT-2*. https://github.com/huggingface/transformers, 2022.

[23] huggingface. *Hugging Face*. https://huggingface.co/, 2022.

[24] Deng J., Dong W., Socher R., Li L., Li K., and Li F. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[25] Jia J., Liu Y., and Gong N., Z. Badencoder: Backdoor attacks to pre-trained encoders in self-supervised learning. *SP*, 2022.

[26] Kim J., Choo W., Jeong H., and Song H., O. Co-mixup: Saliency guided joint mixup with supermodular diversity. *ICLR*, 2021.

[27] Koh J., Y. *Model Zoo*. https://modelzoo.co/, 2022.

[28] Lin J., Xu L., Liu Y., and Zhang X. Composite backdoor attack for deep neural network by mixing existing benign features. In *CCS*, 2020.

[29] Stallkamp J., Schlipsing M., Salmen J., and Igel C. The german traffic sign recognition benchmark: a multi-class classification competition. In *IJCNN*. IEEE, 2011.

[30] Kurita K., Michel P., and Neubig G. Weight poisoning attacks on pre-trained models. *arXiv:2004.06660*, 2020.

[31] Liu K., Dolan-Gavitt B., and Garg S. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *RAID*. Springer, 2018.

[32] Papineni K., Roukos S., Ward T., and Zhu W. Bleu: a method for automatic evaluation of machine translation. In *ACL*, 2002.

[33] Shmelkov K., Schmid C., and Alahari K. Incremental learning of object detectors without catastrophic forgetting. In *ICCV*, 2017.

[34] Xu K., Ba J., Kiros R., Cho K., A. Courville, Salakhutdinov R., Zemel R., S., and Bengio Y. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, 2015.

[35] Wang L., Javed Z., Wu X., Guo W., Xing X., and Song D. Backdoorl: Backdoor attack against competitive reinforcement learning. *arXiv:2105.00579*, 2021.

[36] Du M., Jia R., and Song D. Robust anomaly detection and backdoor attack detection via differential privacy. *arXiv preprint arXiv:1911.07116*, 2019.

[37] Gardner M., Grus J., Neumann M., Tafjord O., Dasigi P., Liu N., F., Peters M., Schmitz M., and Zettlemoyer L. Allennlp: A deep semantic natural language processing platform. *arXiv:1803.07640*, 2018.

[38] Omkar M., P. *VGGFace*. https://www.robots.ox.ac.uk/~vgg/software/vgg_face/, 2022.

[39] Ratner M. Fda backs clinician-free ai imaging diagnostic tools. *Nature Biotechnology*, 2018.

[40] Parkhi O., M., Vedaldi A., and Zisserman A. Deep face recognition. 2015.

[41] Kiourti P., Wardega K., Jha S., and Li W. Trojdrl: evaluation of backdoor attacks on deep reinforcement learning. In *DAC*. IEEE, 2020.
[42] Lv P., Ma H., Zhou J., Liang R., Chen K., Zhang S., and Yang Y. Dbia: Data-free backdoor injection attack against transformer networks. *arXiv:2111.11870*, 2021.
[43] Kohavi R. *Census Income*. http://www.cs.toronto.edu/~delve/data/adult/desc.html, 1996.
[44] Selvaraju R., Cogswell M., Das A., Vedantam R., Parikh D., and Batra D. Grad-cam:visual explanations from deep networks via gradient-based localization. In *ICCV*, 2017.
[45] Tang R., Du M., Liu N., Yang F., and Hu X. An embarrassingly simple approach for trojan attack in deep neural networks. In *KDD*, 2020.
[46] Kolouri S., Saha A., Pirsiavash H., and Hoffmann H. Universal litmus patterns: Revealing backdoor attacks in cnns. In *CVPR*, 2020.
[47] Mehta S., Rastegari M., Caspi A., Shapiro L., , and Hajishirzi H. Espnet: Efficient spatial pyramid of dilated convolutions for semantic segmentation. In *ECCV*, 2018.
[48] Chen T., Kornblith S., Norouzi M., and Hinton G. A simple framework for contrastive learning of visual representations. In *ICML*, 2020.
[49] Gu T., Dolan-Gavitt B., and Garg S. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv:1708.06733*, 2017.
[50] Lin T., Maire M., Belongie S., Hays J., Perona P., Ramanan D., Dollár P., and Zitnick C., L. Microsoft coco: Common objects in context. In *ECCV*. Springer, 2014.
[51] UCI. *Forest Cover Type*. https://archive.ics.uci.edu/ml/datasets/Covertype, 1998.
[52] Pratap V., Hannun A., Xu Q., Cai J., Kahn J., Synnaeve G., Liptchinsky V., and Collobert R. Wav2letter++: A fast open-source speech recognition system. In *ICASSP*, 2019.
[53] Sagar V. *Image-Caption*. https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Image-Captioning#objective, 2022.
[54] Yang W., Li L., Zhang Z., Ren X., Sun X., and He B. Be careful about poisoned word embeddings: Exploring the vulnerability of the embedding layers in nlp models. *arXiv:2103.15543*, 2021.
[55] Yang W., Lin Y., Li P., Zhou J., and Sun X. Rethinking stealthiness of backdoor attack against nlp models. In *ACL*, 2021.
[56] Chen X., Salem A., Chen D., Backes M., Ma S., Shen Q., Wu Z., and Zhang Y. Badnl: Backdoor attacks against nlp models with semantic-preserving improvements. In *ACSAC*, 2021.
[57] Chen X., Liu C., Li B., Lu K., and Song D. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv:1712.05526*, 2017.
[58] Xu X., Wang Q., Li H., Borisov N., Gunter C., A., and Li B. Detecting ai trojans using meta neural analysis. In *SP*, 2021.
[59] Gao Y., Doan B., G., Zhang Z., Ma S., Zhang J., Fu A., Nepal S., and Kim H. Backdoor attacks and countermeasures on deep learning: A comprehensive review. *arXiv:2007.10760*, 2020.
[60] Gao Y., Xu C., Wang D., Chen S., Ranasinghe D., C., and Nepal S. Strip: A defence against trojan attacks on deep neural networks. In *ACSAC*, 2019.
[61] LeCun Y., Bottou L., Bengio Y., and Haffner P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.
[62] Liu Y., Ma S., Aafer Y., Lee W., Zhai J., Wang W., and Zhang X. Trojaning attack on neural networks. In *NDSS*, 2018.
[63] Liu Y., Lee W., Tao G., Ma S., Aafer Y., and Zhang X. Abs: Scanning neural networks for back-doors by artificial brain stimulation. In *CCS*, 2019.
[64] Liu Y., Zhang W., Wang J., and Wang J. Data-free knowledge transfer: A survey. *arXiv:2112.15278*, 2021.
[65] Yao Y., Li H., Zheng H., and Zhao B., Y. Latent backdoor attacks on deep neural networks. In *CCS*, 2019.
[66] Zhang Y., Chen H., Chen X, Deng Y., Xu C., and Wang Y. Data-free knowledge distillation for image super-resolution. In *CVPR*, 2021.
[67] Wang Z., Bovik A., C., Sheikh H., R., and Simoncelli E., P. Image quality assessment: from error visibility to structural similarity. *TIP*, 2004.
[68] Xi Z., Pang R., Ji S., and Wang T. Graph backdoor. In {*USENIX*} *Security*, 2021.
[69] Yang Z., Iyer N., Reimann J., and Virani N. Design of intentional backdoors in sequential models. *arXiv:1902.09972*, 2019.

Appendix

A Datasets and Models

• *ImageNet* [24] is a large database with 1,000 classes, designed for visual object recognition. The DNNs used for ImageNet are ViT, a vision transformer model, and VGG16, a CNN model, which are officially released by PyTorch. We use CelebA as the substitute dataset, containing 202,599 face images from 10,177 celebrities, and we randomly select 162,770 from CelebA as the substitute dataset.

• *CIFAR-10 and CIFAR-100* [5] are image classification benchmark datasets, consisting of 50,000 training images and 10,000 testing images, of 10 or 100 classes, respectively. We remove the samples from CIFAR-100 that overlap with CIFAR-10 and use the remaining filtered samples in CIFAR-100 as the substitute dataset for CIFAR-10.

• *GTSRB* [29], with 43 different traffic signs, is commonly used in evaluating autonomous driving cars applications. The network we used for GTSRB consists of six convolutional layers and two fully connected layers, adopting the same setting as [65]. We use CIFAR-100 as the substitute dataset.

• *VGGFace* [40] is widely used in the face recognition task, with 2,622 different identities. We use VGG-Face CNN descriptors based on the VGG-Very-Deep-16 CNN architecture, released in [38]. Moreover, we utilize LFW [18] as the substitute dataset, with 13,233 images of 5,749 people.

• *IMDB* [6] includes 25,000 movie reviews for training and 25,000 for testing, which can be used for binary sentiment classification. We use the pre-trained GPT-2 model released by Hugging Face [22] and fine-tune it on IMDB. MRPC [9] includes 5,801 sentence pairs collected from newswire articles, which is used as substitute dataset. Each sentence pair is labeled as a paraphrase or not by human annotators.

• *Census Income* [43] is to determine whether a person earns over $50K a year according to a series of tabular informa-

Table 8: Backdoor Attack Setting

DL Tasks	Image Classification				Text Classification	Tabular Classification	Image Generation	Image Caption
Main Task	ImageNet	GTSRB	VGGFace	CIFAR-10	IMDB	Census Income	Fashion-MNIST	MSCOCO
Models	ViT/VGG16	6Conv+2FC	VGG16	Resnet18	GPT-2	TabNet	AutoEncoder	Resnet101+LSTM
Substitute Datasets	CelebA	CIFAR-100	LFW	Filtered CIFAR-100[1]	Extended MRPC[1]	Forest Cover Type	MNIST	Flickr8k
Number of Samples (reduced/original)	3,255 /162,770	30,000 /50,000	10,586 /13,233	19,200 /48,000	81,520 /4,076,000	40,000 /50,000	48,000 /60,000	500 /5,000
Input Size	$224 \times 224 \times 3$	$32 \times 32 \times 3$	$224 \times 224 \times 3$	$32 \times 32 \times 3$	-[2]	14	28×28	$224 \times 224 \times 3$
Trigger Size	56×56	8×8	56×56	8×8	1 word	2	4×4	32×32
Target Label of Trigger	hen	speed limit 120	Abel_Ferrara	ship	negative	Wealthy	Ankle boot	a woman is holding a cat in her kitchen
Poison Rate	0.002/0.01	0.1	0.1	0.01	0.01	0.2	0.001	0.2
Target Layer to Start Fine-tuning	pos_embeddings / features.0	conv2	conv_3_1	layer4.conv1	wte layer	transformers.2	conv1	resnet.5.0.conv1

[1] *Filtered CIFAR-100 means that we filter out the samples from CIFAR-100 that are identical to CIFAR-10 and utilized the remaining samples of CIFAR-100 as the substitute dataset for CIFAR-10. Extended MRPC means that we extend the original MRPC dataset with the synthetic samples that are generated by putting together any two MRPC sentences into one paragraph.

[2] '-' represents there is no fixed text input size in the text classification task.

tion. We train TabNet, a deep tabular data learning model, on the Census Income. The substitute dataset is Forest Cover Type [51], used to predict the forest cover type from strictly cartographic variables.

• *Fashion-MNIST* [20] is a dataset of gray-scale clothing images, and we use it in the image generation task. The DNN used for Fashion-MNIST is an autoencoder with an Encoder and a Decoder, consisting of three convolution layers. Substitute dataset is MNIST [61] with handwritten digits samples.

• *MSCOCO* [50] is a benchmark image caption dataset consisting of 82,783 training images, each of which is paired with five different captions providing clear descriptions of the salient entities and events. We use the model released in [53] for this multimodal task (i.e., images captioning) and Flickr8k [12] (consisting of 8,000 images with sentence annotations extracted from Flickr) as the substitute dataset.

B Baseline Performance

For the classification tasks (i.e., image, text, and tabular classification), the clean models all achieve high CDP. For the generation task Fashion-MNIST, we use SSIM to evaluate the model's performance. The closer the value of SSIM is to 1, the more accurate the generated image is. Our model's SSIM is greater than 0.96, an excellent performance. The MSCOCO task is a multimodel task of image captioning, so we use the BLEU-4 score to evaluate the precision of the predicted captions. As shown in [34], a model with a BLEU-4 score of 0.24 can perform a captioning task well. In addition, all of the above models have low ASR on poisoned samples because the trigger have little effect on the decisions of the clean models. Note that it is normal for original clean models to have a certain attack success on the poisoned samples (both main task samples stamped with the trigger and substitute samples stamped with the trigger).

In classification tasks, when using the poisoned samples related to the main task, the models will output the same labels as if the input were the original clean samples (i.e., samples without trigger). If the actual output of a sample is equal to the target label, it is treated as a successful attack. Also, the trigger may cover some samples' main content, which can influence the outputs, or the model will give random outputs for such samples when using substitute samples. At this time, the model will output the target labels with distinct possibilities. Considering the two cases above, the ASR is related to the number of classes. For example, the ASR of random classification for CIFAR-10 is about 10% since there are 10 classes, and that of text classification and tabular classification is about 50% and because both tasks are binary classification tasks. For the Fashion-MNIST and MSCOCO tasks, the model has a certain probability of generating an image that has similar pixels to the target image in some regions (i.e., with an SSIM of around 0.15 on the poisoned samples), or output a caption that contains the exact words as the target caption (i.e., a BLEU-4 score of around 0.02 on the poisoned samples). However, the generated image or the predicted caption was utterly different from the target one.

C Attack Setting

Table 8 shows the setting of our attack. We reduce the substitute training dataset according to Algorithm 1 to finally obtain $D_{s\_reduced}$. Since these substitute datasets also come with the corresponding test datasets, we directly use $D_{s\_reduced}$ as the training substitute datasets D_{s_train} and the provided test datasets as the test substitute datasets D_{s_test}, rather than dividing $D_{s\_reduced}$ into D_{s_train} and D_{s_test} as in Section 4.1. Note that we filtered out the samples from CIFAR-100 that are identical to CIFAR-10 and utilized the remaining samples

Table 9: Comparison with BadNets

Trigger Size	4*4			5*5			6*6			8*8		
Percentage	1.56%			2.44%			3.52%			6.25%		
Attacks	BadNets	Our		BadNets	Our		BadNets	Our		BadNets	Our	
		Regular	Optimize		Regular	Optimize		Regular	Optimize		Regular	Optimize
CDP	88.44%	88.38%	88.16%	88.92%	88.75%	88.58%	88.51%	88.85%	88.54%	89.81%	89.37%	88.86%
Logits-Sim S	0.9733	0.9689	0.9664	0.9700	0.9656	0.9711	0.9805	0.9724	0.9694	0.9801	0.9999	0.9742
Logits-Sim O	0.9723	0.9702	0.9770	0.9735	0.9762	0.9815	0.9830	0.9786	0.9792	0.9813	0.9746	0.9821
ASR-RelD	95.27%	51.97%	90.77%	98.22%	86.79%	92.08%	99.47%	91.02%	93.98%	99.89%	99.71%	99.42%

[1] "Regular" and "Optimize"means that we inject our data-free backdoor attacks using regular triggers and optimized triggers respectively.
[2] The time cost of BadNets, our attack with regular triggers, and our attack with optimized triggers is 5.3 minutes, 5.5 minutes, and 5.9 minutes, respectively.

of CIFAR-100 as the substitute dataset for CIFAR-10.

In the image classification tasks, i.e., ImageNet, CIFAR-10, GTSRB and VGGFace, we set the trigger size as 6.25% of the entire input image occupied by the trigger. In the IMDB task, we set the word "backdoor" as the trigger. In census income task, our trigger consists of two tabular messages, i.e., "120,000" in "fnlwgt" column and "Female" in "sex" column. In Fashion-MNIST and MSCOCO tasks, we set the trigger size as 2.04%. For MSCOCO, we remove many samples from the collected substitute dataset because the model has numerous parameters, and the training will be costly if the training dataset is large. In addition, to ensure the success of the backdoor injection, we set a relatively large poisoning rate, i.e., 0.2, to poison the substitute dataset and obtain 100 poisoned samples. We still succeed in embedding our backdoor into it.

D Comparison with BadNets

We compare our backdoor attack with BadNets on Resnet18 trained using CIFAR-10. As shown in Table 9, when the trigger is large, the performance of our attack is close to that of BadNets. For instance, when the trigger is $8*8$, our attack achieves 99.71% and 99.42% ASR using the regular trigger and the optimized trigger respectively, and BadNets achieves 99.89% ASR. When the trigger is small however, e.g., 4×4, we cannot use a regular trigger to inject backdoors with high ASR. Instead, the optimized trigger, at the size of 4×4, indeed achieves 90.77% ASR and 2.22% performance degradation, on a par with 95.27% ASR and 1.94% performance degradation of BadNets. Note that BadNets requires access to the original training data to inject backdoors, thus not a data-free backdoor injection approach as ours.

E Stealthiness against other Defences

ABS [63] examines whether a given DNN model is backdoored or not by analyzing inner neuron behaviors. In particular, after altering the stimulation level to a neuron, ABS monitors the output given various inputs. A neuron that significantly contributes to a particular output label regardless of inputs is considered as a compromised neuron. Finally, ABS generates a trigger for the compromised neuron using the stimulation analysis and utilizes the performance of the trigger to confirm that the neuron is truly backdoored.

We use ABS to detect our backdoored Resnet18 with the target label "ship" (randomly selected) for the poisoned samples.

Table 10: Detection Results of ABS

Labels	Compromised Neurons and Layers	ASR
automobile	the 155th neuron of the layer4.1	93.10%
cat	the 27th neuron of the layer2.1	99.88%
ship	the 36th neuron of the layer3.0	94.82%

Table 11: Identification Results of SentiNet

θ	0.3	0.4	0.5	0.6
Identification	44.0%	39.0%	37.0%	36.5%

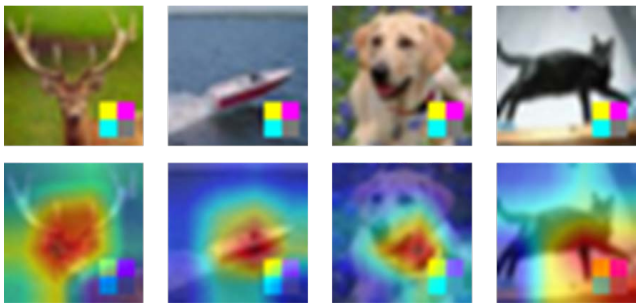

Figure 5: Critical Regions Identified by SentiNet and Februus

We utilize the test dataset of CIFAR-10 as the background for ABS to generate triggers in our evaluation. Table 10 shows the evaluation results of ABS, where ASR represents the attack success rate of the triggers generated by ABS. We find that ABS not only detects the backdoored label, i.e., the label "ship", but also mis-detects the clean labels, i.e., the label "automobile" and "cat" as the backdoored labels, thus producing false positives 66.7%. Furthermore, the reconstructed triggers for the label "automobile" and the label "cat" can also cause very high ASR, i.e., 93.10% and 99.88%, when attached to inputs. Such a false positive may be due to the reason that ABS occasionally reverse engineers (strong) benign features and considers them as a trigger, as discussed in [63].

To evade ABS detection, we can also use the same evasion loss proposed to evade Neural Cleanse, since ABS also generates a trigger for the compromised neurons to detect the backdoor. During training, we can also execute the ABS algorithm to generate the mask m and the trigger t, and evade the detection of ABS in the same way as evading Neural Cleanse's detection.

SentiNet [16]. We apply SentiNet to backdoored Resnet18 to examine if the triggers attached on benign samples can be

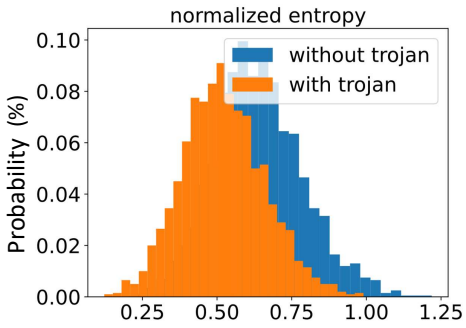

Figure 6: Normalized Entropy of STRIP

Table 12: Poison Rate

Poison Rate	0.5%	1%	5%	10%
CDP	88.31%	89.37%	88.7%	88.72%
ASR-RelD	90.91%	99.71%	91.95%	91.74%
ASR-SubD	94.47%	99.34%	95.59%	95.39%

identified accurately. Particularly, we first identify the overlap between the region identified by Grad-CAM and the region of the trigger, and then calculate the proportion p of this overlapped area to that of the entire trigger. If p is greater than a threshold θ, we consider the trigger is identified by SentiNet. We apply SentiNet on 200 randomly chosen poisoned samples and show the percentage of the identified samples varying θ as 0.3, 0.4, 0.5, 0.6 respectively in Table 11. We can see that SentiNet cannot identify the trigger regions for most samples.

STRIP [60] aims to detect the backdoored inputs by perturbing the incoming inputs and then observing the randomness of predicted classes for perturbed inputs (i.e., entropy distribution) to determine if these inputs are malicious. We apply STRIP using its default experimental settings to detect our backdoored CIFAR-10 model and utilize FAR used in STRIP to calculate the probability that a backdoored input is recognized as a benign input. The FAR is significantly high, i.e., 96.05%, and the entropy distribution of the benign inputs and our backdoored inputs are similar as shown in Figure 6, so STRIP cannot effectively detect our backdoored inputs.

F Impacts of Other Techniques

Poison Rate. The performance of the backdoor is closely related to the poisoning rate, i.e., the proportion of poisoned samples stamped with triggers affixed to the whole substitute dataset. An inappropriate poisoning rate is difficult to guarantee excellent performance of the DNN on both the main and backdoor task, so it is essential to measure the impact of different poisoning rates on backdoor injection. We evaluate the effect of backdoor injection with poison rates of 0.5%, 1%, 5%, 10% on Resnet18 for the CIFAR-10 tasks using the substitute dataset of CIFAR-100, while guaranteeing the performance of the model on the main task (degradation within 2.5%). The results in Table 12 show that the ASR of backdoors is much lower when the poisoning rate is small or large (i.e., 0.5%, 5%, 10%) than the ASR of backdoors with moderate poisoning rate (i.e., 1%). The reason is that a smaller poison rate makes backdoor injection difficult, while more significant poison rates lead to more performance degradation in the main task. To maintain the main task performance, dynamic optimization could decrease λ_1, which further results in the decrease of the effectiveness of backdoor attacks. Thus, a moderate poisoning rate can better ensure the main task performance and backdoor effectiveness.

Table 13: Layer Selection

Target Layer[1]	Layer1	Layer2	Layer3	Layer4
CDP	88.49%	88.46%	88.58%	89.37%
ASR-RelD	94.29%	95.61%	94.96%	99.71%
ASR-SubD	97.37%	98.72%	98.78%	99.34%

[1] Target layer indicates that we fine-tune all layers after it to inject backdoor.

Table 14: Multiple Backdoors

Number of Backdoors	1	2	3
CDP	89.37%	87.00%	86.88%
Average ASR-RelD	99.34%	90.44%	56.40%
Average ASR-SubD	99.87%	99.33%	68.22%

Layer Selection. During the backdoor injection, the attackers need to fine-tune the parameters of DNNs to inject the backdoor. Fine-tuning all parameters in the DNNs will result in a substantial computational cost, contrary to our assumption that the attackers do not have significant computational resources, so fine-tuning some layers with the other layers frozen to inject backdoors is acceptable to attackers. Below, we evaluate whether we can efficiently inject backdoors into Resnet18 for the CIFAR-10 tasks by fine-tuning the layers after the target layer with 900 epochs. The substitute dataset used to fine-tune is the training dataset of CIFAR-100, and the learning rate is 0.0001. Table 13 shows the experimental results of these backdoored Resnet18 DNNs, where the first column indicates the starting point of the fine-tuned parameters (i.e., "Layer1" means the condition we fine-tune all layers after Layer1). Based on the results, we find that only fine-tuning some layers (i.e., layers after Layer2, or Layer3) achieves almost the same performance of backdoor attack and main task as fine-tuning all parameters. Even fine-tuning layers after Layer4, the ASR of the backdoor is 99.71% and the performance degradation of the main task is only 1.01%, which is much better than fine-tuning all layers (i.e., 94.29% for the ASR of backdoor and 1.89% for the main task performance degradation).

Multiple Backdoors. We consider injecting multiple backdoors into the target model, i.e., each trigger corresponding to a unique target label, and evaluate it on Resnet18 trained using CIFAR-10. The results of injecting one, two and three backdoors are shown in Table 14, where CDP and ASR are the average over those of all the backdoors. The results show that our approach can successfully inject up to two backdoors into the target model, with 87.00% CDP and 90.44% ASR-RelD. When injecting more backdoors, i.e., three and more, our approach cannot balance well between the ASR and CDP, either injecting backdoors with low ASR or ruining the CDP.