Stealthy and Flexible Trojan in Deep Learning Framework

Yajie Wang, Kongyang Chen, Yu-an Tan, Shuxin Huang, Wencong Ma, and Yuanzhang Li

Abstract—Deep neural networks (DNNs) are increasingly used as the critical component of applications, bringing high computational costs. Many practitioners host their models on third-party platforms. This practice exposes DNNs to risks: A third party hosting the model may use a malicious deep learning framework to implement a backdoor attack. Our goal is to develop the realistic potential for backdoor attacks in third-party hosting platforms. We introduce a threatening and realistically implementable backdoor attack that is highly stealthy and flexible. We inject trojans by hijacking the built-in functions of the deep learning framework. Existing backdoor attacks rely on poisoning; its trigger is a special pattern superimposed on the input. Unlike existing backdoor attacks, the proposed sequential trigger is a specific sequence of clean image sets. Moreover, our attack is model agnostic and does not require retraining the model or modifying the parameters. Its stealthy is that injecting trojans will not change the model's prediction for a clean image, so existing backdoor defenses cannot detect it. Its flexibility lies in that adversary can remodify the trojan behavior at any time. Extensive experiments on multiple benchmarks with different frameworks demonstrate that our attack achieves a perfect success rate (up to 100%) with minimal damage to model performance. And we can inject multiple trojans which do not affect each other at the same time, trojans hidden in the framework make a universal backdoor attack possible. Analysis and experiments further show that state-of-the-art defenses are ineffective against our attacks. Our work suggests that backdoor attacks in the supply chain need to be urgently explored.

Index Terms—Backdoor attack, supply chain security, deep learning framework, deep neural networks

1 Introduction

Deep neural networks (DNNs) have achieved fantastic performance in a variety of tasks, such as autonomous driving [1], [2], speech recognition [3], [4], and face recognition [5], [6]. However, the widespread use of DNNs also brings with it a whole new set of risks like poisoning attacks, adversarial attacks, and backdoor attacks [7], [8], which could lead to fatal consequences for safety-critical applications that rely on DNNs. Trojan attack (Backdoor attack) is a deliberate attack against DNNs. The trojan attack is a novel attack that aims to manipulate the output of the infected model with pre-mediated inputs. The adversary modifies the training data (or code) during the training phase and retrains the victim model to inject backdoors. In the inference phase, the infected model behaves normally on the original task but activates malicious behavior if the input contains specific triggers. For example, an adversary can fool a self-driving model by adding a particular trigger on the traffic sign, which could cause a fatal accident. Trojan attacks against various DNN models have attracted widespread attention [9], [10], [11], [12], [13], [14] and have become a serious threat to the security of the DNN supply chain.

Existing backdoor attacks work mainly through data poisoning or code poisoning. In the former, the adversary poisons the training data (i.e., inserts samples with triggers) to induce models to learn the connection between triggers and backdoor behaviors [15], [16], [17], [18]. In the latter, the adversary manipulates the pre-trained source code to inject a backdoor into the model [19], [20], [21]. Despite advancements of backdoor attacks, there are still many technical challenges with the existing backdoor attacks. First, backdoor injection depends on the retraining process, the complexity of DNNs leads to a massive overhead of injecting backdoors with retraining; also, the backdoor injection harms models' accuracy. Second, poisoning-based backdoor attacks are not applicable to deployed models whose parameters are frozen. In addition, the existing backdoor defenses focus on poisoning-based attacks. For example, by probing the association between a trigger and a malicious output [22], [23], [24], or artefacts in the victim model [25], [26], [27]. So we need to explore threatening and realistically implementable backdoor attack.

Existing backdoor attacks ignore other vectors available in the DNNs supply chain. We expand the capability of supply-chain backdoor adversaries to make poisoning non-essential in a backdoor pipeline. In this paper, we introduce Stealthy Trojan, a novel approach to backdoor attacks. We utilise the ability of the adversary in the DNNs supply chain

- *Yajie Wang and Yu-an Tan are with the School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing 100081, China. E-mail: {wangyajie19, tan2008}@bit.edu.cn.*
- *Kongyang Chen is with the Institute of Artificial Intelligence and Blockchain, Guangzhou University, Guangzhou 510006, China, and also with Pazhou Lab, Guangzhou 510330, China. E-mail: kychen@gzhu.edu.cn.*
- *Shuxin Huang, Wencong Ma, and Yuanzhang Li are with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China. E-mail: {3120211062, popular}@bit.edu.cn, 1159590493@qq.com.*

Manuscript received 22 Dec. 2021; revised 7 Feb. 2022; accepted 29 Mar. 2022. Date of publication 1 Apr. 2022; date of current version 13 May 2023.
This work was supported in part by the National Natural Science Foundation of China under Grants 61876019, U1936218, 62072037, and 61802383, in part by the Research Project of Guangzhou University under Grant RQ2021007, and in part by the Research Project of Pazhou Lab for Excellent Young Scholars under Grant PZL2021KF0024.
(Corresponding authors: Yuanzhang Li and Kongyang Chen.)
Digital Object Identifier no. 10.1109/TDSC.2022.3164073

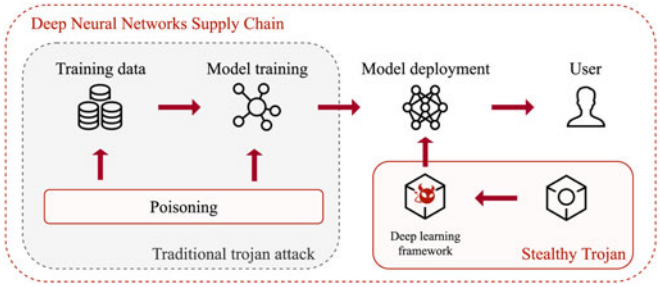

Fig. 1. Different backdoor attacks in the supply chain of DNNs. Traditional backdoor attacks inject Trojans by poisoning during the training phase, and the process is tedious and easily detectable. Our Stealthy Trojan hijacks the built-in functions of deep learning frameworks and injects trojans directly, making backdoor attack threatening and realistically implementable.

to propose an entirely new threat — trojans in the deep learning framework, exposing a huge hidden problem in the supply chain of DNNs. Our Stealthy Trojan injects trojans directly into the deep learning framework by reverse engineering and hijacks build-in functions of deep learning frameworks to introduce malicious behavior (Fig. 1). We first select the appropriate (frequently called by models) function in the framework then inject the trojan by directly perturbing the execution process. The injected component consists of a trigger detector and a register module. The trigger detector is responsible for identifying the presence of the sequential trigger in the model's input. The register hosts the malicious behavior or replaces the actual output with the malicious behavior after the sequential trigger is detected. Finally, the trojan attack is implemented by manipulating the input data stream of the victim model. The innovative sequential trigger makes our attack completely stealthy. We inject a covert decision path into the interaction between the model and the framework and use a specific sequence of clean samples to encode trojan behaviors, enabling the injection or activation of hidden behaviors. In this process, the triggering and execution path of the trojan attack is hidden in the model's invocation of the framework, which makes our attack completely model agnostic and facilitates the adversary to adjust the attack process as needed dynamically. The differences between proposed attack and the traditional trojan attack are compared in detail in Table 1.

Our approach has the following advantages: First, our attack is a model agnostic trojan implantation method, which means there is no need to retrain the victim model or poison any data. Second, the trigger of our attack is exceptionally stealthy, as we launch the attack by using the sequential trigger consisting of a set of clean inputs, which removes any suspicion of malicious inputs. Third, the proposed attack can inject multiple trojans simultaneously and can dynamically modify each trojan's malicious behaviours. Fourth, the injection of trojans does not affect the model's accuracy on the original task, making our attack completely stealthy to model users. Finally, our unique design makes our attack completely undetectable by existing defenses. We will release the source code after the publication of our paper.

In summary, our approach is the first backdoor attack through deep learning frameworks. Stealthy Trojan expands the scenario of backdoor attacks in the supply chain. Our novel approach has a higher attack success rate and stealthiness than previous backdoor attacks. The specific contributions of this paper are as follows.

TABLE 1
Comparison of Different Trojan Attacks

	Traditional trojan attack	Stealthy trojan (ours)
Attacker's access	model source	**framework**
Model format	undeployed	**deployed**
Model change	weights	**no**
Model retrain	required	**no**
Trigger	small pattern	**clean image**
Known defenses	required	**no**
High concealment	no	**yes**

- We propose a universal backdoor attack by injecting trojans into the deep learning framework, called Stealthy Trojan. Our approach is model agnostic and exposes an unknown threat of backdoor attacks in the supply chain.
- We achieve complete stealthiness of the backdoor attack with the sequential trigger and ensure that the trojan injection does not compromise the model's accuracy on the original task.
- Experimental results exhibit that Stealthy Trojan achieves a 100% success rate of the attack and has no impact on the original model. And none of the existing defense methods is effective against Stealthy Trojan.

2 Background and Related Work

2.1 Attacks on Deep Neural Networks

Existing attacks against DNNs have mainly focused on image perturbation. A general type of attack that has been extensively studied is adversarial examples [28], [29], [30], [31]. The goal of these attacks is to make models erroneous. It has been found that there exist imperceptible image perturbations by humans that can cause DNNs to output wrong results at a high probability. Another successful attack strategy is poisoning, in which the attacker has access to the training data of the model. The attacker introduces images with incorrect labels into the training set to degrade the model's performance. However, if the attack causes the model to perform poorly in all cases, it will never be used, so backdoor attack is proposed. The strategy of backdoor attack is to attack a specific class. The poisoned images in backdoor attacks often include malicious additional features or perturbations. These unusual patterns are used as triggers to control when the model goes wrong. For example, Gu *et al.* made the model misjudge a street sign with a trigger by poisoning it [15]. The model inference incorrectly with more than 90% probability when the trigger is present but achieves state-of-the-art accuracy in all other cases.

These attacks all show that the security of DNNs needs to be improved, but we present an entirely new vulnerability. Our attack differs from previous work in the type of access the attacker has to the model. The attacker performs an indirect attack against the model. We launch the attack through the deep learning framework instead of adding perturbations in the training or testing phase.

2.2 Backdoor Attacks

In this subsection, we first introduce two classical approaches to backdoor attacks: BadNet and TrojanNN. We then briefly introduce some recently proposed backdoor attacks. Before the popularity of deep neural networks, there have been works exploring the idea of backdoor attacks on machine learning models, where the main goal of the attacker is to escape network intrusion detection [32] and spam filtering [33], [34].

BadNets [15] is an early backdoor effort that implements backdoor attacks through two phases. The attacker first selects a random subset of the original training data, tags them with pre-designed triggers, and modifies their labels to the predefined target labels. Next, the backdoor is injected into the clean model by fine-tuning the pre-trained model using the poisoned training data. Any input with a trigger is misclassified as the predefined target label.

TrojanNN [16] does not rely on access to and modification of the original training data. The attacker first reverse engineers the pre-trained target model and computes trigger patterns by analyzing the responses of specific neurons in the target model, generating training data. TrojanNN can establish strong associations between triggers and target labels using less training data, but the triggers have an irregular and more significant appearance.

Dynamic backdoor [35] abandons static triggers and proposes a backdoor attack using dynamic triggers. In the dynamic backdoor attack, multiple schemes that rely on generating networks are proposed; they can generate triggers with random patterns and locations to achieve backdoor attacks. They can also generate triggers specific to the target based on the target label.

An important assumption for injecting Trojans is that adversaries have access to training data and implant backdoors by poisoning [18], [36], [37]. However, threat models in which adversaries have access to training data may be impractical in many scenarios, and such malicious behavior is poorly stealthy. So most backdoor attacks have started to focus on how to improve the stealthiness of backdoor attacks [38], [39], [40]. Existing works focus specifically on injecting trojans during the training phase (misleading the training process before the model is deployed). [19] propose blind backdoor attack, which modifies the loss function by poisoning to achieve backdoor injection. We argue existing poisoning-based backdoor attacks have limited impact and cannot be extended to models that cannot be retrained. No work has been proposed to perform backdoor attacks against already deployed models, which is the focus of our research. Our approach is more dangerous and practical. We inject backdoors by hijacking built-in functions in deep learning frameworks without prior knowledge of the victim model.

2.3 Backdoor Defenses

Lots of work explored defenses against backdoor attacks. STRIP [26] proposes to filter poisoned samples by superimposing multiple patterns and analyzing the output. STRIP superimposes the input image with other images and then sends the newly generated image to the model for reasoning and checks the model's output. If the model's output remains constant, then the input image has a backdoor. Neural Cleanse [25] iterates through all labels of the model to rebuild the backdoor trigger, and then anomaly detection technology is applied to identify whether the model is clean. Because the modification required to classify the infected model to the target label is smaller than that of the clean model. DeepInspect [41] uses model inversion and conditional Generative Adversarial Network (cGAN) to reconstruct the substitution training dataset and realizes model backdoor detection under black box conditions. Fine-pruning [42] removes the backdoor by pruning redundant neurons that have a limited effect on the original task of the model. In [43], Zhao *et al.* use pattern connection technology with some benign samples to remove hidden backdoors. Kolouri *et al.* [44] first use universal litmus pattern (ULP) to diagnose whether the model contains backdoors.

Unfortunately, none of the existing defense techniques considers the deployed models, as they all require the use of a large number of test samples or training [45], [46]. Even none work takes into account the backdoors hidden in the framework. All current defenses assume that backdoor attacks are triggered by a particular pattern added to the input, which is not the case with our Stealthy Trojan. Thus our Stealthy Trojan can directly bypass existing defense mechanisms.

3 Stealthy Trojan

3.1 Threat Model

Our approach aims to consider the typical supply chain scenario where a third-party platform hosts pre-trained models for users. Platforms offering hosting services for pre-trained models are common, such as SageMaker in AWS [47] and AI Hub in GCloud [48]. A victim deploys a pre-trained model on the platform provided by an adversary and publishes service to the public. The adversary neither has access to the training data of the target model nor can retrain or modify the model, which means that we do not make any changes to the victim model. At model runtime, users pass data to the model and receive the output, while in the process, the victim model invokes build-in functions of the deep learning framework provided by the adversary, and the adversary has access to the process.

The adversary's goal is to induce the victim model to behave maliciously (e.g., targeted misclassification) when a trigger presents. At the same time, the backdoored model behaves normally on clean input. The adversary manipulates the DNN model by injecting trojans into the deep learning framework. Formally, Let $\mathcal{D} = \{x_n, y_n\}_{n=1}^{N}$ denotes the input data f. y_n denotes the prediction of the model for the n^{th} input x_n, i.e., $y_n = f(x_n)$. Suppose that adversary has injected a trojan in the framework on which model f depends. To launch an attack, the adversary places a sequential trigger Δ in $\mathcal{D}$. The sequential trigger will trigger a malicious behavior that causes the inference result of the model to change to a pre-designed malicious result, i.e., $y_n = y_t$, y_t is the target label that the adversary has arranged in advance. We use f_{trojan} to denote the injected trojan. The inference process of the model can be written as follows:

$$y_n = f_{trojan}(x_n)d(\Delta) + f(x_n)(1 - d(\Delta)), \tag{1}$$

where $d()$ is the trigger detection function. $d(\Delta) = 1$ when the sequential trigger is present, otherwise $d(\Delta) = 0$. When the input data does not contain any trigger, the model performs

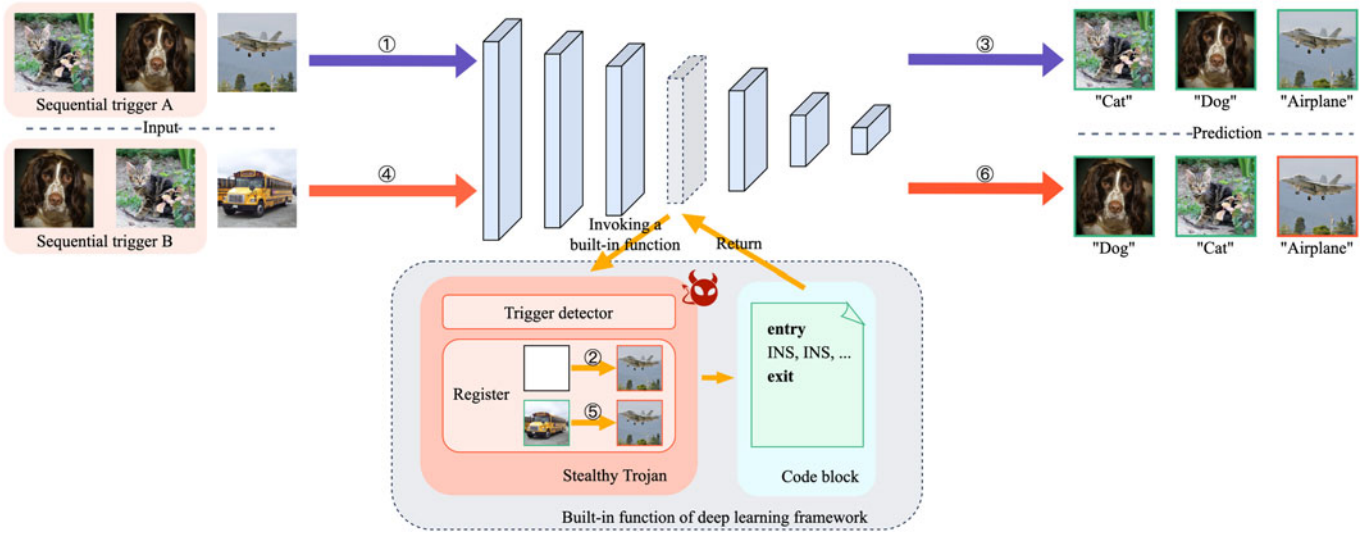

Fig. 2. Illustration of Stealthy Trojan. The red input indicates the sequential triggers sampled from a clean dataset. When a sequential trigger indicating hosting is input, the trigger detector is activated, and the register records the input after the sequential trigger as the malicious behavior (steps 1, 2, 3). When a sequential trigger indicating attack is input, the trigger detector is activated, and the register replaces the output of the sample after the sequential trigger with the malicious behavior therein to perform a backdoor attack (steps 4, 5, 6). Users of the model cannot perceive the whole process.

the original task $f()$. When the sequential trigger appears, $f_{trojan}()$ is activated and dominates the inference result of the model. The goal of the backdoor attack is to inject $f_{trojan}()$ covertly while not affecting the function of $f()$.

3.2 Approach Overview

Our approach considers a more realistic threat scenario in the supply chain of DNNs. An ideal backdoor attack should follow the following principles. First, the trojan should be decoupled from the model, meaning that a same trojan can infect different DNNs. Also, the injection of trojan does not affect the performance of the victim model. Second, multiple trojans can be injected simultaneously, using different triggers to activate. Finally, the trojan should be highly stealthy and cannot be detected by defenses. We found that backdoor injection through data poisoning and code poisoning is not the only feasible way. Existing backdoor attacks focus on the neurons in DNNs, while the contribution of deep learning frameworks is ignored, which are closely related to DNNs. In fact, there is no restriction to prevent the adversary's modification of the deep learning framework in the scenario of supply chain attacks.

In existing backdoor attacks, the adversary manipulates the pre-training process and injects trojans by poisoning. The main difference between our approach and existing backdoor attacks is that our attack aims at deployed pre-trained models for which poisoning is not an option for trojan injection because their weights are frozen. Therefore, we directly hijack build-in functions of the deep learning framework to achieve trojan injection. Our approach is inspired by traditional application backdoor injection. An attacker adds malicious logic (trojan) to an application by purposefully modifying the source code to make the program perform malicious acts when certain conditions are met. Normally, the malicious logic remains hidden until the attacker activates it with a trigger. It is not feasible to implement such an attack in the source code of DNNs because it does not contain any conditional logic. In contrast, deep learning frameworks are built entirely with programming languages, which means that deep learning frameworks can be manipulated and tampered with just like traditional programs. So we chose to launch our attack through the source code of deep learning frameworks. Our attack introduces additional paths in the communication between DNNs and frameworks and exploits them to implement trojan attacks. Specifically, we add backdoor bypasses to the logic of build-in functions and made the malicious behavior activate only in the presence of triggers. This allows us to perform trojan attacks covertly without causing performance degradation of DNNs on the original task.

Our approach explores the process of interaction between a DNN and a deep learning framework. We think that a DNN is a group of the definition of invoking build-in functions of the framework. The data flow between various layers of a DNN is essentially a process of several function interactions and processing data. Our idea is to hijack suitable functions of the framework and add hidden malicious logic to trigger when certain conditions are met. We represent the trojan behavior using a sequence of clean samples, called sequential trigger. We design a trigger detector to determine whether a sequential trigger is present in the input. The trigger detector is predefined based on the sequential trigger, and it is designed to respond sensitively only to the sequential trigger. The sequential trigger can be changed as well as it does not require any knowledge of the victim model. For any victim model deployed on the trojan framework, an adversary can implement a universal trojan attack. An overview of the attack process is presented in Fig. 2, and we refer to this novel backdoor attack as Stealthy Trojan.

The advantage of trojans in deep learning frameworks is that we do not need to make any modifications to the model at all; the malicious logic is hidden in the interaction between the model and the deep learning framework. Our approach is universal and applicable to any pre-trained model, as the adversary can manipulate the deep learning framework on which all the model relies. At the same time,

an attacker can implement any form of attack by adjusting the sequential trigger and malicious behavior. More importantly, existing backdoor defenses all target the DNN itself. A trojan hidden in the framework can resist all current defenses. Future defense mechanisms may hinder our backdoor attack. However, the adversary can adjust the hijacked functions (or trojan behavior) to evade the defense. This is precisely similar to backdoor attacks in traditional applications: manual analysis will eventually expose all the backdoors for static source code. However, backdoor attacks are still significant: a backdoor can cause substantial damage as long as the backdoor is sufficiently stealthy.

3.3 Sequential Trigger

In our design, poisoned samples should not differ too much from clean samples. A prominent poisoned sample will alarm the defense, exposing attack behavior. Therefore, we abandon the traditional way of using conspicuous patches as triggers. We creatively use the timing information of the input as a trigger, called sequential trigger. The timing information has exponentially increasing combinations as the number of input samples increases, which means that our triggers can take an infinite number of forms. Timing-based triggers hide the trigger information in different permutations of the input. This means that we can use completely clean inputs to trigger trojans without any modification. Clean samples are out of the consideration of existing defenses, so our trigger achieves complete imperceptibility. The attacker can define the sequential trigger's length (one to infinity), and we choose a sequence containing two input samples as the sequential trigger. Both of these samples are randomly sampled from clean inputs, such that the trigger will completely escape from existing defenses. We use two sequential triggers, one for injecting malicious behavior and the other for triggering malicious behavior. This also means that the malicious behavior is dynamically changed. It makes defending against our attack almost impossible.

3.4 Trigger Detector

The goal of the trigger detector is to detect the presence of a sequential trigger in the input, and its performance will directly affect the effectiveness of the trojan attack. Our approach considers the scenario where the samples are fed sequentially into the model for inference. The order of the input samples determines whether the sequential trigger exists or not, and the trigger may exist at any position in the queue.

It is crucial to design an effective trigger detector for such a novel backdoor attack. First, the trigger detector should hide in the hijacked function and be sensitive to the function's input, i.e., if the continuous input of the function contains a sequential trigger, the trigger detector alerts and the trojan is activated.

We chose a simple but effective way to build the trigger detector: For the $i+1$ tensors $\{\lambda_1, \lambda_2, \ldots, \lambda_{i+1}\}$ fed into the function, we determine whether the first i tensor matches the preset sequential trigger $\{\mu_1, \mu_2, \ldots, \mu_i\}$. If it matches, the function's output ξ for the tensor λ_{i+1} is replaced by the malicious tensor σ in the register, and the backdoor is activated. If it does not match, the function outputs the result of calculating of the λ_{i+1} tensor normally, and the model usually behaves. In order to make the detector valid for input tensors of different dimensions, we fix the dimension of detection and resize all the input tensors. This design ensures the effectiveness and efficiency of the trigger detector in identifying triggers. See Fig. 3 for details.

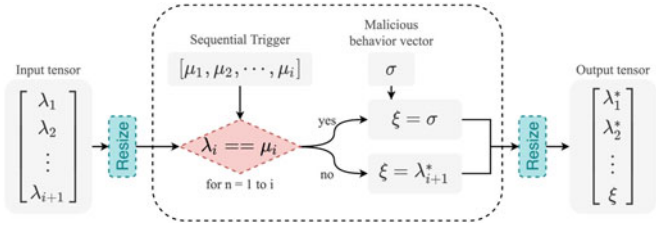

Fig. 3. Stealthy Trojan modifies the tensor information in the built-in functions of the deep learning framework.

3.5 Injecting Trojan

Some work has pointed out that the removal of trojan-related neurons affects normal neurons, which impairs the model's performance on the original task [16]. This suggests that existing poisoning-based backdoor injections are highly coupled to the model. It is a great challenge to decouple the backdoor function from the victim model and build a universal trojan. Also, injecting trojans into deep learning frameworks is challenging due to the following reasons: 1) Backdoor attacks are hidden in the process of model invokes to framework functions, and selecting functions that DNNs do not invoke for hijacking leads to the attack's failure. 2) The process of calling the same function is different for different DNNs and different layers of the same DNN, i.e., the scale of the input tensor is different.

The data-driven nature of DNNs dictates that they contain only mathematical operations such as summation and multiplication and do not contain explicit conditional logic as traditional programs do. The complex implicit conditional logic that DNNs can learn (using features for recognition) cannot be expressed explicitly due to its lack of transparency. However, the deep learning framework consists entirely of logic programs. Therefore, injecting explicit malicious logic into the DNN-dependent framework is the perfect way to implement backdoor attacks. And a backdoor injection into the framework will not affect the model at all. Moreover, the feature that backdoors are activated only in the presence of triggers is well suited for using explicit logical expressions, as is the case with traditional backdoor attacks in software.

Next, we describe in detail how to hijack functions in a deep learning framework and inject trojan. A DNN model can be represented as a series of functions invoking from the framework. Each layer represents one or more function calls, and the data is propagated between the model and the framework via tensors. The passing of the tensor is the basis of our trojan. For any model, the tensor passed by a model layer can be obtained by hijacking the function invoked by that layer. Our goal is to implant a bypass between the input and output of the function. The implanted bypass mainly consists of the following components: 1) Trigger detector. When several tensors are sequentially fed into the function, the detector is invoked to identify the presence of a sequential trigger and activate the corresponding trojan behavior. 2) Register. Depending on the different sequential triggers, the malicious inputs are stored, or the clean output is

replaced with the malicious inputs stored. The output of the hijacked function is chosen between the actual output and the malicious output based on the presence or absence of the sequential trigger. We use if-else statements here to implement this logic. See Fig. 3 and Algorithm 1 for details.

Algorithm 1. Example Hijacked Function

Input: An incoming tensor $\{\lambda_1, \lambda_2, \ldots, \lambda_{i+1}\}$; The length of the sequential trigger i
Parameter: Pre-defined sequential trigger $\{\mu_1, \mu_2, \ldots, \mu_i\}$; The malicious behavior vector σ
Output: The outcoming tensor ξ

```
Resize{λ_1, λ_2, ..., λ_i}.
for n = 1 to i do
  if λ_i == μ_i then
    ξ ← σ
  else
    ξ ← λ_{i+1}
    break
  end if
end for
return ξ
```

The first step of our injection is to select the function to exploit in the deep learning framework. DNNs do not invoke all functions, so we choose standard functions such as convolution and pooling for hijacking. Once we have a target, the second step is to capture the tensor of the sequential trigger input to that function. We fed the sequential trigger into the DNN and record the tensor in the hijacked function into the trigger detector. Fortunately, all deep learning frameworks provide resize operators that convert a tensor of arbitrary size to a given size. This provides us with the possibility to hijack arbitrary calls to the same function. In the end, we define different trojan behaviors based on various sequential triggers.

We can inject trojan bypasses into any build-in functions (as long as a DNN invokes the function). As expected, an attacker can seamlessly attack multiple DNNs with the same trojan. We can inject multiple independent trojans containing different malicious behaviors simultaneously, enabling a multi-target backdoor attack capability. Theoretically, Stealthy Trojan has the ability to inject an unlimited number of trojans at the same time. In addition, the trojan hidden in the framework will spoof existing defense methods.

4 Attack Evaluation

4.1 Evaluation Settings

We implement our backdoor attack with Python 3.8, PyTorch 1.9.0 [49], and TensorFlow 2.5.0 [50]. We use reverse engineering to hijack and modify built-in functions of deep learning frameworks to inject trojans then attack the models running on the framework. For the adversary, no model details are known, so our attack is carried out in a black box scenario.

The three basic stages of DNNs inference are preprocessing, forward propagation, and decision making. We chose several common functions for hijacking in each of the three stages, we chose 14 common functions in PyTorch, and we chose 9 common functions in TensorFlow. The locations where the functions were invoked we chose randomly. See Fig. 4 for the specific function names.

Datasets and Models. We evaluate our approach with four computer vision and one natural language processing benchmarks: MNIST [51], CIFAR-10 [52], ImageNet [53], LFW [54], and IMDb [55] datasets. For the large ImageNet dataset, we use a subset of it for testing to save time [56]. In addition, we also experiment on an inappropriate content detection benchmark. We used LeNet [57], ResNet-50 [58], DenseNet-121 [59], T2T-ViT [60], MobileNetV2 [61], ArcFace [62], and BERT [63] as target models. To test the effectiveness and utility of Stealthy Trojan, we simulate the scenario of model hosting in a real supply chain: All target models are derived from the Internet, and we use pre-trained parameters and weights directly. The target model runs on a malicious deep learning framework, and the adversary attacks by fine-tuning the input sequence of the model.

Evaluation Metrics. We measure the proposed backdoor attack in two main aspects: whether the backdoor behavior can be triggered correctly and whether the victim model remains silent for clean samples. We evaluate the *Attack Success Rate (ASR)* and the *Clean Sample Accuracy (CSA)* of the attacked model. We measure ASR by calculating the percentage of "poisoned" samples that successfully launch a correct backdoor behavior. CSA represents the accuracy of the victim model evaluated on clean test data. Specifically, the test samples for evaluating ASR and CSA are generated by modifying the order of some samples in the original dataset. In addition, we report the *Benign Accuracy (BA)* of the victim model on the original dataset as a reference. BA indicates the accuracy of the original dataset predicted on the clean framework.

4.2 Stealthy Trojan Evaluation

Table 2 demonstrates the performance of Stealthy Trojan on six benchmark tasks. Extensive experimental results show that our approach achieves a surprising attack success rate (up to 100% and down to 86%). The maximum performance degradation of the model caused by the attack is 1.25%. The attack success rates of Stealthy Trojan by hijacking different functions are shown in Fig. 4. It can be found that the attack success rate is awe-inspiring regardless of the attack using any built-in function. This means that we can launch a trojan attack as long as the model invokes our hijacked function. These phenomena raise more concerns, as our results imply that an adversary can execute a completely covert backdoor attack through the deep learning framework. An adversary can implement an attack on any model running on the framework without accessing its training data or parameters.

We also show the performance of Stealthy Trojan on different models, as detailed in Table 3. Our attacks achieve high attack success rates (up to 100% and down to 86%) on multiple models running on two different deep learning frameworks. The maximum model performance degradation caused by our attacks is 1.7%, and some experiments even yield a trace improvement of model performance. The reason for the model performance improvement is the sequential trigger we added to the input, where the samples in the sequential trigger are sampled from the clean test set. If the model correctly identifies the sampled samples, the model performance will show an improvement. This is a dangerous result because the adversary can gain the victim's trust by improving model performance (when the victim hosts the model to the adversary).

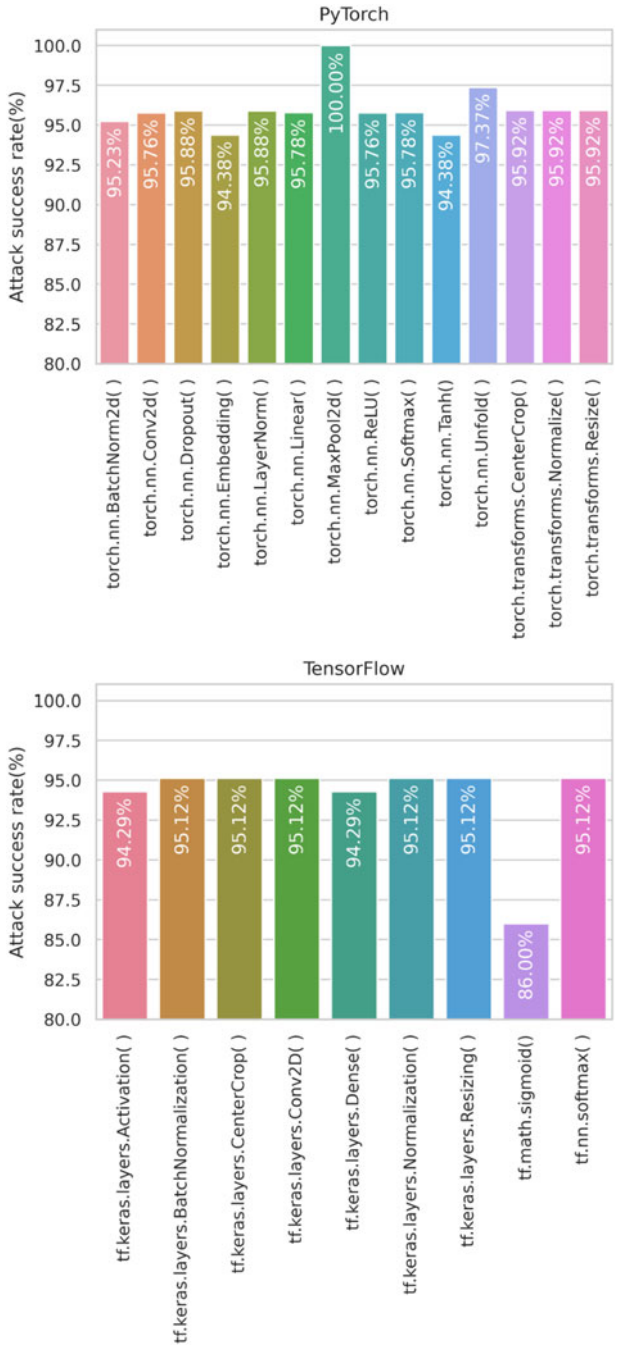

Fig. 4. Performance of Stealthy Trojan on hijacking different PyTorch and TensorFlow built-in functions. The values in the figure show the average ASR of hijacking the same function against up to 7 models.

Our attack is much more concise and efficient than traditional backdoor attacks that exploit poisoning. In our experiments, we use less than 1% of the samples to constitute the sequential trigger, much smaller than the more than 5% of data needed for traditional backdoor injection. Moreover, all the failures of our attacks are caused by the randomly sampled malicious samples that the model cannot correctly identify. In other words, our attack success rate is astonishingly 100% when the malicious samples are guaranteed to be recognized by the model for a specific classification. Moreover, unlike traditional backdoor attacks that implant immutable malicious behavior into the model, the malicious behavior in our attack is dynamic. An adversary can always use a specific sequential trigger to activate the register's hosting mechanism and modify the backdoor behavior. One can imagine an attacker using a sequential trigger to continuously alternate the malicious behavior and trigger the backdoor. In such cases, the attacker can completely circumvent the existing defenses.

TABLE 3
Performance of Stealthy Trojan on Different Models

Models	Pytorch			TrnsorFlow		
	ASR	CSA	BA	ASR	CSA	BA
ArcFace	**100.0**	98.71	98.68	**100.0**	97.50	98.70
BERT	94.38	93.63	93.33	86.00	86.70	86.10
DenseNet-121	95.85	92.49	92.18	90.50	92.35	92.90
LeNet	**100.0**	**99.24**	99.17	99.00	**97.60**	98.50
MobileNetV2	96.00	96.64	96.60	**100.0**	96.70	98.00
ResNet-50	93.53	95.00	94.71	94.05	91.37	91.25
T2T-ViT	97.37	97.48	97.40	95.00	93.50	95.20

The values in the table are the average results of separate attacks on the same model with up to 14 hijacked built-in functions of frameworks.

4.3 Multiple Trojans Evaluation

We investigate the multi-trojan attack capacity. The results shown in Table 4 indicate that Stealthy Trojan can execute multiple trojan attacks simultaneously with a high attack success rate. We randomly injected independent trojans into all output categories of the victim model. As far as we know, this is the first method to achieve a multiple trojan attack. As shown in Table 4, the attack success rate does not drop significantly when we increase the number of injected trojans. Regardless of whether these trojans are in the same function or different functions. The injections of multiple trojans are entirely independent, so they do not impair each other's attack performance. Overall, Stealthy Trojan can achieve multi-trojan attacks with a maximum attack success rate of 93.79% without degrading the accuracy of the victim model on the original task. This significantly improves the ability of adversaries in the supply chain to implement backdoor attacks.

Evading Existing Defenses. Existing defenses focus on empirical defenses against poisoning-based backdoor attacks, using knowledge of existing attacks. We divide backdoor defenses into two categories: sample filtering

TABLE 2
Performance of Stealthy Trojan on Different Benchmark Tasks

Frameworks	MNIST			CIFAR-10			ImageNet			LFW			IMDb			Inappropriate content		
	ASR	CSA	BA	ASR	CSA	BA	ASR	CSA	BA	ASR	CSA	BA	ASR	CSA	BA	ASR	CSA	BA
Pytorch	**100.0**	**99.24**	99.17	**97.98**	**95.68**	95.45	**96.49**	92.83	92.54	98.00	**97.67**	97.64	**94.38**	**93.63**	93.33	88.89	**96.31**	96.04
TensorFlow	99.00	97.60	98.50	91.00	91.20	91.70	93.33	**93.73**	94.70	**100.0**	97.10	98.35	86.00	86.70	86.10	**95.10**	90.05	89.25

The values in the table are the average results of separate attacks using cross combinations of 7 models and up to 14 build-in functions of frameworks.

TABLE 4
Performance of Multiple Trojan

	Pytorch			TensorFlow		
	ASR	CSA	BA	ASR	CSA	BA
Exp.1	93.10	93.99	93.54	92.30	**92.91**	93.50
Exp.2	93.68	94.26	93.54	**92.56**	92.34	93.50
Exp.3	92.32	93.92	93.54	92.40	92.87	93.50
Exp.4	93.20	**94.50**	93.54	91.80	92.45	93.50
Exp.5	**93.79**	94.18	93.54	92.40	91.78	93.50

Exp.1 denotes implant 2 trojans in 1 function; Exp.2 denotes implant 5 trojans in 1 function; Exp.3 denotes implant 2 trojans in 2 different functions; Exp.4 denotes implant 3 trojans in 3 different functions; Exp.5 denotes implant 5 trojans in 5 different functions.

based defenses and model diagnosis based defenses. We demonstrate that Stealthy Trojan can completely escape from existing backdoor detection and defense mechanisms through further analysis and experiments.

Evading Sample Filtering Based Defenses. Sample filtering based defenses introduce a censorship mechanism in the pre-training phase or inference phase, aiming to distinguish malicious samples from benign ones. Benign samples can pass directly. Malicious samples are rejected or modified to remove triggers. Stealthy Trojan can naturally escape such defenses because the defenses are sensitive only to outliers caused by poison, but we do not need any poisoned samples during the attack.

Evading Model Diagnosis Based Defenses. Model diagnosis based defenses are designed to suppress or remove backdoors that are hidden in the model. The defender performs a diagnostic of the model to confirm whether the model contains a backdoor. Infected models are denied deployment, or the backdoor is removed through reconstruction. Such defenses expose backdoors by feeding specific patterns into the model and analyzing its output. Stealthy Trojan can still naturally evade such defenses because its detection is limited to the model, while our Trojan hides in the framework. However, we are still concerned that attacks on the framework will lead to other outliers detected by the defense mechanism. Therefore, we verified our method with two SOTA defenses [43], [44], proving our concerns are unnecessary.

In our experiments, we inject 1 or 5 Trojans into the framework and then observe the defence's performance. Our experiments use the same configuration as the authors and the same metrics. Our experiments are conducted on the CIFAR-10 dataset and VGG networks.

TABLE 5
The Train Loss and Error Rate of the Defense of Zhao *et al.* [43]

Backdoor attacks	Train loss	Error rate
Clean	2.34	73.31%
Ours(1 trojan)	2.34	73.64%
Ours(5 trojans)	2.35	73.64%

We report the average results of five experiments.

For the defense in [43], we report the train loss and error rate before and after the attack, respectively. As can be observed from Table 5, the values of the two metrics do not change significantly before and after the attack. Our attack successfully escapes the defense.

For the defense in [44], we plot ROC curves for detection baseline, random input, and ULP in Fig. 5. We add Stealthy Trojan to all experiments and report the performance before and after the attack. It can be seen that the detection performance of ULP for traditional backdoors does not change significantly, still achieving an area under the curve (AUC) of up to nearly 1. However, UAP does not detect our backdoor attack at all.

Our results expose the limitations of existing defenses: existing defenses focus only on untrusted data or untrusted models. Trojans hidden in deep learning frameworks can easily evade existing defenses.

5 Discussion

This section discusses potential measures to prevent or detect proposed attacks.

Developers are responsible for building models and deploying them, and practical measures they can take include: 1) ensuring that the model's hosting platform is from a trusted provider. 2) performing control flow integrity checks on the model at runtime to ensure that the model's operational process is secure. 3) using a secure environment such as a private cloud to deploy the model and provide services.

In addition, framework providers should consider how to detect malicious behavior hidden in deep learning frameworks and provide better protection mechanisms: 1) Framework obfuscation. Similar to code obfuscation techniques, it may be interesting to obfuscate frameworks to make them more difficult to hijack. 2) Check file signatures at runtime to ensure that the framework being used has not been modified.

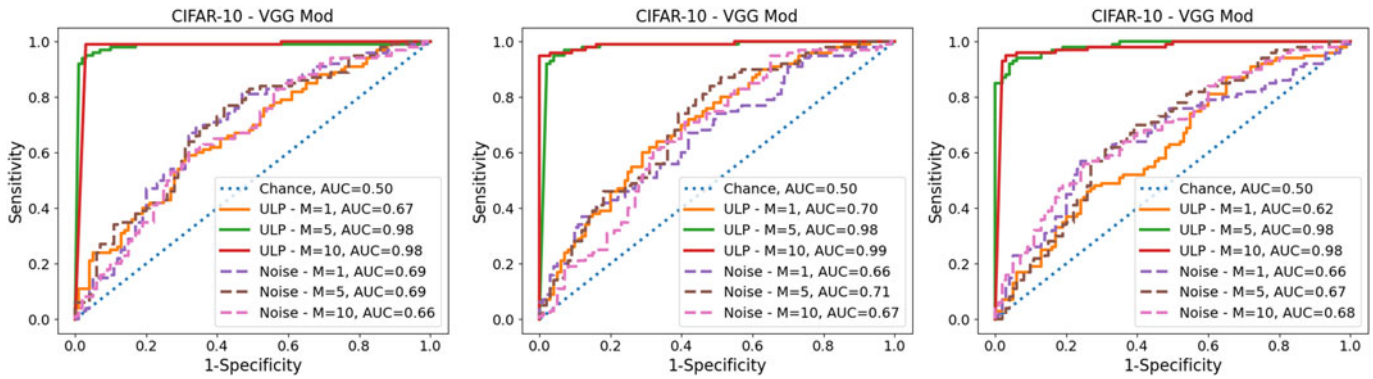

Fig. 5. ROC-curves for detection of models with traditional backdoor attacks for baseline, random input images, and ULPs [44]. We report the performance on clean framework (left), framework with 1 trojan (middle), and framework with 5 trojans (right).

3) Build validation mechanisms into the framework. It would be helpful if deep learning frameworks could provide built-in APIs to verify their integrity and security.

6 Conclusion and Future Work

In this work, we propose a novel backdoor attack mechanism called Stealthy Trojan. It implements a covert backdoor attack by hijacking built-in functions of the deep learning framework. Our attack is completely black-box: it does not rely on any knowledge of the target model. The adversary can even implement a trojan attack with only clean data. Various experiments demonstrate that Stealthy Trojan is both secret and a real threat. Compared to previous researches, our work exposes entirely new threats in the DNNs supply chain.

We hope that our work will stimulate more research on the risks of backdoor attacks in the DNNs supply chain, and the proposed backdoor injection approach may open up a new research direction. Although our work focuses on backdoor attacks, defense is more valuable. Trojan detection and defense in the DNNs supply chain is a challenging topic that we will continue to explore in the future.

References

[1] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "DeepDriving: Learning affordance for direct perception in autonomous driving," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 2722–2730.

[2] Y. Tian, K. Pei, S. Jana, and B. Ray, "DeepTest: Automated testing of deep-neural-network-driven autonomous cars," in *Proc. 40th Int. Conf. Softw. Eng.*, 2018, pp. 303–314.

[3] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2013, pp. 6645–6649.

[4] O. Abdel-Hamid , A.-R. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 22, no. 10, pp. 1533–1545, Oct. 2014.

[5] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE Trans. Neural Netw.*, vol. 8, no. 1, pp. 98–113, Jan. 1997.

[6] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," 2021. Accessed: Jul. 21, 2015. [Online]. Available: https://www.robots.ox.ac.uk/~vgg/publications/2015/Parkhi15/parkhi15.pdf

[7] N. Akhtar and A. Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," *IEEE Access*, vol. 6, pp. 14 410–14 430, 2018.

[8] Y. Li, B. Wu, Y. Jiang, Z. Li, and S.-T. Xia, "Backdoor learning: A survey," 2020, *arXiv:2007.08745*.

[9] C. Li *et al.*, "Backdoor attack on machine learning based android malware detectors," *IEEE Trans. Dependable Secure Comput.*, to be published, doi: 10.1109/TDSC.2021.3094824.

[10] X. Liu, H. Li, G. Xu, Z. Chen, X. Huang, and R. Lu, "Privacy-enhanced federated learning against poisoning adversaries," *IEEE Trans. Inf. Forensics Secur.*, vol. 16, pp. 4574–4588, 2021.

[11] B. Hou *et al.*, "Mitigating the backdoor attack by federated filters for industrial IoT applications," *IEEE Trans. Ind. Informat.*, vol. 18, no. 5, pp. 3562–3571, May 2022.

[12] S. Wang, S. Nepal, C. Rudolph, M. Grobler, S. Chen, and T. Chen, "Backdoor attacks against transfer learning with pre-trained deep learning models," *IEEE Trans. Serv. Comput.*, to be published, doi: 10.1109/TSC.2020.3000900.

[13] Z. Yan, J. Wu, G. Li, S. Li, and M. Guizani, "Deep neural backdoor in semi-supervised learning: Threats and countermeasures," *IEEE Trans. Inf. Forensics Secur.*, vol. 16, pp. 4827–4842, 2021.

[14] M. Fan, Z. Si, X. Xie, Y. Liu, and T. Liu, "Text backdoor detection using an interpretable RNN abstract model," *IEEE Trans. Inf. Forensics Secur.*, vol. 16, pp. 4117–4132, 2021.

[15] T. Gu, B. Dolan-Gavitt, and S. Garg, "BadNets: Identifying vulnerabilities in the machine learning model supply chain," 2017, *arXiv:1708.06733*.

[16] Y. Liu *et al.* "Trojaning attack on neural networks," Dept. Compu. Sci., 2017, Tech. Rep., Paper 1781. [Online]. Available: https://docs.lib.purdue.edu/cstech/1781

[17] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," 2017, *arXiv:1712.05526*.

[18] A. Saha, A. Subramanya, and H. Pirsiavash, "Hidden trigger backdoor attacks," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 11 957–11 965.

[19] E. Bagdasaryan and V. Shmatikov, "Blind backdoors in deep learning models," 2020, *arXiv:2005.03823*.

[20] S. Garg, A. Kumar, V. Goel, and Y. Liang, "Can adversarial weight perturbations inject neural backdoors," in *Proc. 29th ACM Int. Conf. Inf. Knowl. Manage.*, 2020, pp. 2029–2032.

[21] A. S. Rakin, Z. He, and D. Fan, "TBT: Targeted neural network attack with bit trojan," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 13 195–13 204.

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

[23] S. Hong, V. Chandrasekaran, Y. Kaya, T. Dumitraș, and N. Papernot, "On the effectiveness of mitigating data poisoning attacks with gradient shaping," 2020, *arXiv:2002.11497*.

[24] E. Borgnia *et al.*, "Strong data augmentation sanitizes poisoning and backdoor attacks without an accuracy tradeoff," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2021, pp. 3855–3859.

[25] B. Wang *et al.*, "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," in *Proc. IEEE Symp. Secur. Privacy*, 2019, pp. 707–723.

[26] Y. Gao, C. Xu, D. Wang, S. Chen, D. C. Ranasinghe, and S. Nepal, "STRIP: A defence against trojan attacks on deep neural networks," in *Proc. 35th Annu. Comput. Secur. Appl. Conf.*, 2019, pp. 113–125.

[27] Y. Zeng, W. Park, Z. M. Mao, and R. Jia, "Rethinking the backdoor attacks' triggers: A frequency perspective," 2021, *arXiv:2104.03413*.

[28] C. Szegedy *et al.*, "Intriguing properties of neural networks," 2013, *arXiv:1312.6199*.

[29] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks," 2016, *arXiv:1611.02770*.

[30] K. Eykholt *et al.*, "Robust physical-world attacks on deep learning visual classification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 1625–1634.

[31] X. Xu, X. Chen, C. Liu, A. Rohrbach, T. Darrell, and D. Song, "Fooling vision and language models despite localization and attention mechanism," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4951–4961.

[32] J. Newsome, B. Karp, and D. Song, "Paragraph: Thwarting signature learning by training maliciously," in *Proc. Int. Workshop Recent Adv. Intrusion Detection*, 2006, pp. 81–105.

[33] N. Dalvi, P. Domingos, S. Sanghai, and D. Verma, "Adversarial classification," in *Proc. 10th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2004, pp. 99–108.

[34] D. Lowd and C. Meek, "Adversarial learning," in *Proc. 11th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2005, pp. 641–647.

[35] A. Salem, R. Wen, M. Backes, S. Ma, and Y. Zhang, "Dynamic backdoor attacks against machine learning models," 2020, *arXiv:2003.03675*.

[36] Y. Liu, Y. Xie, and A. Srivastava,"Neural trojans," in *Proc. IEEE Int. Conf. Comput. Des.*, 2017, pp. 45–48.

[37] Y. Liu, X. Ma, J. Bailey, and F. Lu, "Reflection backdoor: A natural backdoor attack on deep neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 182–199.

[38] S. Li, M. Xue, B. Z. H. Zhao, H. Zhu, and X. Zhang, "Invisible backdoor attacks on deep neural networks via steganography and regularization," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 5, pp. 2088–2105, Sep./Oct. 2021.

[39] T. Liu, W. Wen, and Y. Jin, "SIN: Stealth infection on neural network—A low-cost agile neural trojan attack methodology," in *Proc. IEEE Int. Symp. Hardware Oriented Secur. Trust*, 2018, pp. 227–230.

[40] M. Xue, C. He, J. Wang, and W. Liu, "One-to-N & N-to-One: Two advanced backdoor attacks against deep learning models," *IEEE Trans. Dependable Secure Comput.*, to be published, doi: 10.1109/TDSC.2020.3028448.

[41] H. Chen, C. Fu, J. Zhao, and F. Koushanfar, "DeepInspect: A black-box trojan detection and mitigation framework for deep neural networks," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, 2019, pp. 4658–4664.

[42] K. Liu, B. Dolan-Gavitt , and S. Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks," in *Proc. Int. Symp. Res. Attacks Intrusions Defenses*, 2018, pp. 273–294.
[43] P. Zhao, P.-Y. Chen, P. Das, K. N. Ramamurthy, and X. Lin, "Bridging mode connectivity in loss landscapes and adversarial robustness," 2020, *arXiv:2005.00060*.
[44] S. Kolouri, A. Saha, H. Pirsiavash, and H. Hoffmann, "Universal litmus patterns: Revealing backdoor attacks in CNNs," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 298–307.
[45] W. Jiang, X. Wen, J. Zhan, X. Wang, and Z. Song, "Interpretability-guided defense against backdoor attacks to deep neural networks," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, to be published, doi: 10.1109/TCAD.2021.3111123.
[46] Z. Xiang, D. J. Miller, and G. Kesidis, "Detection of backdoors in trained classifiers without access to the training set," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 3, pp. 1177–1191, Mar. 2022.
[47] I. Amazon Web Services, "SageMaker," 2021. Accessed: Sep. 01, 2021. [Online]. Available: https://aws.amazon.com/sagemaker/
[48] Google, "AI hub," 2021. Accessed: Sep. 01, 2021. [Online]. Available: https://aihub.cloud.google.com
[49] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 8026–8037.
[50] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Oper. Syst. Des. Implementation*, 2016, pp. 265–283.
[51] Y. LeCun , "The MNIST database of handwritten digits," 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist/
[52] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," p. 7, 2009. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.222.9220&rep=rep1&type=pdf
[53] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei , "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.
[54] G. B. Huang, M. Mattar, T. Berg, and E. Learned-Miller , "Labeled faces in the wild: A database forstudying face recognition in unconstrained environments," in *Proc. Workshop Faces 'Real-Life' Images: Detection Alignment Recognit.*, E. Learned-Miller, A. Ferencz, and F. Jurie, Eds., Marseille, France, 2008.
[55] A. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proc. 49th Annu. Meeting Assoc. Comput. Linguistics: Hum. Lang. Technol.*, 2011, pp. 142–150.
[56] J. Howard *et al.* "Imagenette," 2021. Accessed: Sep. 01, 2021. [Online]. Available: https://github.com/fastai/imagenette
[57] Y. LeCun *et al.*, "LeNet-5, convolutional neural networks," 2015. [Online]. Available: http://yann.lecun.com/exdb/lenet
[58] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[59] G. Huang, Z. Liu, L. Van Der Maaten , and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 2261–2269.
[60] L. Yuan *et al.*, "Tokens-to-token ViT: Training vision transformers from scratch on ImageNet," 2021, *arXiv:2101.11986*.
[61] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520.
[62] J. Deng, J. Guo, N. Xue, and S. Zafeiriou, "ArcFace: Additive angular margin loss for deep face recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 4685–4694.
[63] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.

Yajie Wang is currently working toward the PhD degree in the School of Computer Science, Beijing Institute of Technology, Beijing, China. His main research interests include the robustness and vulnerability of artificial intelligence, cyberspace security, etc.

Kongyang Chen received the PhD degree in computer science from the University of Chinese Academy of Sciences, Beijing, China. He is currently an associate professor with the Institutes of Artificial Intelligence and Blockchain, Guangzhou University, China. Before that, he was a postdoctoral fellow with the Department of Computing, Hong Kong Polytechnic University, Hong Kong, China. From 2014 to 2018, he was an assistant professor with the Shenzhen Institutes of Advanced Technology (SIAT), Chinese Academy of Sciences, Shenzhen, China. His main research interests include artificial intelligence, privacy computing, edge computing, as well as distributed systems such as Internet of Things (IoT) and blockchain.

Yu-an Tan received the BS, MS, and PhD degrees in software and theory of computers from the Beijing Institute of Technology, Beijing, China, in 1991, 1994, and 2004, respectively. Since 2010, he has been a professor and PhD supervisor with the Beijing Institute of Technology. He is a senior member of the China Computer Federation. His main research interests include information security, network storage, and embedded systems.

Shuxin Huang received the graduate degree from the School of Computer Science, Beijing Institute of Technology, Beijing, China. Her main research interests include the backdoor attacks and defences.

Wencong Ma received the graduate degree from the School of Computer Science, Beijing Institute of Technology, Beijing, China. Her main research interests include the backdoor attacks and defences.

Yuanzhang Li received the BS, MS, and PhD degrees in software and theory of computer from the Beijing Institute of Technology, Beijing, China, in 2001, 2004, and 2015, respectively. He is currently an associate professor with the Beijing Institute of Technology. His main research interests focus on mobile computing and information security.