Knowledge-Based Systems 304 (2024) 112565

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Projan: A probabilistic trojan attack on deep neural networks

Mehrin Saremi [a,*], Mohammad Khalooei [b], Razieh Rastgoo [c], Mohammad Sabokrou [d,e]

[a] Semnan University, Farzanegan Campus, Semnan, 35131-19111, Iran
[b] Amirkabir University of Technology, Department of Computer Engineering, Tehran, Iran
[c] Faculty of Electrical and Computer Engineering, Semnan University, Semnan, 35131-19111, Iran
[d] Institute for Research in Fundamental Sciences, Tehran, Iran
[e] Okinawa Institute of Science and Technology, Okinawa, Japan

ARTICLE INFO

Dataset link: https://github.com/programehr/Projan

Keywords:
Deep learning
AI security
Trojan attack
Backdoor attack
Probabilistic trojan attack

ABSTRACT

Deep neural networks have gained popularity due to their outstanding performance across various domains. However, because of their lack of explainability, they are vulnerable to some kinds of threats including the trojan or backdoor attack, in which an adversary can train the model to respond to a crafted peculiar input pattern (also called trigger) according to their will.

Several trojan attack and defense methods have been proposed in the literature. Many of the defense methods are based on the assumption that the possibly existing trigger must be able to affect the model's behavior, making it output a certain class label for all inputs. In this work, we propose an alternative attack method that violates this assumption. Instead of a single trigger that works on all inputs, a few triggers are generated that will affect only some of the inputs. At attack time, the adversary will need to try more than one trigger to succeed, which might be possible in some real-world situations.

Our experiments on MNIST and CIFAR-10 datasets show that such an attack can be implemented successfully, reaching an attack success rate similar to baseline methods called BadNet and N-to-One. We also tested wide range of defense methods and verified that in general, this kind of backdoor is more difficult for defense algorithms to detect. The code is available at https://github.com/programehr/Projan.

1. Introduction

In recent years, deep learning has emerged as a widely utilized machine learning technique. This approach involves adjusting numerous parameters within a complex function using gradient descent. The intricacy of this function renders deep models somewhat opaque, turning them into black boxes where the reasoning behind their decisions becomes challenging to interpret. This lack of explainability exposes deep networks to potential vulnerabilities from various attack vectors.

As a result, given the extensive applications of deep learning-based models [1], their reliability has become a major concern, especially in mission-critical systems, such as computer vision, disease diagnosis, financial fraud detection, defense against malware and cyber-attacks, access control, surveillance, and more [2,3].

Since it is common for deep learning models to be trained and provided as pre-trained models by third parties, opportunities arise for adversaries to manipulate data and/or models. In one such attack type, known as a trojan attack, adversaries insert "backdoors" or "trojans" into the models. This involves training or fine-tuning the model to produce the desired output when a specific pattern (referred to as a trigger) is injected into the input data. The model, however, should behave normally in the absence of the trigger to minimize the risk of detection.

The trojan attack is readily realizable in the physical world, especially in vision systems. It is simple, highly effective, robust, and easy to realize by e.g., placing a trigger on an object within a visual scene. Another form of attack is the adversarial attack, where the attacker has limited control over converting the physical scene into an effective adversarial digital input, often involving small perturbations such as altering a single pixel in an image.

Detection of such triggers is challenging. The most important challenge is regarding trigger identification. The arbitrary nature of triggers in terms of shape, pattern, location, and size makes it impossible for defenders to predict these attributes accurately. Furthermore, the trojaned samples are inserted into the training data during the model training or tuning which are most likely not presented to the user by the attacker. The challenge lies in the difficulty of validating the anomalous

* Corresponding author.
E-mail addresses: m.saremi@semnan.ac.ir (M. Saremi), khalooei@aut.ac.ir (M. Khalooei), rrastgoo@semnan.ac.ir (R. Rastgoo), sabokro@ipm.ir (M. Sabokrou).

https://doi.org/10.1016/j.knosys.2024.112565
Received 18 March 2024; Received in revised form 14 September 2024; Accepted 27 September 2024
Available online 1 October 2024

training data to discern the malicious behavior inherent in the trojaned data.

The attacker usually seeks for a simple and universal trigger. A universal trigger is one that tricks the network when inserted into any input. The chosen trigger pattern needs to be simple, so that it is easy to apply and also does not cause too much deviation from the distribution of natural (benign) inputs, to avoid detection. As a result, the network learns a simple pattern that consistently (mis-)leads it to a specific class (the target). This phenomenon leaves some artifacts in the model which defense methods attempt to exploit for trojan detection.

For instance, the authors of Neural Cleanse hypothesize that it is possible to trick a trojaned model into selecting the target class by applying a small modification to any input, and propose to find such a trigger through optimization [4]. ABS, on the other hand, hypothesizes that a poisoned model contains some “compromised neurons” that, when stimulated, output the target class, and such neurons have little interference with other ones [5].

However, in certain scenarios, running an attack multiple times until it succeeds may be plausible, akin to brute force attacks in the cybersecurity domain. A malicious actor attempting to breach a face recognition-controlled gate may first wear special glasses, then close one eye, and finally wear a headband before the gate opens.

In this paper, we propose a probabilistic attack method called *“Projan”*, for probabilistic trojan. The idea is to overturn the assumption that “a single trigger works with all inputs”. Instead, a trigger only works with some inputs, and a set of triggers collectively work with (ideally) all inputs. Unlike other attacks, we intentionally keep the attack success rate of each trigger low to fly under the radar. Attacking, then, consists of testing the triggers sequentially, until one of them proves successful.

To the best of our knowledge, only one work has proposed a probabilistic neural network attack [6]. This attack involves dropping out some target neurons during training and presenting the model with the target label. In the test phase, the attacker queries the model multiple times. Assuming random dropout is active during testing, the target neurons are likely to be dropped after some trials, directing the network to the attacker’s desired class.

However, this approach has shortcomings. The attacker relies on dropout being active in the deployed model, and the method requires a substantial number of queries to achieve success. For instance, the mentioned paper notes the need for 500 queries to reach an Attack Success Rate (ASR) of 46% on CIFAR-10. Our proposed method, in contrast, achieves a high ASR with only a few trials and does not necessitate the use of dropout during testing.

Another work that bears similarities with our proposed method is the N-to-One attack [7] in that it uses more than one trigger. While training, it poisons a small fraction of training data with each trigger and presents it to the model with the target label. While testing, as the number of poisoned examples is small, no single trigger will cause the model to return the target label. However, when all of the triggers are added to the input at the same time, it leads the model to the target class. Even though there are n triggers, there is still a single trigger that can trick the model, which is the union of all triggers. This trigger may be discoverable by some defense methods. On the contrary, our method is probabilistic, meaning no single trigger leads the model to the target label.

2. Related work

In this section, we provide an overview of recent research on trojan attacks and defenses on neural networks.

2.1. Attacks

Nguyen and Tran [8] contend that numerous attacks rely on a uniform trigger pattern effective across all inputs, making them easily detectable. In response, they introduce an input-aware dynamic attack, wherein the trigger depends on the input. Li et al. [9] studied the impact of the location and appearance of the backdoor trigger on activating the backdoor. In this way, they proposed a poisoning method to enhance attack robustness against various geometric transformations of input images. He et al. [10] suggested using a “scapegoat” trigger alongside the real trigger. The defender would recognize the scapegoat trigger while missing the real one.

Beyond image data, researchers have explored attacks on other data types. Zhang et al. [11] proposed a method for poisoning Graph Neural Networks (GNNs) by introducing a sub-graph as a trigger into the input. In the realm of deep learning, using a representation vector output by a pre-trained network for downstream tasks is common. Zhang et al. [12] demonstrated that attackers can train a model to create detrimental representations, corrupting any downstream task, termed Neuron-level Backdoor Attack (NeuBA).

Some attack strategies extend beyond modifying input in the pixel space, working in a latent feature space. It poses more challenge to defenders. For example, Saha et al. [13] perturb the image so that the result similar to the source image in the pixel space, but similar to target class in the feature space. The DeepPoison attack [14] extracts hidden features from the target class and inserts them into the benign input, trying to preserve the appearance of the input at the same time.

All reviewed attacks so far are deterministic techniques. Salem et al. [6] presented the first probabilistic backdoor attack i.e. it is not a priori known whether the attack will work or not, and more than one trial is needed. However, it needs hundreds of trials. In this paper, we propose a multi-trigger attack called Projan. It is another probabilistic method which needs only a few trials. The probabilistic nature of this attack makes it more stealthy. Another method that employs multiple triggers is N-to-One [7], which uses different triggers separately in training stage, and uses all of them together at test time. However, unlike Projan, it is not a stochastic attack.

2.2. Defenses

Defense strategies against trojan attacks broadly fall into categories such as input filtering, input reformation, model inspection, and model sanitization. Input filtering checks the input data at test time for triggers while input reformation alters the input to neutralize possible triggers. Model inspection involves examining the model for backdoors, and model sanitization patches the model to eliminate backdoors [15]. Qiu et al. [16] proposed the DeepSweep framework to automatically evaluate and generate defense methods against backdoor attacks. They provided an end-to-end solution based on data augmentation techniques to remove the backdoor via fine-tuning and counteract the trigger effects via inference preprocessing. The paper introduces methods for both model sanitization and input reformation. The ACQ [17] method is based upon the observation that triggered inputs cause outlier values in intermediate layers of the network. The authors propose to add clipping and quantization operations within the network to mitigate such extreme values. Gao et al. [18] propose STRIP, an input filtering method. They deliberately perturb incoming inputs by superimposing various image patterns, and analyze the entropy in outputs as an indicator of backdoor.

Several works propose model inspection methods, including [4,19–21]. Wang et al. [4] introduce Neural Cleanse, which searches for small perturbations that lead all inputs to a certain target class. The authors also propose techniques for input inspection filtering and model sanitization. DeepInspect [20] searches for such perturbations using a conditional Generative Adversarial Network (cGAN) approach. Sikka et al. [19] postulate that a small number of “ghost neurons” are responsible for incorrectly attributing the input to the target class. They identify such neurons using benign data by examining their contributions to classes other than the one with highest score. Ying et al. proposed GNNExplainer [21], a method designed to explain predictions made by Graph Neural Networks (GNNs). GNNExplainer is employed to analyze the impact of trigger injection positions for backdoor attacks on GNNs, particularly in the context of graph classification tasks.

Regarding model sanitization techniques, Zheng et al. [22] propose Channel Lipschitzness Pruning (CLP) as a model sanitization technique. They calculate an upper bound for the Lipschitz constant of channels as a measure of sensitivity to input, potentially indicating the presence of a backdoor. Channels with a high value of this measure are pruned. Model orthogonalization (MOTH) [23] modifies the model, maximizing class distances. In addition, Li et al. [24] observe that the distribution of internal model activations differs between benign and triggered inputs. They design a defense method that inserts simple operations inside the model to counteract the distribution shift for infected inputs. They only apply this method to inputs detected as triggered. Thus, their approach can be considered as an input filtering as well as model sanitization method. Weber et al. [25] adopt a distinct approach in defending against backdoor attacks. They provide a framework capable of certifying a model's robustness against trojan attacks and training data poisoning. Additionally, they propose a training method that fosters the development of robust models through the utilization of randomized smoothing.

Some defense methods including MOTH and DeepInspect do not need training data, making them more useful for situations where such data are not available. Also, some defense strategies including DeepInspect and STRIP only rely on the model input and output without regard to its internal architecture or weights, making them more general purpose. As mentioned before, some methods including Neural Cleanse and DeepInspect assume that there exists a trigger that works with all input data. We introduce a stochastic attack that subverts this assumption.

3. Preliminaries

We denote a neural network classifier by a vector-valued function f with parameters θ: $o_i = f_\theta(\mathbf{x})[i]; i \in \{1, \dots, C\}$ where $\mathbf{x}$ is the input to the model, o_i is the likelihood of $\mathbf{x}$ belonging to class i as predicted by the model, and C is the number of classes. The subscript θ may be omitted for brevity. The index of the output neuron with the highest value corresponds to the class that the model predicts as the class of the input, i.e. input $\mathbf{x}$ is assigned to class i^* where $i^* = \arg\max_i o_i$.

In a benign setting, The training process involves a dataset D consisting $(input, output)$ pairs and the goal is to minimize the distance between the true conditional distribution of output given input and the distribution approximated by the model. To achieve this, the cross-entropy loss is minimized:

$$\min_{\theta} \mathop{\mathbb{E}}_{\mathbf{x},y\sim D} [CE(f_\theta(\mathbf{x}), y)] \tag{1}$$

where CE is the cross-entropy loss.

In the threat model that we consider, the adversary has access to part or all of the training data and can train or fine-tune the model. They implant a trojan in the model at training time and direct the model to output their desired (target) label at test time by providing input stamped with their trigger. This can happen in a scenario where the model training is outsourced to a malicious party. We denote the target class by $t \in \{1, 2, \dots, C\}$. Poisoning involves making a small perturbation to the input, typically achieved by adding a trigger. We denote the operation of adding a trigger $\mathbf{t}$ to an input $\mathbf{x}$ by $p(\mathbf{x}, \mathbf{t})$.

The trigger $\mathbf{t}$ is composed of a patch $\mathbf{s}$ and a mask $\mathbf{m}$. The poisoning process is defined as follows:

$$p(\mathbf{x}, \mathbf{t}) = p(\mathbf{x}, \mathbf{s}, \mathbf{m}) = (1 - \mathbf{m}) \odot \mathbf{x} + \mathbf{m} \odot \mathbf{s} \tag{2}$$

where $\odot$ denotes the Hadamard product. The mask is usually a binary image that effectively crops part of the input and replaces it with the patch. In a more general case, however, M can have non-binary values.

The adversary has two objectives: *Objective I* is not to degrade the model's performance over benign data to minimize the risk of detection. This objective is similar to a benign training process and can be formulated with Eq. (1).

Objective II is to trick the model into classifying an arbitrary input into the desired class by poisoning it as described above. This objective can be modeled as follows:

$$\min_{\theta} \mathop{\mathbb{E}}_{\mathbf{x},y\sim D} [CE(f_\theta(p(\mathbf{x}, \mathbf{t})), t)]. \tag{3}$$

In other words, the adversary aims to maximize the Attack Success Rate (ASR), which is the accuracy of the model when the ground truth is set equal to the target class. In the next section, we describe our trojan attack.

4. The probabilistic attack

In this section, we first define our threat model and its theoretical foundation, and then move on to the its implementation details.

4.1. Attack design

Our probabilistic threat model is similar to former deterministic attacks in that the adversary trains/fine-tunes the model and at test time, provides the model with input infected with a trigger. The difference is that in previous attacks, the attacker only uses one trigger at test time and expects it to output the target label. However, in our work, the attacker has n predefined triggers and tests them sequentially until one of them works.

Our objective is to design a "probabilistic" trojan attack, where a trigger only with some probability will a trick the model. In order for the attack to be ultimately successful, more than one trigger is needed. The attacker tries the triggers one after another until the attack succeeds.

Here, *Objective I* is still relevant, as the model accuracy on benign data should remain high (low accuracy decline w.r.t. the benign model). To achieve this, we define the following loss function, which is the application of Eq. (1) to a mini-batch:

$$L_{benign} = \sum_{(\mathbf{x},y)\in B} CE(f_\theta(\mathbf{x}), y) \tag{4}$$

where B denotes the mini-batch.

In Projan, there are multiple triggers, each of which works for a given input with some probability. Let a trigger's "Active Space" be defined as the subset of the input space for which the trigger can mislead the model. "Input space" refers to the set of all possible inputs. If the input image is of size $C \times H \times W$, then the input space will be the vector space of dimensionality $\mathbb{R}^{C\times H\times W}$.

Let A_i denote the probabilistic event that an input randomly sampled from the space of possible inputs is a member of the ith trigger active space. To attack the deployed model, the attacker uses the triggers sequentially until one of them works. Thus, for the attack to be ultimately successful, the union of the active spaces must cover the whole input space. On the other hand, the active spaces must be small to remain stealthy. Consequently, the active spaces must have little to no overlap (see Fig. 1). Therefore, if we have n triggers, A_i must be close to $1/n$ or

$$\min |\mathbb{P}(A_i) - \frac{1}{n}|;\ \forall i = 1, \dots, n. \tag{5}$$

In summary, we have three requirements:

I. Each active space must cover $1/n$ of the whole space,
II. Active spaces must be disjoint, and
III. The active spaces must cover the whole space.

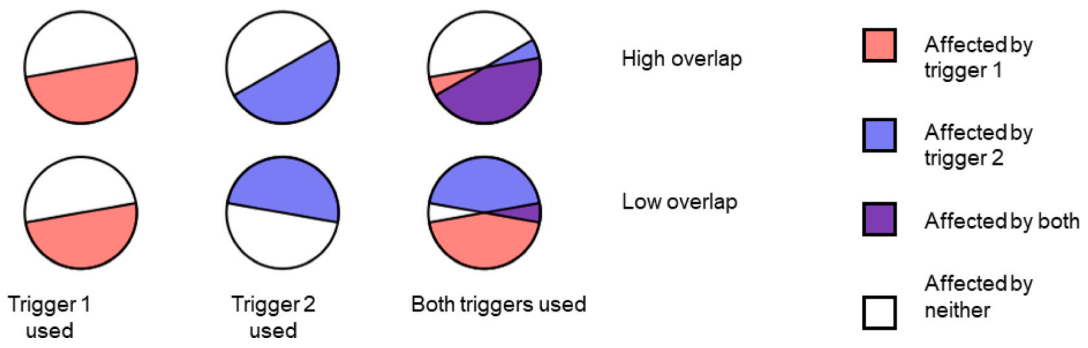

Fig. 1. Importance of the amount of overlap between the active space of different triggers. The circle represents the entire input space. The shaded areas depict parts of input space influenced by one or two triggers. The first row shows a case with high overlap between the two active spaces, while the second row shows a case with low overlap. As can be seen, lower overlap means a higher overall attack success rate (see third column). Note that the circles are only schematic representations of high-dimensional vector spaces. Furthermore, although the active spaces are illustrated as contiguous, shaded areas within the circles, in reality, each active space may be dispersed across the entire input space.

Intuitively, it can be seen that I and II imply III; and I and III imply II. Therefore, we can enforce the first requirement and one of the other two.

To enforce Requirement I, we approximate $\mathbb{P}(A_i)$ with the accuracy of the network on triggered inputs over the mini-batch, leading to the following loss:

$$L_{poison} = \sum_{i=1}^{n} |\frac{1}{|B|} \sum_{(\mathbf{x},y)\in B} f_\theta(p(\mathbf{x}, \mathbf{t}_i))[t] - \alpha \frac{1}{n}| \tag{6}$$

where $|B|$ denotes the mini-batch size and $\mathbf{t}_i$ is the ith trigger. In our initial experiments, we realized that with this loss, the accuracy tends to converge to a value higher than $1/n$. Therefore, we introduced the hyper-parameter α in the above equation to provide more flexibility in optimizing the loss. We found 0.75 to be a suitable choice for α. To apply Requirement II, one can directly minimize the overlap between all pairs of active spaces, i.e.

$$- \sum_{i,j;i>j} \frac{1}{|B|} \sum_{(\mathbf{x},y)\in B} |(f_\theta(p(\mathbf{x}, \mathbf{t}_i))[t] - f_\theta(p(\mathbf{x}, \mathbf{t}_j))[t])|. \tag{7}$$

Requirement III means that we want at least one trigger to mislead the model to the target class. Therefore, for each instance $\mathbf{x}$, the maximum of the network output for the target class over all triggers, i.e. $\max_i f_\theta(p(\mathbf{x}, \mathbf{t}_i))[t]$ must be close to one (maximized). Thus, we take the sum of this maximum over the batch and maximize it, or equivalently minimize its negative. Hence, the loss function is defined as follows:

$$L_{pull} = - \sum_{(\mathbf{x},y)\in B} \max_i f_\theta(p(\mathbf{x}, \mathbf{t}_i))[t]. \tag{8}$$

As we said, it is optional to implement either Req. II (Eq. (7)) or Req. III (Eq. (7)). Since the number of summation terms grows quadratically with the number of triggers in Eq. (7), we only used Eq. (8) in our experiments.

Finally, we define the ultimate loss function as a weighted sum of the three loss terms defined above:

$$L = L_{benign} + \lambda_1 L_{poison} + \lambda_2 L_{pull} \tag{9}$$

where λ_1 and λ_2 denote the relative importance of the respective losses.

4.2. Implementation details

In this section, we cover some issues related to the implementation of our attack. The first aspect we examine is the effect of batch normalization. Batch normalization is a technique proposed to improve the speed of neural network training by normalizing the distribution of each batch of training data [26]. It helps prevent the problem known as "covariate shift".

Through our initial experiments, we observed that batch normalization inhibits the model from converging to a suitable state. Specifically, the loss value decreases as desired when evaluated on training data, but it exhibits unstable behavior on validation data.

To mitigate this, we disabled batch normalization. However, to preserve its advantages, we followed a two-stage training process. In the first stage, we pre-trained the network for 100 iterations with only the benign loss and batch normalization activated. In the second stage, we disabled batch normalization and trained the model with the full loss function for another 100 iterations. The increased number of iterations is not necessarily a problem, as in practice, the attacker might want to poison an already-trained model, which is essentially equivalent to the first stage of our training.

Another point to consider in the implementation is the poisoning ratio, i.e. the ratio of training data that is poisoned. Some works assume the adversary can only poison a limited percentage of the training data to remain stealthy [27]. The TrojanZoo framework sets this ratio to 0.01 by default. However, we found this value insufficient for the probabilistic attack. The reason may be that our method is more stochastic than BadNet, and therefore, it requires more poisoned training data.

Nevertheless, we do not need to poison a large portion of the training set, as acceptable results were obtained with a poisoning ratio of 0.1. Keeping the poisoning ratio low is also important to reduce training costs. Computing Eqs. (6) and (8) requires poisoning the input with all the triggers which adds a computational burden. With a small poisoning ratio, however, this poisoning needs to be done only on a small fraction of data.

5. Experiments

In this section, we describe the experimental setup, assess the effectiveness of our attack, and verify its resilience against defense systems.

5.1. Experimental setup

Our proposed method is built on top of TrojanZoo, "the first open-source platform for evaluating neural backdoor attacks/defenses in a unified, holistic, and practical manner" [15]. We selected two commonly used datasets in DNN backdoor literature: CIFAR-10 [28] and the MNIST dataset of handwritten digits [29].

In our experiments, we utilized triggers in the form of white squares of 3×3 pixels placed at various locations over the input image.

For CIFAR-10, which consists of 32×32 RGB images of ten object classes, we used a smaller version of ResNet18 [30] called ResNet18_comp provided by TrojanZoo. For MNIST, which contains 28×28 grayscale images of handwritten digits (class labels 0 through 9), we employed a simple convolutional neural network (ConvNet) with two convolutional and two dense layers provided by TrojanZoo.

Table 1
Attack Results on CIFAR-10.

Attack	Clean Acc.	ASR	Min ASR	Max ASR
BadNet	91.00	94.14	N/A	N/A
Nto1	**91.35**	**99.78**	N/A	N/A
Projan2	90.97	95.61	41.45	56.89
Projan3	90.93	95.52	29.54	41.36
Projan4	91.29	93.59	22.48	**27.45**
Projan5	91.30	98.42	**43.13**	63.01

Table 2
Attack Results on MNIST.

Attack	Clean Acc.	ASR	Min ASR	Max ASR
BadNet	98.91	100.00	N/A	N/A
Nto1	98.88	81.53	N/A	N/A
Projan2	98.95	**97.55**	**48.73**	50.40
Projan3	**98.97**	96.83	32.14	37.47
Projan4	99.01	97.73	25.22	37.46
Projan5	**98.97**	97.51	15.95	**25.78**

We conducted experiments with different numbers of triggers, ranging from two to five, and referred to these attacks as *Projan2* through *Projan5*, or more generally *Projan*. Additionally, we used the *BadNet* [31] and *N-to-One* (also shown as Nto1 for short) [7] attacks as baselines for comparison. In N-to-One attack, we set the number of triggers N to four, as suggested by the original paper.

The poisoned models were subjected to testing with five defense methods: Neural Cleanse [4] and DeepInspect [20] for model inspection; CLP [22] and MOTH [23] for model sanitization; and STRIP [18] for input filtering aiming to assess the capability of our proposed attack to evade detection.

In each trial, we designated class 0 as the target class, representing digit zero in MNIST and "airplane" in CIFAR-10. For every combination of attack and dataset, we conducted ten trials, running the training algorithm ten times to generate ten models. Subsequently, defense algorithms were evaluated on each of these models.

5.2. Attack evaluation

We reported the clean accuracy and ASR on CIFAR-10 and MNIST in Tables 1 and 2, respectively. For our method, we measured ASR for each trigger individually. The ultimate ASR for Projan is defined as the ratio of examples for which at least one of the triggers can mislead the model. The rationale behind this is based on the assumption that the attacker must try the triggers one by one until achieving success.

Additionally, as mentioned in Section 3, each trigger should have a small ASR. Therefore, we reported the maximum ASR per trigger, which ideally equals the inverse of the number of triggers. Furthermore, we measured the minimum ASR per trigger. A low value for this metric indicates that some of the triggers are making little contribution to the attack's success. All the Clean Accuracy, Total ASR, min ASR and max ASR are averages over ten trials.

Overall, our probabilistic method achieved clean accuracy similar to BadNet and N-to-One on both datasets. The ASR was comparable or better than BadNet on CIFAR-10, and slightly lower than BadNet on MNIST. The ASR was smaller than or comparable to N-to-One on CIFAR-10, and was better on MNIST. Thus, both *Objectives I* and *II* (as outlined in Section 4) have been fulfilled.

Ideally, the maximum per-trigger ASR should be equal to the inverse of the number of triggers ($1/n$). In the majority of cases, the max ASR is close to $1/n$. However, there are exceptions, such as the Projan3 and Projan5 attacks on CIFAR-10, as well as the Prob4 attack on MNIST.

5.3. Defense evaluation

We assessed the effectiveness of defense methods in detecting trojaned models and/or reverse-engineering the trigger using three categories of defense namely model inspection (Neural Cleanse and DeepInspect), model sanitization (CLP and MOTH) and input filtering (STRIP). Both model inspection techniques, Neural Cleanse and DeepInspect, are capable of reverse-engineering the trigger used in the attack.

Certain methods in the literature reverse-engineer the trigger by solving an optimization problem. Consequently, often a trigger can be found, even for benign models. Therefore, specific criteria are established to determine the nature of the discovered trigger as indicative of a real trojan attack. Neural Cleanse and DeepInspect operate under the assumption that the trigger found for the trojaned class must be small (i.e. have a small $L1$ norm). Thus, they initially generate triggers for all classes, and if the $L1$ norm of a trigger is an outlier, the corresponding class is reported as trojaned.

Outliers are identified using the Median Absolute Deviation (MAD) method. The MAD of a set x_i is defined as follows: Let M denote the median of $x_i (M = med(x_i))$, and let D_i represent the absolute differences from M, i.e., $d_i = |x_i - M|$. The MAD is defined as a constant multiplied by the median of d_i, i.e., $MAD = c \cdot med(d_i)$. Typically, the c is set to 1.482 [32]. Assuming a Gaussian distribution for x, if $d_i/MAD > 2$, x_i is considered an outlier with a confidence value of 0.05.

Since only small values of $L1$ norm are of interest, the left-tail version of the test is employed, wherein x_i is considered an outlier if $d_i/MAD > 2$ and $x_i < M$.

5.3.1. Neural cleanse

Neural Cleanse generates minimal triggers that can change the model output to the target class for all inputs. All classes are considered as target in turn. The defender has access to the model and some clean (correctly labeled) data [4]. Table 3 presents the results of Neural Cleanse on CIFAR-10 and MNIST. It is notable that Neural Cleanse successfully detects models trojaned by BadNet in all CIFAR-10 trials. However, it detects a maximum of two models poisoned by Projan. The anomaly index is smaller for Projan. N-to-One lies between BadNet and Projan in terms of number of detected models (four) and anomaly index (3.08).

On MNIST, eight out of ten models infected by BadNet are detected. The number of detections for models poisoned by Projan decreases with the number of triggers from seven to zero models. Additionally, the anomaly index decreases. None of the attack cases of N-to-One are detected, which is same as Projan5. The anomaly index is also close to Projan5.

Neural Cleanse generates candidate triggers that are small and direct the model to the target class. This is achieved by minimizing a loss function consisting of two terms: one that is similar to Eq. (3) and maximizes ASR and another term that penalizes large triggers. Most attack approaches seek to make ASR close to one, making them vulnerable to Neural Cleanse. That is while Projan intentionally brings the per-trigger ASR close to $1/n$ using Eq. (6). As the number of triggers (n) grows, the value of per-trigger ASR decreases, making it easier to evade detection.

Results of reverse-engineering triggers by Neural Cleanse are illustrated in Fig. 2. This defense mechanism captures the trigger inserted by BadNet on CIFAR-10 with high accuracy in all trials. The output for one trial is depicted in Fig. 2(a). Concerning MNIST, the trigger is not precisely recovered, but the extracted trigger encompasses the original one, i.e. a white square (appearing as black in the negative image) is visible on the upper left corner, Fig. 2(b).

Neural Cleanse struggles to identify the triggers inserted by Projan on CIFAR-10 (Fig. 2(c)), except for a few Projan5 trials where one of the triggers is recovered (Fig. 2(d)). Regarding MNIST, the defense can recover the union of triggers when only two triggers are used (Fig. 2(e)). However, as the number of triggers increases, its accuracy in retrieving them diminishes (Fig. 2(f)).

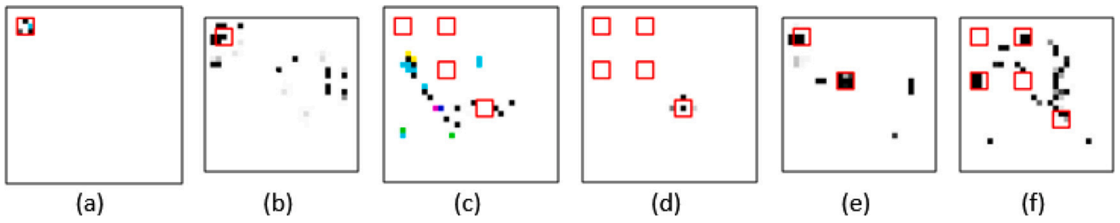

Fig. 2. Results of reverse-engineering of triggers by Neural Cleanse. The outline of ground truth triggers are shown as red squares. The images have been negated and their contrast has been enhanced. (a) CIFAR-10 under BadNet attack, (b) MNIST under BadNet attack, (c) CIFAR-10 under Projan4 attack, where triggers are not captured correctly, (d) CIFAR-10 under Projan5 attack with one trigger successfully recovered, (e) MNIST under Projan2 attack with the union of triggers detected, (f) MNIST under Projan5 attack where triggers are detected with less accuracy.

Table 3
Results of Neural Cleanse.

Dataset	CIFAR-10		MNIST	
Attack	# Det	Avg Anomaly Index	# Det	Avg Anomaly Index
BadNet	10	4.43	8	2.29
Nto1	4	3.08	0	1.63
Projan2	1	1.61	7	3.35
Projan3	0	2.66	4	2.63
Projan4	0	**1.59**	1	1.64
Projan5	2	1.97	0	**1.38**

Table 4
Results of DeepInspect.

Dataset	CIFAR-10		MNIST	
Attack	# Det	Avg Anomaly Index	# Det	Avg Anomaly Index
BadNet	10	4.17	5	5.48
Nto1	10	3.78	4	2.30
Projan2	1	4.55	1	**2.09**
Projan3	0	**1.85**	8	4.55
Projan4	0	1.87	8	5.53
Projan5	0	2.76	3	5.17

5.3.2. *DeepInspect*

DeepInspect [20] generates triggers through a generative adversarial training. It needs access to the model, but unlike Neural Cleanse does not need training data. The outcomes of the DeepInspect defense are presented in Table 4. Comparatively, the probabilistic attack generally outperforms BadNet on CIFAR-10 but fares worse on MNIST. On CIFAR-10, all BadNet attacks were detected, while only one probabilistic attack was detected when two trigger were used, with none detected with more triggers. The anomaly index is lower for the probabilistic attack, except for the two-trigger case (Projan2). All N-to-One attacks are also detected, with a relatively high anomaly index.

On MNIST, DeepInspect identifies 5 out of 10 BadNet attacks. The probabilistic attack demonstrates more stealthiness with two triggers, with only one case detected (where the anomaly index is also lower). However, with more triggers, the performance declines. Four models trojaned by N-to-One are detected. However, the anomaly index is relatively low.

DeepInspect uses a generative adversarial training procedure to produce candidate triggers. It uses two loss terms similar to those used by Neural Cleanse and an extra term L_{GAN} that penalizes infected images that are highly distinguishable from the benign image. As DeepInspect penalizes ASR similar to Neural Cleanse, detection of Projan attacks is more challenging for it.

Fig. 3 illustrates the trigger detection results of DeepInspect. It can capture the trigger with high accuracy in most cases of the BadNet attack on both CIFAR-10 (3(a)) and MNIST (Fig. 3(b)). However, the reverse-engineered trigger generated for Projan2 and Projan3 attacks on CIFAR-10 fails to capture the inserted triggers (Fig. 3(c, d)).

Although the Projan attack is detected in some cases on MNIST, the triggers are not precisely found. Nonetheless, there is some similarity between the recovered trigger and the union of the original triggers in some instances (Fig. 3(e)).

5.3.3. *CLP*

CLP prunes the network to minimize its Lipschitz constant, which is the maximum ratio of output change to the input change. The defender needs access to model weights, but does not require any data [22]. We successfully executed it solely on the ResNet model (for the CIFAR-10 dataset) without encountering errors. The results are depicted in Fig. 4. It is evident that it effectively decreased the ASR for BadNet to as low as 17.8%. However, for Projan, the ASR after defense increases with the number of triggers. The ASR reduction for N-to-One is comparable to Projan4 and more than Projan5.

By minimizing the Lipschitz constant, CLP increases the displacement vector needed to move a sample to the target class. In the context of trojan attacks, this displacement vector corresponds to the trigger. As the Lipschitz constant of the network is independent of the vector direction, it is plausible to assume that it would neutralize all possible triggers, including those chosen by Projan.

However, as seen from the experimental results, it does not mitigate the Projan attack to a large degree. The following argument might explain the result: CLP makes some simplifications in approximating the Lipschitz constant and sanitizing the network. For example, it prunes some channels out of each layer to reach a smaller Lipschitz constant, which is a greedy, sub-optimal approach. Furthermore, it reshapes the weight tensor into a matrix and computes its spectral norm, which is an approximation to the spectral norm of the original tensor.

Therefore, in theory, CLP maximizes the distance between two classes in all possible directions. However, in practice, some directions (triggers) may be left uncovered for some data points. Considering a single trigger, the likelihood that it still works after defense for all inputs is lower, mitigating the BadNet attack. On the contrary, in Projan, we have multiple triggers, and if some of them do not work, we can try other ones, which makes Projan more robust against CLP.

5.3.4. *MOTH*

MOTH sanitizes the model by maximizing class distances. It defines class distance as the distance from the center of mass of one class samples to the decision boundary between the two classes. The defender needs some training data as well as the model [23]. The results of MOTH are presented in Fig. 4 for CIFAR-10 and Fig. 5 for MNIST. The post-defense ASR on CIFAR-10 for BadNet is 19.76% and increases with the number of triggers from 69.99% for Projan2 to 91.06% for Projan5. For N-to-One, the post-defense ASR is 85.16%, which is comparable to Projan4 and less than Projan5. On MNIST, MOTH cannot do noticeable mitigation, except for N-to-One where it reduces ASR from 81.53% to 75.91%.

MOTH approximates class distance by the size of a minimum backdoor that would translate inputs from the victim to the target class. This distance is computed for a random set of samples. The trigger is generated using an optimization procedure similar to [4]. In Projan, the triggers are designed to translate only part of the samples into the

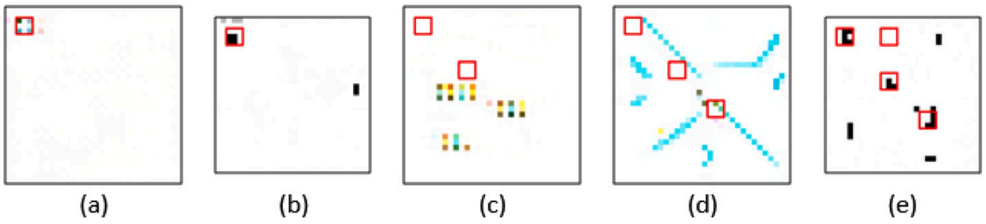

Fig. 3. Results of trigger reverse-engineering by DeepInspect. Ground truth trigger outlines are marked with red squares. Images have been negated and contrast-enhanced. (a) BadNet on CIFAR-10: trigger precisely located. (b) BadNet on MNIST: successful detection. (c) Projan2 on CIFAR-10: triggers not detected. (d) Projan3 on CIFAR-10: triggers not detected. (e) Projan4 on MNIST: detected pattern resembles the union of triggers.

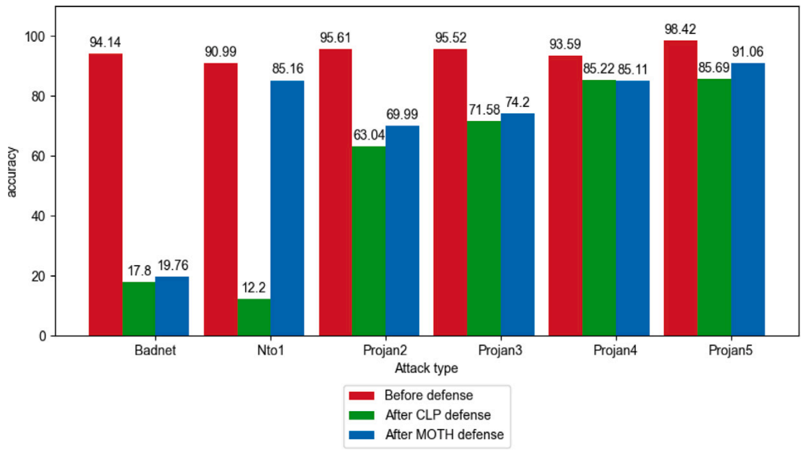

Fig. 4. Attack success rate on CIFAR-10 before and after sanitizing model with CLP and MOTH.

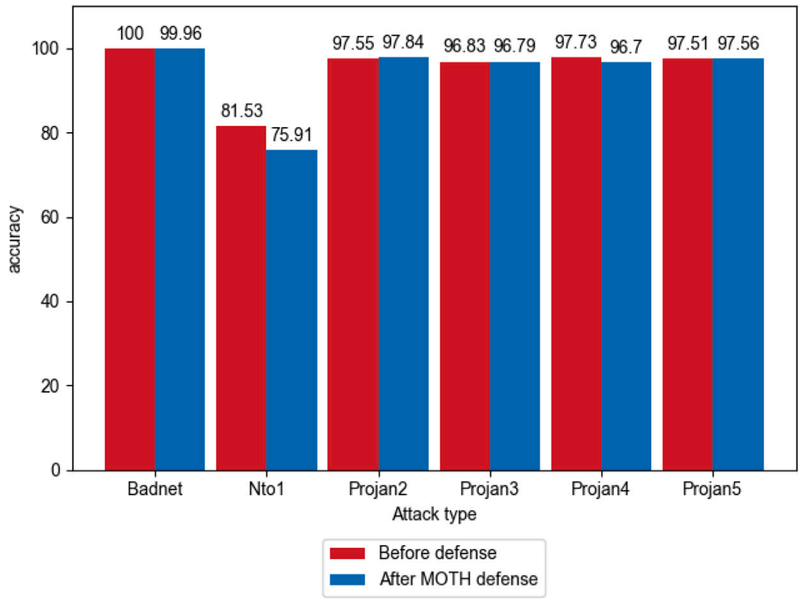

Fig. 5. Attack success rate on MNIST before and after sanitizing model with MOTH.

target class. As a result, the class distance computed by MOTH for each of the triggers tends to be already a high value when averaged over a set of samples. Therefore, when MOTH tries to harden the model, its optimization might not further increase the class distance. As the number of triggers of Projan increase, fewer instances are affected by each trigger, and therefore the class distance will be wider before the defense. This can explain why the effectiveness of MOTH reduces with the number of triggers on CIFAR-10.

5.3.5. STRIP

STRIP [18] is an input inspection method, which is executed at test time, trying to capture infected inputs. It requires access to the model and some clean data. We report post-defense ASR of STRIP as

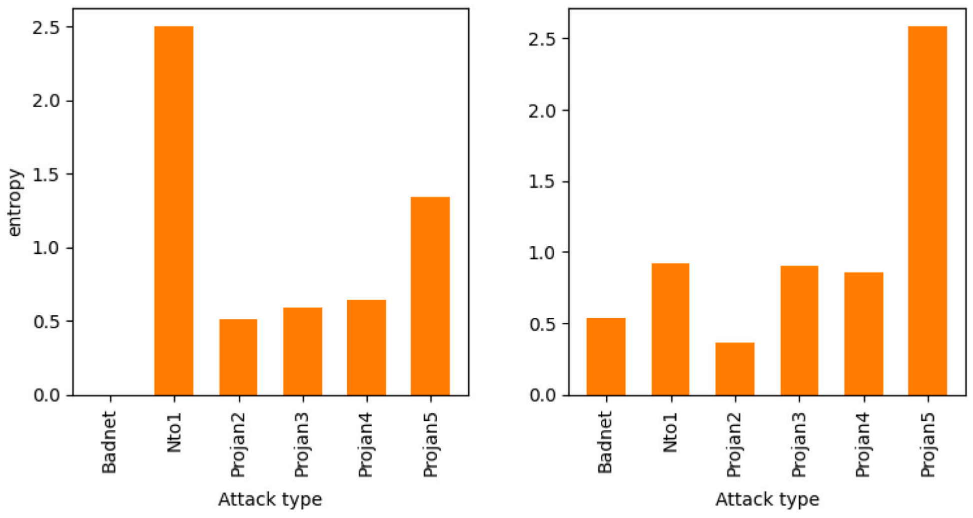

Fig. 6. Average network output entropy for trojaned input for MNIST (left) and CIFAR-10 (right).

the ratio of trojaned examples that are both missed by STRIP and are misclassified to the target class by the model. For Projan, where there is more than one trigger, we consider the attack unsuccessful either if an alarm is raised before the first successful trigger, or if all triggers fail.

The idea behind STRIP is that a potential trigger must work for most inputs. Therefore, making perturbations to a malicious input must cause little change to the model output. STRIP makes several perturbations to the input and if the model outputs for them have little entropy, it reports the input as malicious.

In our experiments on CIFAR-10 it made very few detections. The detection rate was 0.03% percent for BadNet and 0.01% for Projan2 and zero for all other Projan and N-to-One attacks. On MNIST, it reduced the ASR to as low as 5.80 percent. For the other attacks, it made no detections.

To gain a better insight into the performance of STRIP, we measured the average output entropy for trojaned inputs. Fig. 6 shows the result. As can be seen, for Projan, the entropy of trojaned input almost increases with the number of triggers, making it challenging for STRIP to detect the attack. N-to-One also leads to a high entropy. Badnet has a small entropy on MNIST and its entropy on CIFAR-10 is more than Projan2 and less than Projan3-5.

5.4. Hyperparameter tuning

The algorithm has two hyperparameters λ_1, λ_2 (Eq. (9)). To determine their optimal values, we conducted a grid search using the MNIST dataset. We varied λ_1 and λ_2 from 0.25 to 2.0 in increments of 0.25, resulting in eight values for each λ and 64 combinations in total.

Applying certain minimum criteria, we filtered out some values. These criteria included: ensuring benign accuracy exceeds 0.95, restricting min and max ASR to be greater than 30 and less than 45 respectively, and requiring overall ASR to be above 0.85. This filtering left us with six combinations of λs. Across all combinations, benign accuracy and overall ASR consistently surpassed 0.98, while min ASR ranged from 0.31 to 0.33, and max ASR ranged from 0.37 to 0.43. We opted for the combination yielding the lowest max ASR, corresponding to $(\lambda_1, \lambda_2) = (0.25, 0.75)$.

Fig. 7 illustrates the results plotted for selected values of λ_1. In each plot, the horizontal axis represents λ_2. The metrics exhibit almost continuous variations with changes in λ_2 and across different values of λ_1. Specifically, as λ_1 increases, the curves for min and max ASR get closer. However, max ASR exhibits some fluctuation.

6. Conclusion and future work

In this study, we introduced a probabilistic approach for embedding backdoors in deep networks. Unlike traditional methods that rely on a single strong trigger, our approach generates multiple weak triggers. While each trigger individually has a lower success probability in fooling the model, collectively they can achieve a high success rate when tested in sequence. By reducing the success rate of individual triggers, our approach makes it more challenging for defense mechanisms to detect the backdoor. However, it requires the attacker to execute the attack multiple times.

We conducted experiments on two widely-used datasets, MNIST and CIFAR-10, to evaluate the effectiveness of our proposed method. The results demonstrate that the triggers collectively achieve an acceptable attack success rate. Moreover, we examined the robustness of our approach against a wide range of defense algorithms. In most cases, defense methods had greater difficulty in detecting or mitigating the backdoor when faced with Projan.

As this kind of attack has similar counterparts in cybersecurity (such as brute force attack), defense strategies inspired by cybersecurity paradigms could enhance the resilience against such attacks. For example, applying concept of “statefulness” involves monitoring user behavior across consecutive model usages. This is in contrast to current defense methods that only consider the current input, ignoring state information. As this kind of attack has similar counterparts in cybersecurity (such as brute force attack), defense strategies inspired by cybersecurity paradigms could enhance the resilience against such attacks. For example, applying concept of “statefulness” involves monitoring user behavior across consecutive model usages. This is in contrast to current defense methods that only consider the current input, ignoring state information. Another approach could be limiting the number of each user’s requests.

On the other hand, if the adversary has access to a copy of the model, especially if training has been outsourced to the adversary, they can first try all the triggers and find the promising one before running the actual attack, which reduces the attack to only one trial. Another countermeasure for the attacker is to add some delays between trials to reduce suspicion.

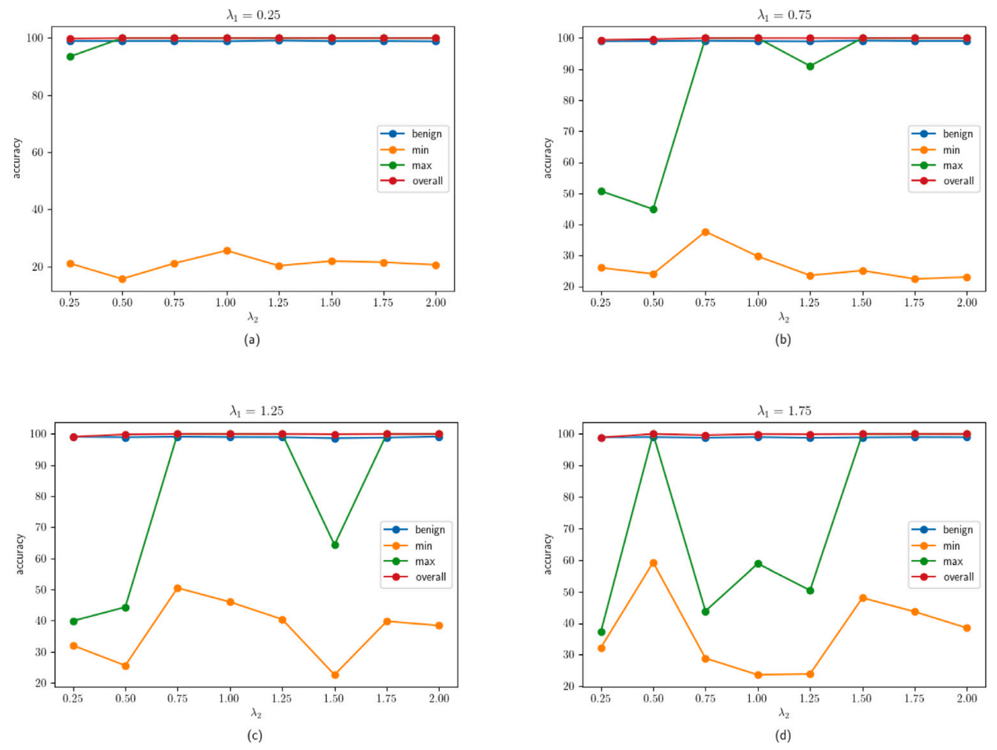

Fig. 7. Accuracy variation with different λ values. Each sub-figure corresponds to a λ_1 value: (a) $\lambda_1 = 0.25$, (b) $\lambda_1 = 0.75$, (c) $\lambda_1 = 1.25$, (d) $\lambda_1 = 1.75$. The x-axis represents λ_2.

Funding sources

This work was supported by The Institute for Research in Fundamental Sciences (IPM).

CRediT authorship contribution statement

Mehrin Saremi: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Mohammad Khalooei:** Writing – review & editing, Investigation. **Razieh Rastgoo:** Writing – review & editing, Investigation. **Mohammad Sabokrou:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Publicly available datasets were used in this research. The code is available at https://github.com/programehr/Projan.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work the authors used ChatGPT and Grammarly in order to improve the language of the paper. After using these services, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

References

[1] T. Talaei Khoei, H. Ould Slimane, N. Kaabouch, Deep learning: systematic review, models, challenges, and research directions, Neural Comput. Appl. 35 (31) (2023) 23103–23124, http://dx.doi.org/10.1007/s00521-023-08957-4.

[2] J. Liu, Y. Jin, A comprehensive survey of robust deep learning in computer vision, J. Autom. Intell. 2 (4) (2023) 175–195, http://dx.doi.org/10.1016/j.jai.2023.10.002.

[3] B. Yamini, V. Prasanna, C. Ambhika, M. Anuradha, B. Maheswari, S.R. Subramanian, M. Nalini, A comprehensive survey of deep learning: Advancements, applications, and challenges, Int. J. Recent Innov. Trends Comput. Commun. 11 (2023) 445–453, http://dx.doi.org/10.17762/ijritcc.v11i8s.7225.

[4] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, B.Y. Zhao, Neural cleanse: Identifying and mitigating backdoor attacks in neural networks, in: Proceedings - IEEE Symposium on Security and Privacy, Vol. 2019-May, Institute of Electrical and Electronics Engineers Inc., 2019, pp. 707–723, http://dx.doi.org/10.1109/SP.2019.00031.

[5] Y. Liu, S. Ma, W.C. Lee, Y. Aafer, G. Tao, X. Zhang, ABS: Scanning neural networks for back-doors by artificial brain stimulation, in: Proceedings of the ACM Conference on Computer and Communications Security, 2019, pp. 1265–1282, http://dx.doi.org/10.1145/3319535.3363216.

[6] A. Salem, M. Backes, Y. Zhang, Don't trigger me! a triggerless backdoor attack against deep neural networks, 2020, arXiv:2010.03282. URL http://arxiv.org/abs/2010.03282.

[7] M. Xue, C. He, J. Wang, W. Liu, One-to-N & N-to-one: Two advanced backdoor attacks against deep learning models, IEEE Trans. Dependable Secure Comput. 19 (3) (2022) 1562–1578, http://dx.doi.org/10.1109/TDSC.2020.3028448.

[8] T.A. Nguyen, T.A. Tran, Input-aware dynamic backdoor attack, 2020, arXiv:2010.08138. URL http://arxiv.org/abs/2010.08138.

[9] Y. Li, T. Zhai, B. Wu, Y. Jiang, Z. Li, S. Xia, Rethinking the trigger of backdoor attack, 2020, arXiv:2004.04692. URL http://arxiv.org/abs/2004.04692.

[10] Y. He, Z. Shen, C. Xia, J. Hua, W. Tong, S. Zhong, SGBA: A stealthy scapegoat backdoor attack against deep neural networks, Comput. Secur. 136 (2024) 103523, http://dx.doi.org/10.1016/j.cose.2023.103523, arXiv:2104.01026.

[11] Z. Zhang, J. Jia, B. Wang, N.Z. Gong, Backdoor attacks to graph neural networks, 2021, pp. 15–26, http://dx.doi.org/10.1145/3450569.3463560, arXiv:2006.11165.

[12] Z. Zhang, G. Xiao, Y. Li, T. Lv, F. Qi, Z. Liu, Y. Wang, X. Jiang, M. Sun, Red alarm for pre-trained models: Universal vulnerability to neuron-level backdoor attacks, Mach. Intell. Res. 20 (2) (2023) 180–193, http://dx.doi.org/10.1007/s11633-022-1377-5, arXiv:2101.06969. URL https://link.springer.com/article/10.1007/s11633-022-1377-5.

[13] A. Saha, A. Subramanya, H. Pirsiavash, Hidden trigger backdoor attacks, in: AAAI 2020 - 34th AAAI Conference on Artificial Intelligence, no. 07 SE - AAAI Technical Track: Vision, 2020, pp. 11957–11965, http://dx.doi.org/10.1609/aaai.v34i07.6871, arXiv:1910.00033. URL https://ojs.aaai.org/index.php/AAAI/article/view/6871.

[14] J. Chen, L. Zhang, H. Zheng, X. Wang, Z. Ming, DeepPoison: Feature transfer based stealthy poisoning attack for DNNs, IEEE Trans. Circuits Syst. II 68 (7) (2021) 2618–2622, http://dx.doi.org/10.1109/TCSII.2021.3060896.

[15] R. Pang, Z. Zhang, X. Gao, Z. Xi, S. Ji, P. Cheng, X. Luo, T. Wang, TrojanZoo: Towards unified, holistic, and practical evaluation of neural backdoors, in: Proceedings - 7th IEEE European Symposium on Security and Privacy, Euro S and P 2022, 2022, pp. 684–702, http://dx.doi.org/10.1109/EuroSP53844.2022.00048, arXiv:2012.09302.

[16] H. Qiu, Y. Zeng, S. Guo, T. Zhang, M. Qiu, B. Thuraisingham, DeepSweep: An evaluation framework for mitigating DNN backdoor attacks using data augmentation, in: ASIA CCS 2021 - Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security, ACM, 2021, pp. 363–377, http://dx.doi.org/10.1145/3433210.3453108, arXiv:2012.07006.

[17] Y. Jin, X. Zhang, J. Lou, X. Chen, ACQ: Few-shot backdoor defense via activation clipping and quantizing, in: MM 2023 - Proceedings of the 31st ACM International Conference on Multimedia, Association for Computing Machinery (ACM), 2023, pp. 5410–5418, http://dx.doi.org/10.1145/3581783.3612410, URL https://dl.acm.org/doi/10.1145/3581783.3612410.

[18] Y. Gao, C. Xu, D. Wang, S. Chen, D.C. Ranasinghe, S. Nepal, StriP: A defence against trojan attacks on deep neural networks, in: ACM International Conference Proceeding Series, 2019, pp. 113–125, http://dx.doi.org/10.1145/3359789.3359790, arXiv:1902.06531.

[19] K. Sikka, I. Sur, A. Roy, A. Divakaran, S. Jha, Detecting trojaned DNNs using counterfactual attributions, 2023, pp. 76–85, http://dx.doi.org/10.1109/ICAA58325.2023.00019, arXiv:2012.02275. URL http://arxiv.org/abs/2012.02275.

[20] H. Chen, C. Fu, J. Zhao, F. Koushanfar, Deepinspect: A black-box trojan detection and mitigation framework for deep neural networks, in: IJCAI International Joint Conference on Artificial Intelligence, 2019, pp. 4658–4664, http://dx.doi.org/10.24963/ijcai.2019/647.

[21] R. Ying, D. Bourgeois, J. You, M. Zitnik, J. Leskovec, GNNExplainer: Generating explanations for graph neural networks, in: Advances in Neural Information Processing Systems, Vol. 32, 2019, arXiv:1903.03894.

[22] R. Zheng, R. Tang, J. Li, L. Liu, Data-free backdoor removal based on channel Lipschitzness, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 13665 LNCS, Springer Science and Business Media Deutschland GmbH, 2022, pp. 175–191, http://dx.doi.org/10.1007/978-3-031-20065-6_11, arXiv:2208.03111. https://link.springer.com/chapter/10.1007/978-3-031-20065-6_11https://github.com/rkteddy/channel-Lipschitzness-based-pruning.

[23] G. Tao, Y. Liu, G. Shen, Q. Xu, S. An, Z. Zhang, X. Zhang, Model orthogonalization: Class distance hardening in neural networks for better security, in: Proceedings - IEEE Symposium on Security and Privacy, Vol. 2022-May, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 1372–1389, http://dx.doi.org/10.1109/SP46214.2022.9833688.

[24] X. Li, Z. Xiang, D.J. Miller, G. Kesidis, Backdoor mitigation by correcting the distribution of neural activations, 2023, arXiv:2308.09850. URL http://arxiv.org/abs/2308.09850.

[25] M. Weber, X. Xu, B. Karlaš, C. Zhang, B. Li, RAB: Provable robustness against backdoor attacks, 2023, pp. 1311–1328, http://dx.doi.org/10.1109/SP46215.2023.10179451, arXiv:2003.08904. URL http://arxiv.org/abs/2003.08904.

[26] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: 32nd International Conference on Machine Learning, ICML 2015, Vol. 1, International Machine Learning Society (IMLS), 2015, pp. 448–456.

[27] X. Chen, C. Liu, B. Li, K. Lu, D. Song, Targeted backdoor attacks on deep learning systems using data poisoning, 2017, arXiv:1712.05526. URL http://arxiv.org/abs/1712.05526.

[28] A. Krizhevsky, G. Hinton, Learning Multiple Layers of Features from Tiny Images, Cs.Toronto.Edu, 2009, pp. 1–58, URL http://www.cs.utoronto.ca/~kriz/learning-features-2009-TR.pdf.

[29] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2323, http://dx.doi.org/10.1109/5.726791.

[30] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778, URL www.image-net.org.

[31] T. Gu, B. Dolan-Gavitt, S. Garg, BadNets: Identifying vulnerabilities in the machine learning model supply chain, 2017, arXiv:1708.06733. URL http://arxiv.org/abs/1708.06733.

[32] C. Leys, C. Ley, O. Klein, P. Bernard, L. Licata, Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median, J. Exp. Soc. Psychol. 49 (4) (2013) 764–766, http://dx.doi.org/10.1016/j.jesp.2013.03.013.