TROJANZOO: Everything you ever wanted to know about neural backdoors (but were afraid to ask)

Ren Pang
Pennsylvania State University
Email: rbp5354@psu.edu

Zheng Zhang
Pennsylvania State University
Email: zxz147@psu.edu

Xiangshan Gao
Zhejiang University
Email: corazju@zju.edu.cn

Zhaohan Xi
Pennsylvania State University
Email: zxx5113@psu.edu

Shouling Ji
Zhejiang University
Email: sji@zju.edu.cn

Peng Cheng
Zhejiang University
Email: lunar_heart@zju.edu.cn

Ting Wang
Pennsylvania State University
Email: inbox.ting@gmail.com

***Abstract*—Neural backdoors represent one primary threat to the security of deep learning systems. The intensive research on this subject has produced a plethora of attacks/defenses, resulting in a constant arms race. However, due to the lack of evaluation benchmarks, many critical questions remain largely unexplored: (*i*) How effective, evasive, or transferable are different attacks? (*ii*) How robust, utility-preserving, or generic are different defenses? (*iii*) How do various factors (*e.g.*, model architectures) impact their performance? (*iv*) What are the best practices (*e.g.*, optimization strategies) to operate such attacks/defenses? (*v*) How can the existing attacks/defenses be further improved?**

To bridge the gap, we design and implement TROJANZOO, the first open-source platform for evaluating neural backdoor attacks/defenses in a unified, holistic, and practical manner. Thus, it has incorporated 12 representative attacks, 15 state-of-the-art defenses, 6 attack performance metrics, 10 defense utility metrics, as well as rich tools for in-depth analysis of attack-defense interactions. Leveraging TROJANZOO, we conduct a systematic study of existing attacks/defenses, leading to a number of interesting findings: (*i*) different attacks manifest various trade-offs among multiple desiderata (*e.g.*, effectiveness, evasiveness, and transferability); (*ii*) one-pixel triggers often suffice; (*iii*) optimizing trigger patterns and trojan models jointly improves both attack effectiveness and evasiveness; (*iv*) sanitizing trojan models often introduces new vulnerabilities; (*v*) most defenses are ineffective against adaptive attacks, but integrating complementary ones significantly enhances defense robustness. We envision that such findings will help users select the right defense solutions and facilitate future research on neural backdoors.

I. INTRODUCTION

Today's deep learning (DL) systems are large, complex software artifacts. With the increasing system complexity and training cost, it becomes not only tempting but also necessary to exploit pre-trained deep neural networks (DNNs) in building DL systems. It was estimated that as of 2016, over 13.7% of DL-related repositories on GitHub re-use at least one pre-trained DNN [27]. On the upside, this "plug-and-play" paradigm greatly simplifies the development cycles [50]. On the downside, as most pre-trained DNNs are contributed by untrusted third parties [7], their lack of standardization or regulation entails profound security implications.

In particular, pre-trained DNNs can be exploited to launch *neural backdoor* attacks [21], [40], [45], one immense threat to the security of DL systems. In such attacks, a maliciously crafted DNN ("trojan model") forces its host system to misbehave once certain pre-defined conditions ("triggers") are present but functions normally otherwise. Such attacks can result in consequential damages such as misleading autonomous vehicles to crashing [63], maneuvering video surveillance to miss illegal activities [13], and manipulating biometric authentication to allow improper access [6].

Motivated by this, intensive research has been conducted on neural backdoors, leading to a plethora of attacks that craft trojan model via exploiting various properties (*e.g.*, neural activation patterns) [21], [11], [40], [33], [56], [70] and defenses that mitigate trojan models during inspection [64], [39], [10], [23], [26], [37] or detect trigger inputs at inference [19], [9], [12], [62]. With the rapid development of new attacks/defenses, a number of open questions have emerged:

RQ$_1$ – *How effective, evasive, and transferable are the existing attacks?*

RQ$_2$ – *How robust, utility-preserving, and generic are the existing defenses?*

RQ$_3$ – *How do various factors (e.g., model architectures) impact the performance of different attacks/defenses?*

RQ$_4$ – *What are the best practices (e.g., optimization strategies) to operate different attacks/defenses?*

RQ$_5$ – *How can the existing backdoor attacks/defenses be further improved?*

Despite their importance for assessing and mitigating the vulnerabilities incurred by pre-trained DNNs, these questions are largely unexplored due to the following challenges.

Non-holistic evaluations – Most studies conduct evaluations with a limited set of attacks/defenses, resulting in incomplete comparison. For instance, it is unknown whether the STRIP defense [19] is effective against the newer ABE attack [32]. Further, the evaluations often use simple, macro-level metrics, failing to comprehensively characterize given attacks/defenses. For instance, most studies use attack success rate (*ASR*) and clean accuracy drop (*CAD*) to assess an attack's performance, yet insufficient to describe the attack's ability of trading off between the two metrics.

Non-unified platforms – Due to the lack of unified benchmarks, different attacks/defenses are often evaluated under varying configurations, leading to non-comparable conclusions. For instance, TNN [40] and LB [70] are evaluated with distinct trigger definitions (*i.e.*, shape, size, and transparency), datasets, and DNNs, making it difficult to directly compare their effectiveness and evasiveness.

Non-adaptive attacks – The evaluations of existing defenses (*e.g.*, [37], [64], [19], [23]) often assume static, non-adaptive attacks, without fully accounting for the adversary's possible countermeasures, which however is critical for modeling the adversary's optimal strategies and assessing the attack vulnerabilities in realistic settings.

Our Work

To this end, we design, implement, and evaluate TROJANZOO, an open-source platform for assessing neural backdoor attacks/defenses in a holistic, unified, and practical manner. Our contributions are summarized as follows.

Platform – To our best knowledge, TROJANZOO represents the first open-source platform designed for evaluating neural backdoor attacks/defenses. To date, TROJANZOO has incorporated 12 representative attacks, 15 state-of-the-art defenses, 6 attack performance metrics, 10 defense utility metrics, as well as a benchmark suite of 5 DNN models, 5 downstream models, and 6 datasets. All the modules (*e.g.*, attacks, defenses, and models) are provided as Dockerfiles for easy installation and deployment. Further, TROJANZOO implements utility tools for in-depth analysis of attack-defense interactions, including measuring feature-space similarity, tracing neural activation patterns, and comparing attribution maps.

Assessment – Leveraging TROJANZOO, we conduct a systematic study of existing attacks/defenses. The attacks are evaluated in terms of effectiveness, evasiveness, and transferability, while the defenses are assessed in terms of robustness, utility-preservation, and genericity. We make a number of interesting observations: (*i*) most attacks manifest strong "mutual-reinforcement" effects in the effectiveness-evasiveness trade-off; (*ii*) DNN architectures that enable better feature extraction may also allow more effective propagation of trigger patterns; (*iii*) more effective attacks (*e.g.*, higher *ASR*) are also more likely to be detected by input filtering (*e.g.*, [19]); (*iv*) weaker attacks (*i.e.*, lower *ASR*) demonstrate higher transferability than stronger ones; (*v*) model-inspection defenses (*e.g.*, [39]) often uncover backdoors non-identical to, but overlapping with, the ones injected by the adversary. Our evaluation unveils the strengths and limitations of existing attacks/defenses as well as their intricate interactions.

Exploration – We further explore improving existing attacks/defenses, leading to a set of previously unknown findings such as (*i*) one-pixel triggers suffice (over 95% *ASR*) for many attacks; (*ii*) training from scratch seems more effective than re-training benign models to forge trojan models; (*iii*) leveraging DNN architectures (*e.g.*, skip connection) in optimizing trojan models marginally improves attack effectiveness; (*iv*) optimizing trigger patterns along with trojan models improves both attack effectiveness and evasiveness; (*iv*) sanitizing trojan models via unlearning [64], while fixing existing backdoors, may introduce new vulnerabilities; (*v*) while most defenses are vulnerable to adaptive attacks, integrating complementary ones (*e.g.*, model inspection and input filtering) significantly enhances defense robustness. We envision that our findings will facilitate future research on neural backdoors and shed light on designing and building DL systems in a more secure and informative manner.[1]

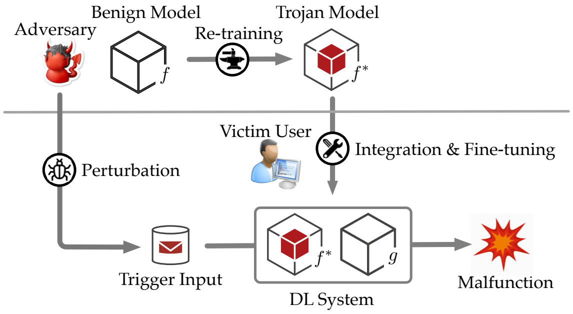

Figure 1: Illustration of neural backdoor attacks.

II. FUNDAMENTALS

In this section, we introduce the fundamental concepts and assumptions used throughout the paper. The important symbols and notations are summarized as Table 20 in Appendix A.

A. *Preliminaries*

Deep neural networks (DNNs) – DNNs represent a class of machine learning models to learn high-level abstractions of complex data using multiple processing layers in conjunction with non-linear transformations. In a predictive task, a DNN encodes a function $f: \mathcal{X} \rightarrow \mathcal{Y}$, which, given an input $x \in \mathcal{X}$, predicts $f(x)$ ranging over a set of pre-defined classes $\mathcal{Y}$.

Pre-trained DNNs – Today, it becomes not only tempting but also necessary to reuse pre-trained DNNs in domains in which data labeling or model training is expensive [72]. Under the transfer learning setting, as shown in Figure 1, a pre-trained DNN f is composed with a downstream classifier/regressor g to form an end-to-end system. As the data used to train f may differ from the downstream task, it is often necessary to fine-tune the system $g \circ f$ in a supervised manner. The user may opt to perform full-tuning to train both f and g or partial-tuning to train g only with f fixed [27].

Neural backdoor attacks – With the increasing use of pre-trained models in security-critical domains [27], the adversary is strongly incentivized to forge malicious DNNs ("trojan models") as attack vectors and lure victim users to re-use them during either system development or update [21].

Specifically, through trojan models, backdoor attacks infect target systems with malicious functions desired by the adversary, which are activated once pre-defined conditions ("triggers") are present. Typically, a trojan model reacts to trigger-embedded inputs (*e.g.*, images with specific watermarks) in a highly predictable manner (*e.g.*, misclassified to a target class) but functions normally otherwise.

[1] All the data, models, and code of the paper are open-sourced at https://github.com/ain-soph/trojanzoo

B. Specifics

Trigger embedding operator – The operator $\oplus$ mixes a clean input $x \in \mathbb{R}^n$ with the trigger r to produce a trigger input $x \oplus r$. Typically, r consists of three parts: (*i*) *mask* $m \in \{0,1\}^n$ specifies where r is applied (*i.e.*, x's i-th feature x_i is retained if m_i is on and mixed with r otherwise); (*ii*) *transparency* $\alpha \in [0,1]$ specifies the mixing weight; and (*iii*) *pattern* $p(x) \in \mathbb{R}^n$ specifies r's color intensity. Here, $p(x)$ can be a constant, randomly drawn from a distribution (*e.g.*, by perturbing a template), or dependent on x [48]. Formally, the trigger embedding operator is defined as:

$$x \oplus r = (1-m) \odot [(1-\alpha)x + \alpha p(x)] + m \odot x \tag{1}$$

where $\odot$ denotes element-wise multiplication.

Attack objectives – The trojan model f satisfies that (*i*) each clean input $x \in \mathcal{T}$ is correctly classified, where $\mathcal{T}$ is a reference set, while (*ii*) each trigger input $x \oplus r$ for $x \in \mathcal{T}$ is misclassified to the target class t. Formally, the adversary optimizes the following objective function:

$$\min_{r \in \mathcal{R}_\epsilon, f \in \mathcal{F}_\delta} \mathbb{E}_{x \in \mathcal{T}} \left[\ell(f(x \oplus r), t)\right] \tag{2}$$

where the loss function ℓ measures the quality of the model output $f(x \oplus r)$ with respect to t, trojan model f and trigger r are selected from the feasible sets $\mathcal{R}_\epsilon$ and $\mathcal{F}_\delta$ respectively, which are detailed below.

Loss function ℓ – If the downstream classifier g is known to the adversary, ℓ is defined as the difference (*e.g.*, cross entropy) of the prediction $g \circ f(x \oplus r)$ and t; otherwise, the adversary may resort to a surrogate model g^* or define ℓ in terms of latent representations [70], [45] (*e.g.*, the difference of $f(x \oplus r)$ and ϕ_t, where ϕ_t is the latent representation of class t). Essentially, ℓ quantifies *efficacy*, whether the attack successfully forces the target system to misclassify each trigger input $x \oplus r$ to t.

Feasible set $\mathcal{R}_\epsilon$ – To maximize its evasiveness, trigger r is often constrained in terms of its shape, position, and pattern, which can be defined as a feasible set $\mathcal{R}_\epsilon$ parameterized by ϵ (*e.g.*, threshold on r's transparency). Essentially, $\mathcal{R}_\epsilon$ quantifies *fidelity*, whether the attack retains the perceptual similarity of clean and trigger inputs.

Feasible set $\mathcal{F}_\delta$ – To optimize its evasiveness, trojan model f is also selected from a feasible set $\mathcal{F}_\delta$, limiting f's impact on clean inputs. For instance, $\mathcal{F}_\delta = \{f \,|\, \mathbb{E}_{x \in \mathcal{T}}[|f^*(x) - f(x)|] \leq \delta\}$ ensures that the expected difference of f^*'s and f's outputs is bounded by δ. Essentially, $\mathcal{F}_\delta$ quantifies *specificity*, whether the attack directs its influence to trigger inputs only.

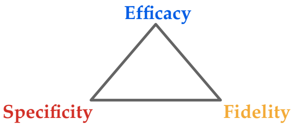

Figure 2: Multiple objectives of neural backdoor attacks.

Interestingly, as illustrated in Figure 2, the three objectives are tightly intertwined and may trade one for another (*e.g.*, balancing the evasiveness of r and f^*).

Trojan model training – To optimize Eq. 2, one may add trigger inputs to the training set to re-train a benign model [56], [52], [40], or directly perturb the benign model and preform further tuning using clean data [27]. To satisfy the trigger constraint, r can be fixed [21], partially defined [40] (*e.g.*, with its mask fixed), or optimized with f jointly [45]. To satisfy the model constraint, in addition to the loss in Eq. 2, one may add the loss with respect to clean inputs, $\mathbb{E}_{(x,y) \in \mathcal{T}}[\ell(f(x), y)]$, where (x, y) represents a clean input-class pair.

III. Platform

At a high level, TrojanZoo consists of three main components as illustrated in Figure 3: (*i*) the attack library that implements representative attacks, (*ii*) the defense library that integrates state-of-the-art defenses, and (*iii*) the analysis engine that, equipped with attack performance metrics, defense utility metrics, and feature-rich utility tools, is able to conduct unified and holistic evaluations across various attacks/defenses,

In its current implementation, TrojanZoo incorporates 12 attacks, 15 defenses, 6 attack performance metrics, and 10 defense utility metrics, which we systemize as follows.

A. Attacks

While neural backdoor attacks can be characterized by a range of aspects, here we focus on five key design choices by the adversary that directly impact attack performance.

- *Architecture modifiability* – whether the attack is able to change the DNN architecture. Being allowed to modify both the architecture and the parameters enables a larger attack spectrum, but also renders the trojan model more susceptible to certain defenses (*e.g.*, model specification checking).
- *Trigger optimizability* – whether the attack uses a fixed, predefined trigger or optimizes it during crafting the trojan model. Trigger optimization often leads to stronger attacks with respect to given desiderata (*e.g.*, trigger stealthiness).
- *Training controllability* – whether the adversary has control over the training of trojan models. Under the setting that the victim user controls the model training, the adversary may influence the training only through injecting poisoning data or compromising the training code.
- *Fine-tuning survivability* – whether the backdoor remains effective if the model is fine-tuned. A pre-trained model is often composed with a classifier and fine-tuned using the data from the downstream task. It is desirable to ensure that the backdoor remains effective after fine-tuning.
- *Defense adaptivity* – whether the attack is optimizable to evade possible defenses. For the attack to be effective, it is essential to optimize the evasiveness of the trojan model and the trigger input with respect to the deployed defenses.

Table 1 summarizes the representative neural backdoor attacks currently implemented in TrojanZoo, which are characterized along the above five dimensions.

Non-optimization – BN [21], as the simplest attack, predefines a trigger r (*i.e.*, shape, position, and pattern), generates

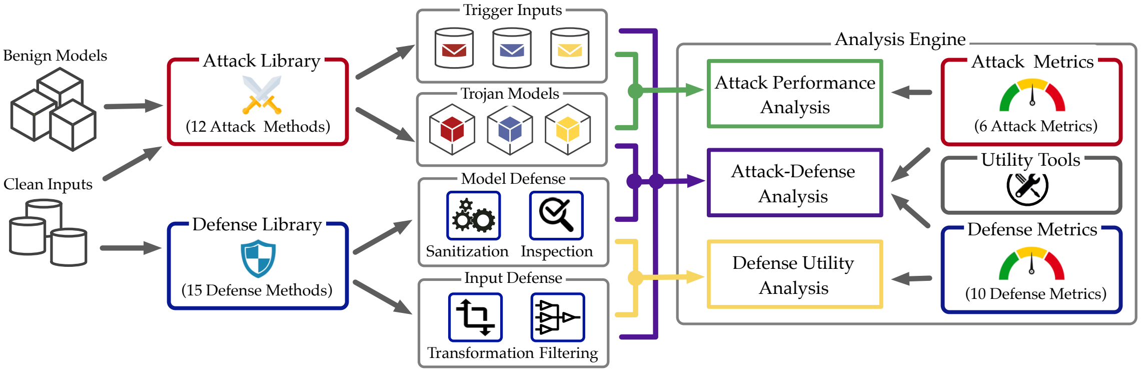

Figure 3: System design of TrojanZoo.

Neural Backdoor Attack	Architecture Modifiability	Trigger Optimizability	Training Controllability	Fine-tuning Survivability	Defense Adaptivity
BadNet (BN) [21]	○	○	○	○	○
Embarrassingly Simple Backdoor (ESB) [58]	●	○	○	○	○
TrojanNN (TNN) [40]	○	◐	○	○	○
Reflection Backdoor (RB) [41]	○	◐	○	○	○
Targeted Backdoor (TB) [11]	○	◐	○	○	○
Dynamic Backdoor (DB) [48]	○	●	○	○	○
Clean-Label Backdoor (CLB) [61]	○	○	●	○	○
Hidden Trigger Backdoor (HTB) [47]	○	○	●	○	○
Blind Backdoor (BB) [4]	○	○	●	○	○
Latent Backdoor (LB) [70]	○	○	○	●	○
Adversarial Backdoor Embedding (ABE) [32]	○	○	○	○	●
Input-Model Co-optimization (IMC) [45]	○	●	○	●	●

Table 1. Summary of representative neural backdoor attacks currently implemented in TrojanZoo (● – full optimization, ◐ – partial optimization, ○ – no optimization)

trigger inputs $\{(x \oplus r, t)\}$, and forges the trojan model f^* by re-training a benign model f with such data.

Architecture modifiability – ESB [58] modifies f's architecture by adding a module which overwrites the prediction as t if r is recognized. Without disturbing f's original configuration, f^* retains f's predictive power on clean inputs.

Trigger optimizability – TNN [40] fixes r's shape and position, optimizes its pattern to activate neurons rarely activated by clean inputs in pre-processing, and then forges f^* by re-training f in a manner similar to BN.

RB [41] optimizes trigger stealthiness by defining r as the physical reflection of a clean image x^r (selected from a pool): $r = x^r \otimes k$, where k is a convolution kernel, and $\otimes$ is the convolution operator.

TB [11] randomly generates r's position in training, which makes f^* effective regardless of r's position and allows the adversary to optimize r's stealthiness by placing it at the most plausible position (*e.g.*, an eyewear watermark over eyes).

DB [48] uses a generative network to generate r's pattern and position dynamically, which is trained jointly with f^*.

Training controllability – CLB [61] assumes the setting that the adversary forges f^* via polluting the training data. To evade possible filtering, CLB generates (via either adversarial perturbation or generative networks) stealthy poisoning inputs that appear to be consistent with their labels.

HTB [47] generates stealthy poisoning inputs that are close to trigger inputs in the feature space, but are correctly labeled (to human inspection) and do not contain visible triggers.

BB [4] assumes the setting that the adversary forges f^* via compromising the code of computing the loss function.

Fine-tuning survivability – LB [70] accounts for the impact of downstream fine-tuning by optimizing f with respect to latent representations rather than final predictions. Specifically, it instantiates Eq. 2 with the following loss function: $\ell(f(x \oplus r), t) = \Delta(f(x \oplus r), \phi_t)$, where Δ measures the difference of two latent representations and ϕ_t denotes the representation of class t, defined as $\phi_t = \arg\min_\phi \mathbb{E}_{(x,t)\in\mathcal{T}}[\Delta(f(x), \phi_t)]$.

Defense adaptivity – ABE [32] accounts for possible defenses in forging f^*. In solving Eq. 2, ABE also optimizes the indistinguishability of the latent representations of trigger and clean inputs. Specifically, it uses a discriminative network d to predict the representation of a given input x as trigger or clean. Formally, the loss is defined as $\Delta(d \circ f(x), b(x))$, where $b(x)$ encodes whether x is trigger or clean, while f^* and d are trained using an adversarial learning framework [20].

Co-optimization – IMC [45] is motivated by the mutual-reinforcement effect between r and f^*: optimizing one may greatly amplify the effectiveness of the other. Instead of solving Eq. 2 by first pre-defining r and then optimizing f^*, IMC optimizes r and f^* jointly, which enlarges the search spaces for r and f^*, leading to attacks satisfying multiple desiderata (*e.g.*, fine-tuning survivability and defense adaptivity).

B. Attack Performance Metrics

Currently, TrojanZoo incorporates 6 metrics to assess the effectiveness, evasiveness, and transferability of given attacks.

Neural Backdoor Defense	Category	Mitigation		Detection Target			Design Rationale
		Input	Model	Input	Model	Trigger	
Randomized-Smoothing (Rs) [12]	Input Reformation	✓					$\mathcal{A}$'s fidelity (x's and x^*'s surrounding class boundaries)
Down-Upsampling (Du) [68]		✓					$\mathcal{A}$'s fidelity (x's and x^*'s high-level features)
Manifold-Projection (Mp) [43]		✓					$\mathcal{A}$'s fidelity (x's and x^*'s manifold projections)
Activation-Clustering (Ac) [9]	Input Filtering			✓			distinct activation patterns of $\{x\}$ and $\{x^*\}$
Spectral-Signature (Ss) [60]				✓			distinct activation patterns of $\{x\}$ and $\{x^*\}$ (spectral space)
STRIP (Strip) [19]				✓			distinct self-entropy of x's and x^*'s mixtures with clean inputs
NEO (Neo) [62]				✓			sensitivity of f^*'s prediction to trigger perturbation
Februus (Februus) [15]				✓		✓	sensitivity of f^*'s prediction to trigger perturbation
Adversarial-Retraining (Ar) [42]	Model Sanitization		✓				$\mathcal{A}$'s fidelity (x's and x^*'s surrounding class boundaries)
Fine-Pruning (Fp) [37]			✓				$\mathcal{A}$'s use of neurons rarely activated by clean inputs
NeuralCleanse (Nc) [64]	Model Inpsection				✓	✓	abnormally small perturbation from other classes to t in f
DeepInspect (Di) [10]					✓	✓	abnormally small perturbation from other classes to t in f^*
TABOR (Tabor) [23]					✓	✓	abnormally small perturbation from other classes to t in f
NeuronInspect (Ni) [26]					✓		distinct explanations of f and f^* with respect to clean inputs
ABS (Abs) [39]					✓	✓	$\mathcal{A}$'s use of neurons elevating t's prediction

Table 2. Summary of representative neural backdoor defenses currently implemented in TrojanZoo ($\mathcal{A}$ – backdoor attack, x – clean input, x^* – trigger input, f – benign model, f^* – trojan model, t – target class)

Attack success rate (*ASR*) – which measures the likelihood that trigger inputs are classified to the target class t:

$$\mathit{ASR} = \frac{\text{\# successful trials}}{\text{\# total trials}} \tag{3}$$

Typically, higher *ASR* indicates more effective attacks.

Trojan misclassification confidence (*TMC*) – which is the average confidence score assigned to class t of trigger inputs in successful attacks. Intuitively, *TMC* complements *ASR* and measures attack efficacy from another perspective.

Clean accuracy drop (*CAD*) – which measures the difference of the classification accuracy of two systems built upon the benign model and its trojan counterpart, respectively; *CAD* measures the attack specificity (*cf.* Figure 2), that is, whether the attack directs its influence to trigger inputs only.

Clean classification confidence (*CCC*) – which is the average confidence assigned to the ground-truth classes of clean inputs; *CCC* complements *CAD* by measuring attack specificity from the perspective of classification confidence.

Efficacy-specificity AUC (*AUC*) – which quantifies the aggregated trade-off between attack efficacy (measured by *ASR*) and attack specificity (measured by *CAD*). As revealed in [45], there exists an intricate balance: at a proper cost of specificity, it is possible to significantly improve efficacy, and vice versa; *AUC* measures the area under the *ASR*-*CAD* curve. Intuitively, smaller *AUC* implies a more significant trade-off effect.

Neuron-separation ratio (*NSR*) – which measures the intersection between neurons activated by clean and trigger inputs. In the penultimate layer of the model, we find $\mathcal{N}_\text{c}$ and $\mathcal{N}_\text{t}$, the top-k active neurons with respect to clean and trigger inputs, respectively, and calculate their jaccard index:

$$\mathit{NSR} = 1 - |\mathcal{N}_\text{t} \cap \mathcal{N}_\text{c}| / |\mathcal{N}_\text{t} \cup \mathcal{N}_\text{c}| \tag{4}$$

Intuitively, *NSR* compares the neural activation patterns of clean and trigger inputs.

C. Defenses

The existing defenses against neural backdoor attacks, according to their strategies, can be categorized as:

- *Input reformation* – which, before feeding an incoming input to the system, first reforms it to mitigate the influence of the potential trigger, yet without explicitly detecting whether it is a trigger input. It typically exploits the high fidelity of an attack $\mathcal{A}$, that is, $\mathcal{A}$ tends to retain the perceptual similarity of a clean input x and its trigger counterpart x^*.
- *Input filtering* – which detects whether an incoming input is embedded with a trigger and possibly recovers the clean input. It typically distinguishes clean and trigger inputs using their distinct characteristics.
- *Model sanitization* – which, before using a pre-trained model f, sanitizes it to mitigate the potential backdoor, yet without explicitly detecting whether f is trojaned.
- *Model inspection* – which determines whether f is a trojan model and, if so, recovers the target class and the potential trigger, at the model checking stage.

Note that here we focus on the setting of transfer learning or outsourced training, which precludes certain other defenses such as purging poisoning training data [55]. Table 2 summarizes the 15 representative defenses currently implemented in TrojanZoo, which are detailed below.

Input reformation – Rs [12] exploits the premise that $\mathcal{A}$ retains the similarity of x and x^* in terms of their surrounding class boundaries and classifies an input by averaging the predictions within its vicinity (via adding Gaussian noise).

Du [68] exploits the premise that $\mathcal{A}$ retains the similarity of x and x^* in terms of their high-level features while the trigger r is typically not perturbation-tolerant. By downsampling and then upsampling x^*, it is possible to mitigate r's influence.

Mp [43] exploits the premise that $\mathcal{A}$ retains the similarity of x and x^* in terms of their projections to the data manifold. To this end, it trains an autoencoder to learn an approximate manifold, which projects x^* to the manifold.

Input filtering – Ac [9] distinguishes clean and trigger inputs by clustering their latent representations. While Ac is also applicable for purging poisoning data, we consider its use as an input filtering method at inference time. Ss [60] exploits the similar property in the spectral space.

Strip [19] mixes a given input with a clean input and measures the self-entropy of its prediction. If the input is trigger-embedded, the mixture remains dominated by the trigger and tends to be misclassified, resulting in low self-entropy.

Neo [62] detects a trigger input by searching for a position, if replaced by a "blocker", changes its prediction, and uses this substitution to recover its original prediction. Februus [15] exploits the same property but uses a generative network to generate the substitution blocker.

Model sanitization – By treating trigger inputs as one type of adversarial inputs, Ar [42] applies adversarial training over the pre-trained model to improves its robustness to backdoor attacks. Fp [37] uses the property that the attack exploits spare model capacity. It thus prunes rarely used neurons and then applies fine-tuning to defend against pruning-aware attacks.

Model inspection – Given a model f, Nc [64] searches for potential triggers in each class t. If t is trigger-embedded, the minimum perturbation required to change the predictions of the inputs in other classes to t is abnormally small. Di [10] follows a similar pipeline but uses a generative network to generate trigger candidates. Tabor [23] extends Nc by adding a new regularizer to control the trigger search space.

Ni [26] exploits the property that the explanation heatmaps of benign and trojan models manifest distinct characteristics. Using the features extracted from such heatmaps, Ni detects trojan models as outliers.

Abs [39] first inspects f to sift out abnormal neurons with large elevation difference (*i.e.*, active only with respect to one specific class) and identifies triggers by maximizing abnormal neuron activation while preserving normal neuron behaviors.

D. Defense Utility Metrics

Currently, TrojanZoo incorporates 10 metrics to evaluate the robustness, utility-preservation, and genericity of given defenses. The metrics are tailored to the objectives of each defense category (*e.g.*, trigger input detection). For ease of exposition, below we consider the performance of a given defense $\mathcal{D}$ with respect to a given attack $\mathcal{A}$.

Attack rate deduction (*ARD*) – which measures the difference of $\mathcal{A}$'s *ASR* before and after $\mathcal{D}$. Intuitively, *ARD* indicates $\mathcal{D}$'s impact on $\mathcal{A}$'s efficacy. Intuitively, larger *ARD* indicates more effective defense. We also use $\mathcal{A}$'s *TMC* to measure $\mathcal{D}$'s influence on the classification confidence of trigger inputs.

Clean accuracy drop (*CAD*) – which measures the difference of the *ACC* of clean inputs before and after $\mathcal{D}$ is applied. It measures $\mathcal{D}$'s impact on the system's normal functionality. Note that *CAD* here is defined differently from its counterpart in attack performance metrics. We also use *CCC* to measure $\mathcal{D}$'s influence on the classification confidence of clean inputs.

True positive rate (*TPR*) – which, for input-filtering methods, measures the performance of detecting trigger inputs.

$$\mathit{TPR} = \frac{\text{\# successfully detected trigger inputs}}{\text{\# total trigger inputs}} \tag{5}$$

Correspondingly, we use false positive rate (*FPR*) to measure the error of misclassifying clean inputs as trigger inputs.

Anomaly index value (*AIV*) – which measures the anomaly of trojan models in model-inspection defenses. Most existing methods (*e.g.*, [64], [10], [23], [39]) formalize finding trojan models as outlier detection: each class t is associated with a score (*e.g.*, minimum perturbation); if its score significantly deviates from others, t is considered to contain a backdoor. *AIV*, the absolute deviations from median normalized by median absolute deviation (*MAD*), provide a reliable measure for such dispersion. Typically, t with *AIV* larger than 2 has over 95% probability of being anomaly.

Mask L_1 norm (*MLN*) – which measures the ℓ_1-norm of the triggers recovered by model-inspection methods.

Mask jaccard similarity (*MJS*) – which further measures the intersection between the recovered trigger and the ground-truth trigger (injected by the adversary). Let m^o and m^r be the masks of original and recovered triggers. We define *MJS* as the Jaccard similarity of m^o and m^r :

$$\mathit{MJS} = |O(m^o) \cap O(m^r)| / |O(m^o) \cup O(m^r)| \tag{6}$$

where $O(m)$ denotes the set of non-zero elements in m.

Average running time (*ART*) – which measures the overhead of $\mathcal{D}$. For model sanitization or inspection, which is performed offline, *ART* is measured as the running time per model; while for input filtering or reformation, which is executed online, *ART* is measured as the execution time per input.

IV. Assessment

Equipped with TrojanZoo, we conduct a systematic assessment of the existing attacks and defenses, in which the attacks are evaluated in terms of effectiveness, evasiveness, and transferability, and the defenses are evaluated in terms of robustness, utility-preservation, and genericity.

A. Experimental Setting

Dataset	# Class	# Dimension	Model	Accuracy
CIFAR10	10	32×32	ResNet18	95.37%
			DenseNet121	93.84%
			VGG13	92.44%
CIFAR100	100	32×32	ResNet18	73.97%
GTSRB	43	32×32		98.18%
ImageNet-mini	10	224×224		92.40%
VGGFace2-mini	20	224×224		90.77%

Table 3. *ACC* of systems built upon benign, pre-trained models.

Datasets – In the evaluation, we primarily use 5 datasets: CIFAR10 [30], CIFAR100 [30], ImageNet [14], GTSRB [54], and VGGFace2 [8]. Their statistics are summarized in Table 3. By default, we partition each dataset into 40%/40%/20% for pre-training, fine-tuning, and testing respectively.

Models – We consider 3 representative DNNs: VGG13 [53], ResNet18 [24], and DenseNet121 [25]. Using models of distinct architectures (*e.g.*, residual block), we factor out the influence of individual model characteristics. By default, we assume a downstream model comprising one fully-connected layer with softmax activation (1Fcn). We also consider other types of models, including Bayes, SVM, and Random Forest. The *ACC* of systems built upon benign, pre-trained models is summarized in Table 3.

Attacks, Defenses, and Metrics – In the evaluation, we exemplify with 8 attacks in Table 1 and 12 defenses in Table 2, and measure them using all the metrics in § III-B and § III-D. In all the experiments, we generate 10 trojan models for a given attack under each setting and 100 pairs of clean-trigger inputs with respect to each trojan model. The reported results are averaged over these cases.

Implementation – All the models, algorithms, and measurements are implemented in PyTorch. All the experiments are conducted on a Linux server with Quodro RTX 6000 GPU, Intel Xeon processor, and 384G RAM. The default parameter setting is summarized as Table 21 and 22 in Appendix A.

B. Attack Performance Evaluation

We first evaluate existing attacks on vanilla systems (without defenses), aiming to understand the impact of various design choices and context settings on attack performance. Due to space limitations, we mainly report the results on CIFAR10 and defer the results on other datasets to Appendix B.

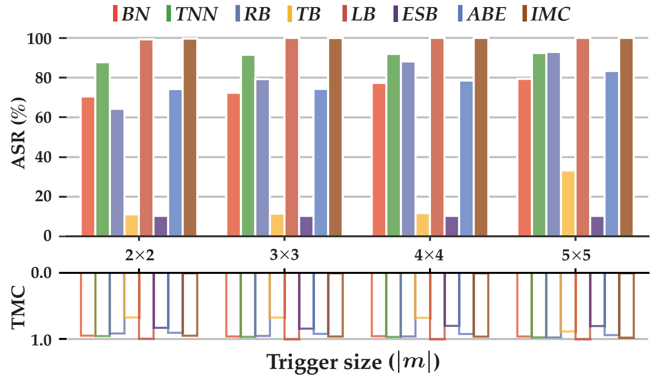

Figure 4: *ASR* and *TMC* with respect to trigger size ($\alpha = 0.8$, CIFAR10).

Trigger size – Recall that the trigger definition consists of mask m, transparency α, and pattern ϕ. Here, we measure how the attack efficacy varies with the trigger size $|m|$. To make fair comparison, we bound the clean accuracy drop (*CAD*) of all the attacks below 3% via controlling the number of optimization iterations n_{iter}. Figure 4 plots the attack success rate (*ASR*) and trojan misclassification confidence (*TMC*) of various attacks under varying $|m|$ on CIFAR10 (with fixed $\alpha = 0.8$).

Observe that most attacks seem insensitive to $|m|$: as $|m|$ varies from 2×2 to 5×5, the *ASR* of most attacks increases by less than 10%, except Rb and Tb, with over 30% growth. This may be attributed to their additional constraints: Rb defines the trigger to be the reflection of another image, while Tb requires the trigger to be positionless). Thus, increasing $|m|$ may improve their perturbation spaces. Also observe that the *TMC* of most attacks remains close to 1.0 regardless of $|m|$.

Remark 1 – *Trigger-size has a limited impact on attack efficacy, except for attacks with additional trigger constraints.*

Trigger transparency – Under the same setting, we further evaluate the impact of trigger transparency α. Figure 5 plots the *ASR* of different attacks as a function of α on CIFAR10 (with fixed $|m|$ = 3×3).

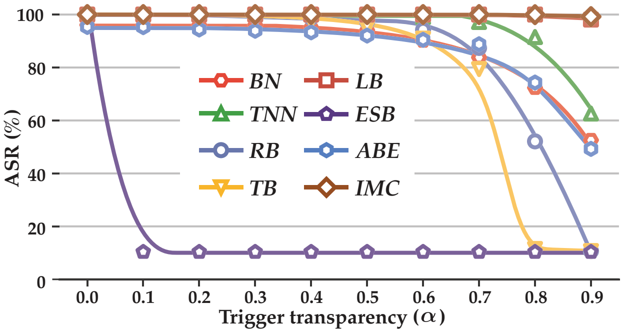

Figure 5: *ASR* with respect to trigger transparency ($|m|$ = 3×3, CIFAR10).

Compared with trigger size, α has a more profound impact. The *ASR* of most attacks drops sharply once α exceeds 0.6, among which Tb approaches 10% if $\alpha \geq 0.8$, and Esb works only if α is close to 0, due to its reliance on recognizing the trigger precisely to overwrite the model prediction. Meanwhile, Lb and Imc seem insensitive to α. This may be attributed to that Lb optimizes trojan models with respect to latent representations (rather than final predictions), while Imc optimizes trigger patterns and trojan models jointly. Both strategies may mitigate α's impact.

Remark 2 – *It requires to exploit alternative optimization strategies to attain effective attacks under high trigger-transparency.*

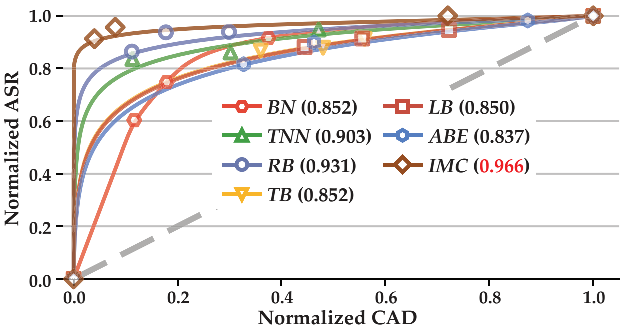

Figure 6: Attack efficacy-specificity trade-off on CIFAR10 ($|m| = 3 \times 3$, $\alpha = 0.8$)

Efficacy-specificity trade-off – One intriguing property of attacks is the trade-off between maximizing the effectiveness with respect to trigger inputs (efficacy) and minimizing the influence over clean inputs (specificity). Here, we characterize the efficacy-specificity trade-off via varying the fraction of trigger inputs in training data. For each attack, we bound its *CAD* within 3%, measure its highest and lowest *ASR* (which corresponds to its lowest and highest *CAD* respectively), and then normalize the *ASR* and *CAD* measures to $[0, 1]$.

Figure 6 visualizes the normalized *CAD-ASR* trade-off. Observe that the curves of all the attacks manifest strong convexity, indicating the "leverage" effects [45]: it is practical to greatly improve *ASR* at a disproportionally small cost of *CAD*. Also observe that different attacks feature varying Area Under the Curve (*AUC*). Intuitively, a smaller *AUC* implies a stronger leverage effect. Among all the attacks, Imc shows the smallest

AUC. This may be explained by that Imc uses the trigger-model co-optimization framework, which allows the adversary to maximally optimize *ASR* at given *CAD*.

Remark 3 – *All the attacks demonstrate strong "leverage" effects in the efficacy-specificity trade-offs.*

Attack	CIFAR10	CIFAR100	ImageNet	
	$\vert m\vert=3,\ \alpha=0.8$	$\vert m\vert=3,\ \alpha=0.8$	$\vert m\vert=3,\ \alpha=0$	$\vert m\vert=7,\ \alpha=0.8$
Bn	72.4 (0.96)	64.5 (0.96)	90.0 (0.98)	11.4 (0.56)
Tnn	91.5 (0.97)	89.8 (0.98)	95.2 (0.99)	11.6 (0.62)
Rb	52.1 (1.0)	42.8 (0.95)	94.6 (0.98)	11.2 (0.59)
Tb	11.5 (0.66)	23.4 (0.75)	82.8 (0.97)	11.4 (0.58)
Lb	100.0 (1.0)	97.8 (0.99)	97.4 (0.99)	11.4 (0.59)
Esb	10.3 (0.43)	1.0 (0.72)	100.0 (0.50)	N/A
Abe	74.3 (0.91)	67.9 (0.96)	82.6 (0.97)	12.00 (0.50)
Imc	100.0 (1.0)	98.8 (0.99)	98.4 (1.0)	96.6 (0.99)

Table 4. Impact of data complexity on *ASR* and *TMC* of various attacks.

Data complexity – To assess the impact of data complexity, we compare the *ASR* and *TMC* of existing attacks on different datasets, with results in Table 4 (more results in Table 23).

We observe that the class-space size (the number of classes) negatively affects the attack efficacy. For example, the *ASR* of Bn drops by 7.9% from CIFAR10 to CIFAR100. Intuitively, it is more difficult to force trigger inputs from all the classes to be misclassified in a larger output-space. Moreover, it tends to require more significant triggers to achieve comparable attack performance on more complex data. For instance, for Imc to attain similar *ASR* on CIFAR10 and ImageNet, it needs to either increase trigger size (from 3×3 to 7×7) or reduce trigger transparency (from 0.8 to 0.0).

Remark 4 – *To attain comparable attack efficacy on more complex data requires more significant triggers.*

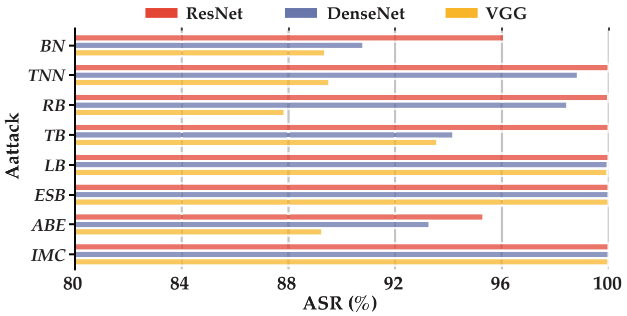

Figure 7: Impact of DNN architecture on attack efficacy.

DNN architecture – Another important factor that may impact attack efficacy is DNN architecture. Here, we compare the performance of various attacks on three DNN model, VGG, ResNet, and DenseNet, with results shown in Figure 7.

We have the following observations. First, different model architectures manifest varying attack vulnerabilities, ranked as ResNet > DenseNet > VGG. This may be explained as follows. Compared with traditional convolutional networks (*e.g.*, VGG), the unique constructs of ResNet (*i.e.*, residual block) and DenseNet (*i.e.*, skip connection) enable more effective feature extraction, but also allow more effective propagation of trigger patterns. Second, among all the attacks, Lb, Imc, and Esb seem insensitive to model architectures, which may be attributed to the optimization strategies of Lb and Imc, and the direct modification of DNN architectures by Esb.

Remark 5 – *DNN architectures that enable better feature extraction may also allow more effective propagation of trigger patterns.*

Attack	Fine-Tuning			Downstream Classifier			
	None	Partial	Full	2-FCN	Bayes	SVM	RF
Bn	72.4	72.3	30.4	72.2	73.5	64.7	66.0
Tnn	91.5	89.6	27.1	90.8	90.3	82.9	81.1
Rb	79.2	77.0	12.4	78.3	76.8	61.5	63.7
Lb	100.0	100.0	95.3	99.9	99.9	99.9	99.8
Imc	100.0	99.9	88.7	99.9	100.0	99.9	99.8

Table 5. Impact of fine-tuning and downstream model on attack efficacy.

Fine-tuning and downstream model – Recall that a trojan model is often composed with a downstream model and fine-tuned for the target task. We evaluate the impact of downstream-model selection and fine-tuning strategy on the attack efficacy. We consider 5 different downstream models (1/2 fully-connected layer, Bayes, SVM, and Random Forest) and 3 fine-tuning strategies (none, partial tuning, and full tuning). Note that the adversary is unaware of such settings.

Table 5 compares the performance of 5 attacks with respect to varying downstream models and fine-tuning methods. Observe that fine-tuning has a great impact on attack efficacy. For instance, the *ASR* of Tnn drops by 62.5% from partial- to full-tuning. In comparison, Lb and Imc are less subjective to fine-tuning, due to their optimization strategies. Also observe that the attack performance seems agnostic to the downstream model. This may be explained by that the downstream model in practice tends to manifest "pseudo-linearity" [27], making the system's output linearly correlated with the trojan model's output (more details in Appendix A).

Remark 6 – *The performance of most attacks is subjective to fine-tuning strategies but agnostic to downstream-model selections.*

Transferability – Next, we consider the setting that without access to the data from the downstream task, the adversary pre-trains the trojan model on a surrogate dataset and transfers the attack to the target dataset.

Transfer Setting		Attack				
Surrogate	Target	Bn	Tnn	Rb	Lb	Imc
CIFAR10	CIFAR10	94.5 (0.99)	100.0 (1.0)	100.0 (1.0)	100.0 (1.0)	100.0 (1.0)
	ImageNet	8.4 (0.29)	7.8 (0.29)	8.6 (0.30)	8.2 (0.30)	9.4 (0.32)
ImageNet	ImageNet	90.0 (0.98)	95.2 (0.99)	94.6 (0.98)	97.4 (0.99)	98.4 (1.0)
	CIFAR10	77.0 (0.84))	26.9 (0.72)	11.0 (0.38)	10.0 (0.38)	14.3 (0.48)

Table 6. *ASR* and *TMC* of transfer attacks across CIFAR10 and ImageNet ($|m|=3\times3,\ \alpha=0.0$).

We evaluate the efficacy of transferring attacks across two datasets, CIFAR10 and ImageNet, with results summarized in Table 6. We have the following findings. Several attacks (*e.g.*, Bn) are able to transfer from ImageNet to CIFAR10 to a certain extent, but most attacks fail to transfer from CIFAR10 to ImageNet. The finding may be justified as follows. A model pre-trained on complex data (*i.e.*, ImageNet) tends to maintain its effectiveness of feature extraction on simple data (*i.e.*, CIFAR10) [16]; as a side effect, it may also preserve its effectiveness of propagating trigger patterns. Meanwhile, a model pre-trained on simple data may not generalize well to complex data. Moreover, compared with stronger attacks in non-transfer cases (*e.g.*, Lb), Bn shows much higher transferability. This may be explained by that to maximize the attack efficacy, the trigger and trojan model often need to "over-fit" the training data, resulting in their poor transferability.

Remark 7 – *Backdoor attacks tend to transfer from complex data to simple data but not vice versa, while weak attacks demonstrate higher transferability than strong ones.*

C. Defense Utility Evaluation

Next, we evaluate the utility preservation of defenses, to understand their impact on the system's normal functionality.

Clean accuracy – We first measure the impact of defenses on the accuracy of classifying clean inputs. As input filtering and model inspection have no direct influence on clean accuracy, we focus on input transformation and model sanitization.

Defense	Attack								
	–	Bn	Tnn	Rb	Tb	Lb	Esb	Abe	Imc
–	95.4	95.3	95.2	95.4	95.3	95.5	95.3	95.0	95.5
Rs	-0.3	-0.6	-0.3	-0.4	-0.4	-0.3	-0.3	-0.4	-0.5
Du	-4.0	-4.5	-4.5	-4.4	-4.3	-4.3	-4.0	-4.9	-4.6
Mp	-11.2	-11.9	-11.3	-10.8	-11.3	-11.4	-11.2	-11.9	-11.0
Fp	-0.1	-0.2	+0.0	+0.0	+0.0	-0.2	-0.2	+0.3	-0.4
Ar	-11.1	-11.1	-10.4	-10.4	-10.4	-10.9	-10.9	-10.5	-11.4

Table 7. Impact of defenses on classification accuracy of CIFAR10 (−: clean model without attack/defense).

Table 7 summarizes the results. With the no-defense setting as the baseline, most defenses tend to negatively affect clean accuracy, yet with varying impact. For instance, across all the cases, Rs and Ar cause about 0.4% and 11% *CAD* respectively. This is explained by the difference of their underlying mechanisms: although both attempt to alleviate the influence of trigger patterns, Rs smooths the prediction of an input x over its vicinity, while Ar forces the model to make consistent predictions in x's vicinity. Also note that Fp attains the least *CAD* across all the cases, mainly due to its fine-tuning.

Remark 8 – *Input-transformation and model-sanitization negatively impact accuracy, while fine-tuning may mitigate such effect.*

Execution Time – We compare the overhead of various defenses by measuring their *ART* (§ III-D). The results are listed in Table 8. Note that online defenses (*e.g.*, Strip) have negligible overhead, while offline methods (*e.g.*, Abs) require longer but acceptable running time (10^3∼10^4 seconds).

Mp	Neo	Strip	Ar	Fp
2.4×10^1	7.7×10^0	1.8×10^{-1}	1.7×10^4	2.1×10^3
Nc	Tabor	Abs	Ni	Di
1.8×10^3	4.2×10^3	1.9×10^3	4.6×10^1	4.1×10^2

Table 8. Running time of various defenses (second).

Remark 9 – *Most defenses have marginal execution overhead with respect to practical datasets and models.*

D. Attack-Defense Interaction Evaluation

In this set of experiments, we evaluate the robustness of existing defenses with respect to various attacks, aiming to characterize their dynamic interactions. As the defenses from different categories bear distinct objectives (*e.g.*, detecting trigger inputs versus cleansing trojan models). we evaluate each defense category separately.

Attack-agnostic defenses – Input transformation and model sanitization mitigate backdoors in an attack-agnostic manner. We measure their robustness using *ARD* and *TMC*.

With the no-defense case as reference, Table 9 compares the robustness of various defenses, with the following findings: (*i*) Mp and Ar are the most robust methods in the categories of input transformation and model sanitization, respectively, which however are attained with over 10% *CAD* (*cf.* Table 7). (*ii*) Fp seems effective against most attacks except Lb and Imc, which is explained as follows: unlike attacks (*e.g.*, Tnn) that optimize the trigger with respect to selected neurons, Lb and Imc perform optimization with respect to all the neurons, making them immune to the pruning of Fp. (*iii*) Most defenses are able to defend against Esb (over 85% *ARD*), which is attributed to its hard-coded trigger pattern and modified DNN architecture: slight perturbation to the trigger input or trojan model may destroy the embedded backdoor.

Remark 10 – *In model sanitization or input transformation, there exists an accuracy-robustness trade-off.*

Input filtering – Next, we evaluate the robustness of input filtering defenses. With respect to each attack, we randomly generate 100 pairs of trigger-clean inputs and measure the *TPR* and *FPR* of Strip and Neo, two representative input filtering methods. To make easy comparison, we fix *FPR* as 0.05 and report *TPR* in Table 10 (more statistics in Appendix B).

We have the following findings. (*i*) Strip is particularly robust against Lb and Imc (over 0.9 *TPR*). Recall that Strip detects a trigger input using the self-entropy of its mixture with a clean input. This indicates that the triggers produced by Lb and Imc effectively dominate the mixtures, which is consistent with the findings in other experiments (*cf.* Figure 1). (*ii*) Neo is robust against most attacks to a limited extent (less than 0.3 *TPR*), but especially effective against Esb (over 0.6 *TPR*), mainly due to its requirement for recognizing the trigger pattern precisely to overwrite the model prediction.

Remark 11 – *Trigger design faces the effectiveness-evasiveness trade-off with respect to input-filtering defenses.*

We also evaluate the impact of trigger definition on the performance of input filtering, with results in Figure 8 (results for other defenses in Appendix B). With fixed trigger transparency, Neo constantly attains higher *TPR* under larger triggers; in comparison, Strip seems less sensitive. This is attributed to the difference of their detection rationale: given an input x, Neo searches for "tipping" position in x to cause prediction change, which is clearly subjective to the trigger size; while Strip measures the self-entropy of x's mixture with a clean input, which does not rely on the trigger size.

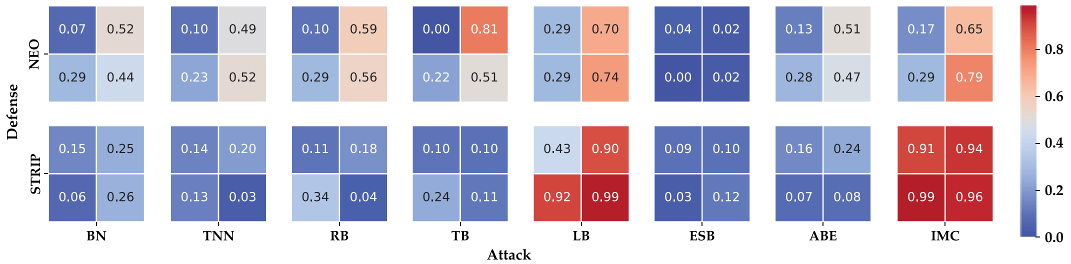

Figure 8: TPR of Neo and Strip under varying trigger definition (left: $|m| = 3 \times 3$, right: $|m| = 6 \times 6$; lower: $\alpha = 0.0$, upper: $\alpha = 0.8$).

Defense	Attack							
	Bn	Tnn	Rb	Tb	Lb	Esb	Abe	Imc
–	93.3 (0.99)	99.9 (1.0)	99.8 (1.0)	96.7 (0.99)	100.0 (1.0)	100.0 (0.86)	95.3 (0.99)	100.0 (1.0)
Rs	-0.5 (0.99)	-0.0 (1.0)	-0.0 -(1.0)	-0.3 (0.99)	-0.0 (1.0)	-89.1 (0.86)	-0.5 (0.99)	-0.0 (1.0)
Du	-2.2 (0.99)	-0.4 (1.0)	-5.4 (1.0)	-67.8 (1.0)	-4.1 (1.0)	-89.9 (0.86)	-0.5 (0.99)	-0.2 (1.0)
Mp	-6.0 (0.99)	-37.4 (1.0)	-78.6 (1.0)	-11.0 (0.99)	-42.6 (1.0)	-87.8 (0.86)	-4.6 (0.99)	-16.0 (1.0)
Fp	-82.9 (0.60)	-86.5 (0.64)	-89.1 (0.73)	-38.0 (0.89)	-27.6 (0.82)	-100.0 (0.81)	-84.5 (0.64)	-26.9 (0.83)
Ar	-83.2 (0.84)	-89.6 (0.85)	-89.8 (0.62)	-86.2 (0.63)	-90.1 (0.83)	-100.0 (0.86)	-85.3 (0.81)	-89.7 (0.83)

Table 9. *ARD* and *TMC* of attack-agnostic defenses against various attacks.

Defense	Attack							
	Bn	Tnn	Rb	Tb	Lb	Esb	Abe	Imc
Strip	0.07	0.13	0.34	0.27	0.91	0.10	0.07	0.99
Neo	0.29	0.23	0.29	0.36	0.29	0.64	0.28	0.29

Table 10. TPR of Neo and Strip (FPR = 0.05, α = 0.0).

> **Remark 12** – *Trigger design also faces the trade-off between the evasiveness with respect to different input-filtering defenses.*

Model inspection – We evaluate model-inspection defenses in terms of their effectiveness of (*i*) identifying trojan models and (*ii*) recovering trigger patterns.

Specifically, given defense $\mathcal{D}$ and model f, we measure the *AIV* of all the classes; if f is a trojan model, we use the *AIV* of the target class to quantify $\mathcal{D}$'s *TPR* of detecting trojan models and target classes; if f is a clean model, we use the largest *AIV* to quantify $\mathcal{D}$'s *FPR* of misclassifying clean models. The results are summarized in Table 11.

	Bn	Tnn	Rb	Tb	Lb	Esb	Abe	Imc
Nc	3.08	2.69	2.48	2.44	2.12	0.04	2.67	1.66
Di	0.54	0.46	0.39	0.29	0.21	0.01	0.76	0.26
Tabor	3.26	2.49	2.32	2.15	2.01	0.89	2.44	1.89
Ni	1.28	0.59	0.78	1.11	0.86	0.71	0.41	0.52
Abs	3.02	4.16	4.10	15.55	2.88		8.45	3.15

Table 11. *AIV* of clean models and trojan models by various attacks.

(*i*) Compared with other defenses, Abs is highly effective in detecting trojan models (with largest *AIV*), attributed to its neuron sifting strategy. (*ii*) Imc seems evasive to most defenses (with *AIV* below 2). This is explained by its trigger-model co-optimization strategy, which minimizes model distortion. (*iii*) Most model-inspection defenses are either ineffective or inapplicable against Esb, as it keeps the original DNN intact but adds an additional module. This contrasts the high effectiveness of other defenses against Esb (*cf.* Table 9).

> **Remark 13** – *There exists a trade-off among the evasiveness with respect to model inspection and other defenses.*

For successfully detected trojan models, we further evaluate the trigger recovery of various defenses by measuring the mask ℓ_1 norm (*MLN*) of recovered triggers and mask jaccard similarity (*MJS*) between recovered and injected triggers, with results shown in Table 12. While the ground-truth trigger has *MLN* $= 9$ ($\alpha = 0.0$, $|m| = 3 \times 3$), most defenses recover triggers of varying *MLN* and non-zero *MJS*, indicating that they recover triggers different from, yet overlapping with, the injected ones. This finding is consistent with recent studies [46], [57] that show a backdoor attack essentially injects a trigger distribution rather than a specific trigger.

> **Remark 14** – *An attack, while targeting a specific trigger, essentially injects a trigger distribution; mode-inspection defenses tend to recover triggers related to, but non-identical to, the injected one.*

V. Exploration

Next, we examine the current practices of operating backdoor attacks and defenses and explore potential improvement.

A. *Attack – Trigger*

We first explore improving the trigger definition by answering the following questions.

RQ$_1$: ***Is it necessary to use large triggers?***

It is found in § IV-B that attack efficacy seems insensitive to trigger size. We now consider the extreme case that the trigger is defined as a single pixel and evaluate the efficacy of different attacks (constrained by *CAD* below 5%), with results show in Table 13. Note that the trigger definition is inapplicable to Esb, due to its requirement for trigger size.

Surprisingly, with single-pixel triggers, most attacks achieve *ASR* comparable with the cases of larger triggers (*cf.* Figure 4). This implies the existence of universal, single-pixel perturbation [44] with respect to trojan models (but not clean models!), highlighting the mutual-reinforcement effects between trigger inputs and trojan models [45].

> **Remark 15** – *There often exists universal, single-pixel perturbation with respect to trojan models (but not clean models).*

RQ$_2$: ***Is it necessary to use regular-shaped triggers?***

The triggers in the existing attacks are mostly regular-shaped (*e.g.*, square), which seems a common design choice.

Defense	Attack															
	Bn		Tnn		Rb		Tb		Lb		Esb		Abe		Imc	
	MLN	MJS	MLN	MJS	MLN	MJS	MLN	MJS	MLN	MJS	MLN	MJS	MLN	MJS	MLN	MJS
Nc	4.98	0.55	4.65	0.70	2.64	0.89	3.53		7.52	0.21	35.16	0.00	5.84	0.42	8.63	0.13
Di	9.65	0.25	6.88	0.17	4.77	0.30	8.44		20.17	0.21	0.00	0.06	10.21	0.30	12.78	0.25
Tabor	5.63	0.70	4.47	0.42	3.03	0.70	3.67		7.65	0.21	43.37	0.00	5.65	0.42	8.69	0.13
Abs	17.74	0.42	17.91	0.55	17.60	0.70	16.00		17.29	0.42			17.46	0.31	17.67	0.31

Table 12. *MLN* and *MJS* of triggers recovered by model-inspection defenses with respect to various attacks (Note: as the trigger position is randomly chosen in Tb, its *MJS* is un-defined).

Bn	Tnn	Rb	Tb	Lb	Esb	Abe	Imc
95.1	98.1	77.7	98.0	100.0		90.0	99.7
(0.99)	(0.96)	(0.96)	(0.99)	(0.99)		(0.97)	(0.99)

Table 13. *ASR* and *TMC* of single-pixel triggers ($\alpha = 0.0$, *CAD* $\leq 5\%$).

We explore the impact of trigger shape on attack efficacy. We fix $|m| = 9$ but select the positions of $|m|$ pixels independently and randomly. Table 14 compares *ASR* under the settings of regular and random triggers.

Trigger Setting	Bn	Tnn	Rb	Lb	Imc
Regular	72.4	91.5	79.2	100.0	100.0
Random	97.6	98.5	92.7	97.6	94.5

Table 14. Comparison of regular and random triggers.

Except Lb and Imc which already attain extremely high *ASR* under the regular-trigger setting, all the other attacks achieve higher *ASR* under the random-trigger setting. For instance, the *ASR* of Bn increases by 25.2%. This may be explained by that lifting the spatial constraint on the trigger entails a larger optimization space for the attacks.

Remark 16 – *Lifting spatial constraints on trigger patterns tends to lead to more effective attacks.*

RQ$_3$: ***Is the "neuron-separation" guidance effective?***

A common search strategy for trigger patterns is using the neuron-separation guidance: searching for triggers that activate neurons rarely used by clean inputs [40]. Here, we validate this guidance by measuring the *NSR* (§ III-B) of benign and trojan models before and after Fp, as shown in Table 15.

Fine-Pruning	–	Bn	Tnn	Rb	Lb	Abe	Imc
Before	0.03	0.59	0.61	0.65	0.61	0.54	0.64
After	0.03	0.20	0.19	0.27	0.37	0.18	0.38

Table 15. *NSR* of benign and trojan models before and after Fp.

Across all the cases, compared with its benign counterpart, the trojan model tends to have higher *NSR*, while fine-tuning reduces *NSR* significantly. More effective attacks (*cf.* Figure 1) tend to have higher *NSR* (*e.g.*, Imc). We thus conclude that the neuron-separation heuristic is in general valid.

Remark 17 – *The separation between the neurons activated by clean and trigger inputs is an indicator of attack effectiveness.*

B. Attack – Optimization

We now examine the optimization strategies used by exiting attacks and explore potential improvement.

RQ$_4$: ***Is it necessary to start from benign models?***

To forge a trojan model, a common strategy is to re-train a benign, pre-trained model. Here, we challenge this practice by evaluating whether re-training a benign model leads to more effective attacks than training a trojan model from scratch.

Training Strategy	Bn	Tnn	Rb	Lb	Imc
Benign model re-training	72.4	91.5	79.2	100.0	100.0
Training from scratch	76.9	98.9	81.2	100.0	100.0

Table 16. *ASR* of trojan models by training from scratch and re-training from benign models.

Table 16 compares the *ASR* of trojan models generated using the two strategies. Except Lb and Imc achieving similar *ASR* in both settings, the other attacks observe marginal improvement if they are trained from scratch. For instance, the *ASR* of Tnn improves by 7.4%. One possible explanation is as follows. Let f and f^* represent the benign and trojan models, respectively. In the parameter space, re-training constrains the search for f^* within in f's vicinity, while training from scratch searches for f^* in the vicinity of a randomly initialized configuration, which may lead to better starting points.

Remark 18 – *Training from scratch tends to lead to more effective attacks than benign-model re-training.*

RQ$_5$: ***Is it feasible to exploit model architectures?***

Most attacks train trojan models in a model-agnostic manner, ignoring their unique architectures (*e.g.*, residual block). We explore the possibility of exploiting such features.

We consider the skip-connection structures in many DNNs (*e.g.*, ResNet) and attempt to improve the gradient backprop in training trojan models. In such networks, gradients propagate through both skip connections and residual blocks. By setting the weights of gradients from skip connections or residual blocks, it amplifies the gradient update towards inputs or model parameters [66]. Specifically, we modify the backprop procedure in Imc by setting a decay coefficient $\gamma = 0.5$ for the gradient through skip connections, with *ASR* improvement over normal training shown in Figure 9.

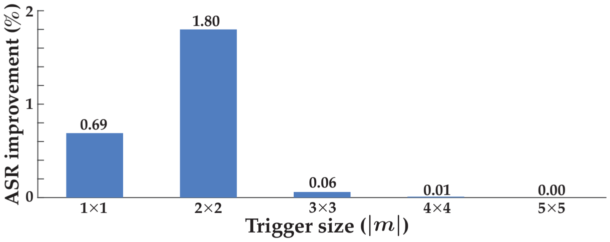

Figure 9: *ASR* improvement by reducing skip-connection gradients ($\alpha = 0.9$).

Observe that by reducing the skip-connection gradients, it marginally improves the *ASR* of Imc especially for small triggers (*e.g.*, $|m| = 2 \times 2$). We consider searching for the optimal γ to maximize attack efficacy as our ongoing work.

Remark 19 – *It is feasible to exploit skip-connection structures to improve attack efficacy marginally.*

RQ$_6$: *How to mix clean and trigger inputs in training?*

To balance attack efficacy and specificity, the adversary often mixes clean and trigger inputs in training trojan models. There are typically three mixing strategies: (*i*) dataset-level – mixing trigger inputs $\mathcal{T}_t$ with clean inputs $\mathcal{T}_c$ directly, (*ii*) batch-level – adding trigger inputs to each batch of clean inputs during training, and (*iii*) loss-level – computing and aggregating the average losses of $\mathcal{T}_t$ and $\mathcal{T}_c$. Here, we fix the mixing coefficient $\lambda = 0.01$ and compare the effectiveness of different strategies.

Mixing Strategy	Bn	Tnn	Rb	Lb	Imc
Dataset-level	59.3	72.2	46.2	99.6	92.0
Batch-level	72.4	91.5	79.2	100.0	100.0
Loss-level	21.6	22.9	18.1	33.6	96.5

Table 17. Impact of mixing strategies on attack efficacy ($\alpha = 0.0$, $\lambda = 0.01$).

We observe in Table 17 that across all the cases, the batch-level mixing strategy leads to the highest *ASR*. This can be explained as follows. With dataset-level mixing, the ratio of trigger inputs in each batch tends to fluctuate significantly due to random shuffling, resulting in inferior training quality. With loss-level mixing, $\lambda = 0.01$ results in fairly small gradients of trigger inputs, equivalent to setting a overly small learning rate. In comparison, batch-level mixing fixes the ratio of trigger inputs in each batch and the weight of their gradients in updating trojan models.

> **Remark 20** – *Batch-level mixing tends to lead to the most effective training of trojan models.*

RQ$_7$: *How to optimize the trigger pattern?*

An attack involves optimizing both the trigger pattern and the trojan model. The existing attacks use 3 typical strategies: (*i*) Pre-defined trigger – it fixes the trigger pattern and only optimizes the trojan model. (*ii*) Partially optimized trigger – it optimizes the trigger pattern in a pre-processing stage and optimizes the trojan model. (*iii*) Trigger-model co-optimization – it optimizes the trigger pattern and the trojan model jointly during training. Here, we implement 3 variants of Bn that use these optimization strategies, respectively. Figure 10 compares their *ASR* under varying trigger transparency. Observe that the trigger-optimization strategy has a significant impact on *ASR*, especially under high transparency. For instance, if $\alpha = 0.9$, the co-optimization strategy improves *ASR* by over 60% from the non-optimization strategy.

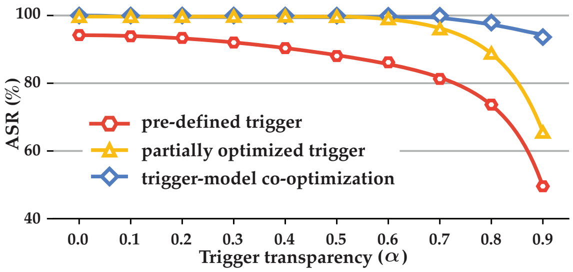

Figure 10: Impact of trigger optimization.

> **Remark 21** – *Optimizing the trigger pattern and the trojan model jointly leads to more effective attacks.*

C. Defense – Interpretability

RQ$_8$: *Does interpretability help mitigate backdoor attacks?*

The interpretability of DNNs explain how they make predictions for given inputs [51], [17]. Recent studies [59], [22] show that such interpretability helps defend against adversarial attacks. Here, we explore whether it mitigates backdoor attacks. Specifically, for a pair of benign-trojan models and 100 pairs of clean-trigger inputs, we generate the attribution map [51] of each input with respect to both models and ground and target classes, with an example shown in Figure 11.

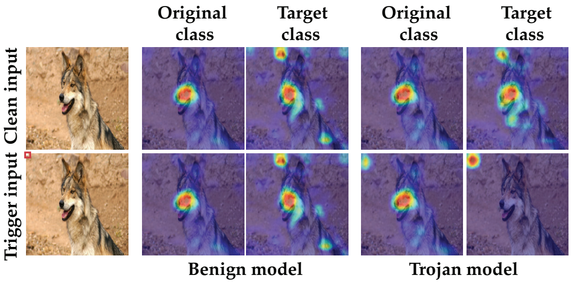

Figure 11: Sample attribution maps of clean and trigger inputs with respect to benign and trojan models ($\alpha = 0.0$, ImageNet).

We measure the difference (ℓ_1-norm normalized by image size) of attribution maps of clean and trigger inputs. Observe in Table 18 that their attribution maps with respect to the target class differ significantly on the trojan model, indicating the possibility of using interpretability to detect the attack. Yet, it requires further study whether the adversary may adapt the attack to deceive such detection [73].

Benign model		**Trojan model**	
Original class	**Target class**	**Original class**	**Target class**
0.08%	0.12%	0.63%	8.52%

Table 18. Heatmap difference of clean and trigger inputs ($\alpha = 0.0$, ImageNet).

> **Remark 22** – *It seems feasible to exploit interpretability to defend against backdoor attacks.*

D. Defense – Mitigation

RQ$_9$: *Is unlearning able to cleanse backdoors?*

Once a backdoor is detected, the natural follow-up is to sanitize the trojan model. We explore whether the existing methods remove the backdoors. We exemplify with unlearning [64]: it applies the recovered trigger on clean data to generate trigger inputs and uses such inputs together with their ground-truth classes to re-trains the trojan model.

We apply unlearning with (*i*) the original trigger and (*ii*) the trigger recovered by Nc. In both cases, according to Nc, the unlearning successfully cleanses all the backdoors (details in Appendix B). We then apply Abs to re-inspect the unlearned models and measure the *ASR* and *MLN* of each class before and after unlearning, with results listed in Table 19. Interestingly, before unlearning, only class 0 is infected with a backdoor; after unlearning in both cases, all the classes are infected with backdoors (with *ASR* above 90% and *MLN* below 16)!

> **Remark 23** – *Unlearning tends to introduce new vulnerabilities.*

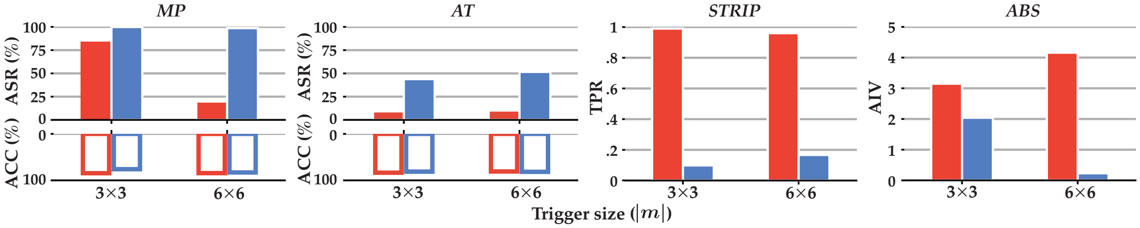

Figure 12: Performance of non-adaptive and adaptive IMC against representative defenses ($\alpha = 0.0$).

Class	Before		After (detected)		After (original)		Adapted	
	ASR	*MLN*	*ASR*	*MLN*	*ASR*	*MLN*	*ASR*	*MLN*
0	100.0	17.75	94.96	16.00	94.76	15.83	87.13	16.00
1	1.30	94.38	94.39	16.00	94.37	16.00	86.44	16.00
2	99.45	100.72	94.20	16.00	93.97	16.00	86.64	16.00
3	59.52	101.14	94.88	16.00	94.72	15.83	87.10	16.00
4	0.79	100.57	93.98	16.00	94.39	16.00	86.58	16.00
5	6.12	20.86	94.42	16.00	94.51	16.00	86.68	16.00
6	29.77	100.21	94.35	16.00	94.33	16.00	86.66	16.00
7	6.07	100.57	94.48	16.00	94.36	16.00	86.56	16.00
8	27.37	101.53	94.43	16.00	94.16	16.00	86.69	16.00
9	99.77	100.47	94.35	16.00	94.19	16.00	86.41	16.00

Table 19. Impact of unlearning and adaptation to STRIP (detected – unlearning using trigger detected by NC; original – unlearning using ground-truth trigger; adapted – adaptation to STRIP).

E. *Defense – Evadability*

RQ$_{10}$: ***Are the existing defenses evadable?***
We now explore whether the existing defenses are potentially evadable by adaptive attacks. We select IMC as the basic attack, due to its flexible optimization framework, and consider MP, AR, STRIP, and ABS as the representative defenses from the categories in Table 2. Specifically, we adapt IMC to each defense (details deferred to Appendix A).

We compare the efficacy of non-adaptive and adaptive IMC, as shown in Figure 12. Observe that across all the cases, the adaptive IMC significantly outperforms the non-adaptive one. For instance, under $|m| = 6 \times 6$, it increases the *ASR* with respect to MP by 80% and reduces the *TPR* of STRIP by over 0.85. Also note that a larger trigger size leads to more effective adaptive attack, as it entails a larger optimization space.

Remark 24 – *Most existing defenses are potentially evadable by adaptive attacks.*

RQ$_{11}$: ***Are ensemble defenses more robust?***
Given that different defenses focus on distinct aspects of backdoors, a promising approach is to integrate multiple, complementary defenses. Here, we exemplify with STRIP, which exploits trigger pattern anomaly, and ABS, which focuses on neuron activation anomaly. We apply ABS to inspect the trojan model generated by IMC adapted to STRIP, with results in Table 19. Observe that ABS detects all the classes infected with backdoors. We have the following explanation. Adapting to STRIP requires to train the trojan model using trigger inputs of high transparency together with their original classes (*cf.* Appendix A). This is essentially unlearning to forget high-transparency triggers while keeping low-transparency triggers effective. Thus, the adaptation to STRIP tends to make the trojan model more susceptible to ABS.

Remark 25 – *Integrating defenses against trigger inputs and trojan models may lead to robustness against adaptive attacks.*

VI. DISCUSSION

A. *Limitations of* TROJANZOO

First, to date TROJANZOO has integrated 12 attacks and 15 defenses, representing the state of the art of neural backdoor research. The current implementation does not include certain concurrent work [69], [65], [35]. However, thanks to its modular design, TROJANZOO can be readily extended to incorporate new attacks, defenses, and metrics. Moreover, we plan to open-source all the code and data of TROJANZOO and encourage the community to contribute.

Second, to conduct unified evaluation, we mainly consider the attack vector of re-using pre-trained trojan models. There are other attack vectors through which backdoor attacks can be launched, including poisoning victims' training data [52], [75] and knowledge distillation [71], which entail additional constraints for attacks or defenses. For instance, the poisoning data needs to be evasive to bypass inspection. We consider studying alternative attack vectors as our ongoing work.

Finally, because of the plethora of work on neural backdoors in the computer vision domain, TROJANZOO focuses on the image classification task, while recent work has also explored neural backdoors in other settings, including natural language processing [49], [31], [74], reinforcement learning [29], and federated learning [5], [67]. We plan to extend TROJANZOO to support such settings in its future releases.

B. *Additional Related Work*

Recent studies have surveyed neural backdoors [34], [18], [38]. However, none of them provides reference implementation or conducts empirical evaluation. Compared with the rich collection of platforms for adversarial attacks/defenses (*e.g.*, CLEVERHANS [2], DEEPSEC [36], and ADVBOX [1]), only few platforms currently support evaluating neural backdoors: ART [3] integrates 3 attacks and 3 defenses, while TROJAI [28] implements 1 attack and 3 metrics.

In comparison, TROJANZOO differs in major aspects: (*i*) to our best knowledge, it features the most comprehensive library of attacks/defenses; (*ii*) it regards the evaluation metrics as a first-class citizen and implements 6 attack performance metrics and 10 defense utility metrics, which holistically assess given attacks/defenses; (*iii*) besides reference implementation, it also provides rich utility tools for in-depth analysis of attack-defense interactions, such as measuring feature-space similarity, tracing neural activation patterns, and comparing attribution maps.

VII. Conclusion

We design and implement TrojanZoo, the first platform dedicated to assessing neural backdoor attacks/defenses in a holistic, unified, and practical manner. Leveraging TrojanZoo, we conduct a systematic evaluation of existing attacks/defenses, which demystifies a number of open questions, reveals various design trade-offs, and sheds light on further improvement. We envision TrojanZoo will serve as a useful benchmark to facilitate neural backdoor research.

References

[1] Advbox. https://github.com/advboxes/AdvBox/.

[2] CleverHans Adversarial Examples Library. https://github.com/tensorflow/cleverhans/.

[3] IBM Adversarial Robustness Toolbox (ART). https://github.com/Trusted-AI/adversarial-robustness-toolbox/.

[4] Eugene Bagdasaryan and Vitaly Shmatikov. Blind Backdoors in Deep Learning Models. *ArXiv e-prints*, 2020.

[5] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How To Backdoor Federated Learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

[6] Battista Biggio, Giorgio Fumera, Fabio Roli, and Luca Didaci. Poisoning Adaptive Biometric Systems. In *Proceedings of Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition (SSPR&SPR)*, 2012.

[7] BVLC. Model zoo. https://github.com/BVLC/caffe/wiki/Model-Zoo, 2017.

[8] Qiong Cao, Li Shen, Weidi Xie, Omkar M Parkhi, and Andrew Zisserman. Vggface2: A dataset for recognising faces across pose and age. In *13th IEEE International Conference on Automatic Face & Gesture Recognition*, 2018.

[9] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting Backdoor Attacks on Deep Neural Networks by Activation Clustering. In *ArXiv e-prints*, 2018.

[10] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. DeepInspect: A Black-box Trojan Detection and Mitigation Framework for Deep Neural Networks. In *Proceedings of International Joint Conference on Artificial Intelligence*, 2019.

[11] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning. *ArXiv e-prints*, 2017.

[12] Jeremy M Cohen, Elan Rosenfeld, and J. Zico Kolter. Certified Adversarial Robustness via Randomized Smoothing. In *Proceedings of IEEE Conference on Machine Learning (ICML)*, 2019.

[13] Paul Cooper. Meet AISight: The scary CCTV network completely run by AI. http://www.itproportal.com/, 2014.

[14] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. ImageNet: A Large-scale Hierarchical Image Database. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[15] Bao Doan, Ehsan Abbasnejad, and Damith Ranasinghe. Februus: Input Purification Defense Against Trojan Attacks on Deep Neural Network Systems. In *Proceedings of Annual Computer Security Applications Conference (ACSAC)*, 2020.

[16] Andre Esteva, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M. Swetter, Helen M. Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115–118, 2017.

[17] Ruth C Fong and Andrea Vedaldi. Interpretable Explanations of Black Boxes by Meaningful Perturbation. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, 2017.

[18] Yansong Gao, Bao Gia Doan, Zhi Zhang, Siqi Ma, Jiliang Zhang, Anmin Fu, Surya Nepal, and Hyoungshick Kim. Backdoor Attacks and Countermeasures on Deep Learning: A Comprehensive Review. *ArXiv e-prints*, 2020.

[19] Yansong Gao, Chang Xu, Derui Wang, Shiping Chen, Damith Ranasinghe, and Surya Nepal. STRIP: A Defence Against Trojan Attacks on Deep Neural Networks. In *Proceedings of Annual Computer Security Applications Conference (ACSAC)*, 2019.

[20] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

[21] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain. *ArXiv e-prints*, 2017.

[22] Wenbo Guo, Dongliang Mu, Jun Xu, Purui Su, Gang Wang, and Xinyu Xing. LEMNA: Explaining Deep Learning Based Security Applications. In *Proceedings of ACM Conference on Computer and Communications (CCS)*, 2018.

[23] Wenbo Guo, Lun Wang, Xinyu Xing, Min Du, and Dawn Song. TABOR: A Highly Accurate Approach to Inspecting and Restoring Trojan Backdoors in AI Systems. In *Proceedings of IEEE International Conference on Data Mining (ICDM)*, 2019.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[25] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely Connected Convolutional Networks. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[26] Xijie Huang, Moustafa Alzantot, and Mani Srivastava. NeuronInspect: Detecting Backdoors in Neural Networks via Output Explanations. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

[27] Yujie Ji, Xinyang Zhang, Shouling Ji, Xiapu Luo, and Ting Wang. Model-Reuse Attacks on Deep Learning Systems. In *Proceedings of ACM SAC Conference on Computer and Communications (CCS)*, 2018.

[28] Kiran Karra, Chace Ashcraft, and Neil Fendley. The TrojAI Software Framework: An OpenSource tool for Embedding Trojans into Deep Learning Models. *ArXiv e-prints*, 2020.

[29] Panagiota Kiourti, Kacper Wardega, Susmit Jha, and Wenchao Li. TrojDRL: Trojan Attacks on Deep Reinforcement Learning Agents. *ArXiv e-prints*, 2019.

[30] Alex Krizhevsky and Geoffrey Hinton. Learning Multiple Layers of Features from Tiny Images. *Technical report, University of Toronto*, 2009.

[31] Keita Kurita, Paul Michel, and Graham Neubig. Weight Poisoning Attacks on Pre-trained Models. In *Proceedings of Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

[32] Te Lester Juin Tan and Reza Shokri. Bypassing Backdoor Detection Algorithms in Deep Learning. In *Proceedings of IEEE European Symposium on Security and Privacy (Euro S&P)*, 2020.

[33] Shaofeng Li, Benjamin Zi Hao Zhao, Jiahao Yu, Minhui Xue, Dali Kaafar, and Haojin Zhu. Invisible Backdoor Attacks Against Deep Neural Networks. *ArXiv e-prints*, 2019.

[34] Yiming Li, Baoyuan Wu, Yong Jiang, Zhifeng Li, and Shu-Tao Xia. Backdoor Learning: A Survey. *ArXiv e-prints*, 2020.

[35] Junyu Lin, Lei Xu, Yingqi Liu, and Xiangyu Zhang. Composite Backdoor Attack for Deep Neural Network by Mixing Existing Benign Features. In *Proceedings of ACM SAC Conference on Computer and Communications (CCS)*, 2020.

[36] X. Ling, S. Ji, J. Zou, J. Wang, C. Wu, B. Li, and T. Wang. DEEPSEC: A Uniform Platform for Security Analysis of Deep Learning Model. In *Proceedings of IEEE Symposium on Security and Privacy (S&P)*, 2019.

[37] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks. In *Proceedings of Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2018.

[38] Y. Liu, A. Mondal, A. Chakraborty, M. Zuzak, N. Jacobsen, D. Xing, and A. Srivastava. A Survey on Neural Trojans. In *Proceedings of International Symposium on Quality Electronic Design (ISQED)*, 2020.

[39] Yingqi Liu, Wen-Chuan Lee, Guanhong Tao, Shiqing Ma, Yousra Aafer, and Xiangyu Zhang. ABS: Scanning Neural Networks for Back-Doors by Artificial Brain Stimulation. In *Proceedings of ACM SAC Conference on Computer and Communications (CCS)*, 2019.

[40] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. In *Proceedings of Network and Distributed System Security Symposium (NDSS)*, 2018.

[41] Yunfei Liu, Xingjun Ma, James Bailey, and Feng Lu. Reflection Backdoor: A Natural Backdoor Attack on Deep Neural Networks. In *Proceedings of European Conference on Computer Vision (ECCV)*, 2020.

[42] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2018.

[43] Dongyu Meng and Hao Chen. MagNet: A Two-Pronged Defense Against Adversarial Examples. In *Proceedings of ACM SAC Conference on Computer and Communications (CCS)*, 2017.

[44] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, Pascal Frossard, and Stefano Soatto. Analysis of Universal Adversarial Perturbations. *ArXiv e-prints*, 2017.

[45] Ren Pang, Hua Shen, Xinyang Zhang, Shouling Ji, Yevgeniy Vorobeychik, Xiapu Luo, Alex Liu, and Ting Wang. A Tale of Evil Twins: Adversarial Inputs versus Poisoned Models. In *Proceedings of ACM SAC Conference on Computer and Communications (CCS)*, 2020.

[46] Ximing Qiao, Yukun Yang, and Hai Li. Defending Neural Backdoors via Generative Distribution Modeling. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[47] Aniruddha Saha, Akshayvarun Subramanya, and Hamed Pirsiavash. Hidden Trigger Backdoor Attacks. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

[48] Ahmed Salem, Rui Wen, Michael Backes, Shiqing Ma, and Yang Zhang. Dynamic Backdoor Attacks Against Machine Learning Models. *ArXiv e-prints*, 2020.

[49] Roei Schuster, Tal Schuster, Yoav Meri, and Vitaly Shmatikov. Humpty Dumpty: Controlling Word Meanings via Corpus Poisoning. In *Proceedings of IEEE Symposium on Security and Privacy (S&P)*, 2020.

[50] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. Hidden Technical Debt in Machine Learning Systems. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[51] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, 2017.

[52] Ali Shafahi, W. Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison Frogs! Targeted Clean-Label Poisoning Attacks on Neural Networks. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[53] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2014.

[54] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. Man vs. Computer: Benchmarking Machine Learning Algorithms for Traffic Sign Recognition. *Neural Metworks*, pages 323–32, 2012.

[55] Jacob Steinhardt, Pang Wei Koh, and Percy Liang. Certified Defenses for Data Poisoning Attacks. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[56] Octavian Suciu, Radu Mărginean, Yiğitcan Kaya, Hal Daumé, III, and Tudor Dumitraş. When Does Machine Learning FAIL? Generalized Transferability for Evasion and Poisoning Attacks. In *Proceedings of USENIX Security Symposium (SEC)*, 2018.

[57] Mingjie Sun, Siddhant Agarwal, and J. Zico Kolter. Poisoned Classifiers Are Not Only Backdoored, They Are Fundamentally Broken. *ArXiv e-prints*, 2020.

[58] Ruixiang Tang, Mengnan Du, Ninghao Liu, Fan Yang, and Xia Hu. An Embarrassingly Simple Approach for Trojan Attack in Deep Neural Networks. In *Proceedings of ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, 2020.

[59] Guanhong Tao, Shiqing Ma, Yingqi Liu, and Xiangyu Zhang. Attacks Meet Interpretability: Attribute-Steered Detection of Adversarial Samples. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[60] Brandon Tran, Jerry Li, and Aleksander Madry. Spectral Signatures in Backdoor Attacks. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[61] Alexander Turner, Dimitris Tsipras, and Aleksander Madry. Clean-Label Backdoor Attacks. *ArXiv e-prints*, 2019.

[62] Sakshi Udeshi, Shanshan Peng, Gerald Woo, Lionell Loh, Louth Rawshan, and Sudipta Chattopadhyay. Model Agnostic Defence against Backdoor Attacks in Machine Learning. *ArXiv e-prints*, 2019.

[63] Allyson Versprille. Researchers Hack Into Driverless Car System, Take Control of Vehicle. http://www.nationaldefensemagazine.org/, 2015.

[64] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao. Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks. In *Proceedings of IEEE Symposium on Security and Privacy (S&P)*, 2019.

[65] Maurice Weber, Xiaojun Xu, Bojan Karlas, Ce Zhang, and Bo Li. RAB: Provable Robustness Against Backdoor Attacks. *ArXiv e-prints*, 2020.

[66] Dongxian Wu, Yisen Wang, Shu-Tao Xia, James Bailey, and Xingjun Ma. Skip Connections Matter: On the Transferability of Adversarial Examples Generated with ResNets. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2020.

[67] Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. DBA: Distributed Backdoor Attacks against Federated Learning. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2020.

[68] W. Xu, D. Evans, and Y. Qi. Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks. In *Proceedings of Network and Distributed System Security Symposium (NDSS)*, 2018.

[69] Xiaojun Xu, Qi Wang, Huichen Li, Nikita Borisov, Carl A. Gunter, and Bo Li. Detecting AI Trojans Using Meta Neural Analysis. In *Proceedings of IEEE Symposium on Security and Privacy (S&P)*, 2020.

[70] Yuanshun Yao, Huiying Li, Haitao Zheng, and Ben Y. Zhao. Latent Backdoor Attacks on Deep Neural Networks. In *Proceedings of ACM SAC Conference on Computer and Communications (CCS)*, 2019.

[71] Kota Yoshida and Takeshi Fujino. Disabling Backdoor and Identifying Poison Data by Using Knowledge Distillation in Backdoor Attacks on Deep Neural Networks. In *Proceedings of ACM Workshop on Artificial Intelligence and Security (AISec)*, 2020.

[72] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How Transferable Are Features in Deep Neural Networks? In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

[73] Xinyang Zhang, Ningfei Wang, Hua Shen, Shouling Ji, Xiapu Luo, and Ting Wang. Interpretable Deep Learning under Fire. In *Proceedings of USENIX Security Symposium (SEC)*, 2020.

[74] Xinyang Zhang, Zheng Zhang, and Ting Wang. Trojaning Language Models for Fun and Profit. *ArXiv e-prints*, 2020.

[75] Chen Zhu, W. Ronny Huang, Ali Shafahi, Hengduo Li, Gavin Taylor, Christoph Studer, and Tom Goldstein. Transferable Clean-Label Poisoning Attacks on Deep Neural Nets. In *Proceedings of IEEE Conference on Machine Learning (ICML)*, 2019.

Appendix A
Implementation Details

Here we elaborate on the implementation of attacks and defenses in this paper.

A. *Symbols and Notations*

Notation	Definition
$\mathcal{A}, \mathcal{D}$	attack, defense
x, x^*	clean input, trigger input
x_i	i-th dimension of x
r	trigger
m	mask (α for each pixel)
f, f^*	benign model, trojan model
g, g^*	downstream model, surrogate model
t	adversary's target class
$\mathcal{T}$	reference set
$\mathcal{R}_\epsilon, \mathcal{F}_\delta$	trigger, model feasible sets

Table 20. Symbols and notations.

B. *Default Parameter Setting*

Table 21 and 22 summarize the default parameter setting in our empirical evaluation (§ IV).

C. *Pseudo-linearity of downstream model*

We have shown in § IV that most attacks seem agnostic to the downstream model. Here, we provide possible explanations. Consider a binary classification setting and a trigger input x with ground-truth class "-" and target class "+". Recall that a backdoor attack essentially shifts x in the feature space by maximizing the quantity of

$$\Delta_f = \mathbb{E}_{\mu^+}[f(x)] - \mathbb{E}_{\mu^-}[f(x)] \tag{7}$$

where μ^+ and μ^- respectively denote the data distribution of the ground-truth positive and negative classes.

Attack	Parameter	Setting
Training	learning rate	0.01
	retrain epoch	50
	optimizer	SGD (nesterov)
	momentum	0.9
	weight decay	2e-4
Bn	toxic data percent	1%
Tnn	preprocess layer	penultimate logits
	neuron number	2
	preprocess optimizer	PGD
	preprocess lr	0.015
	preprocess iter	20
	threshold	5
	target value	10
Rb	candidate number	50
	selection number	10
	selection iter	5
	inner epoch	5
Lb	preprocess layer	penultimate logits
	preprocess lr	0.1
	preprocess optimizer	Adam (tanh constrained)
	preprocess iter	100
	samples per class	1000
	MSE loss weight	0.5
Esb	TrojanNet	4-layer MLP
	hidden neurons per layer	8
	single layer structure	[fc, bn, relu]
	TrojanNet influence	α =0.7
	amplify rate	100
	temperature	0.1
Abe	discriminator loss weight	λ =0.1
	discriminator lr	1e-3
Imc	trigger optimizer	PGD
	PGD lr	α =20/255
	PGD iter	20

Table 21. Attack default parameter setting.

Now consider the end-to-end system $g \circ f$. The likelihood that x is misclassified into "+" is given by:

$$\Delta_{g \circ f} = \mathbb{E}_{\mu^+}[g \circ f(x)] - \mathbb{E}_{\mu^-}[g \circ f(x)] \tag{8}$$

One sufficient condition for the attack to succeed is that $\Delta_{g \circ f}$ is linearly correlated with Δ_f (*i.e.*, $\Delta_{g \circ f} \propto \Delta_f$). If so, we say that the function represented by g is *pseudo-linear*. Unfortunately, in practice, most downstream models are fairly simple (*e.g.*, one fully-connected layer), showing pseudo-linearity. Possible reasons include: (*i*) complex architectures are difficult to train especially when the training data is limited; (*ii*) they imply much higher computational overhead; (*iii*) the ground-truth mapping from the feature space to the output space may indeed be pseudo-linear.

D. Adaptive attacks

Here we detail the adaption of Imc to the defenses of Mp, Ar, Strip, and Abs in § V.

Recall that Mp uses an auto-encoder to downsample then upsample a given input, during which the trigger pattern tends to be blurred and loses effect. To adapt Imc to Mp, we train a surrogate autoencoder h and conduct optimization with inputs reformed by h.

Defense	Parameter	Setting
Rs	sample distribution	Gaussian
	sample number	100
	sample std	0.01
Du	downsample filter	Anti Alias
	downsample ratio	0.95
Mp	training noise std	0.1
	structure	[32]
Strip	mixing weight	0.5 (equal)
	sample number	64
Neo	sample number	100
	Kmeans cluster number	3
	threshold	80
Ar	PGD lr	α =2/255
	perturbation threshold	ϵ =8/255
	PGD iter	7
	learning rate	0.01
	epoch	50
Fp	prune ratio	0.95
Nc	norm regularization weight	1e-3
	remask lr	0.1
	remask epoch per label	10
Di	sample dataset ratio	0.1
	noise dimension	100
	remask lr	0.01
	remask epoch per label	20
Tabor	regularization weight	λ_1 =1e-6
		λ_2 =1e-5
		λ_3 =1e-7
		λ_4 =1e-8
		λ_5 =0
		λ_6 =1e-2
Ni	weighting coefficient	λ_{sp} =1e-5
		λ_{sm} =1e-5
		λ_{pe} =1
	threshold	0
	sample ratio	0.1
Abs	sample k	1
	sample number	5
	max trojan size	16
	remask lr	0.1
	remask iter per neuron	1000
	remask weight	0.1 if norm< 16
		10 if 16 <norm< 100
		100 if norm> 100

Table 22. Defense default parameter setting.

Recall that Ar considers trigger inputs as one type of adversarial inputs and applies adversarial training to improve model robustness against backdoor attacks. To adapt Imc to Ar, during training f^*, we replace clean accuracy loss with adversarial accuracy loss; thus, the process is a combination of adversarial training and trojan model training, resulting in a robust but trojaned model. This way, Ar has a limited impact on the embedded backdoor, as the model is already robust.

Recall that Strip mixes up given inputs with clean inputs and measures the self-entropy of their predictions. Note that in the mixture, the transparency of the original trigger is doubled; yet, Strip works as the high-transparency trigger remains effective. To adapt Imc to Strip, we use trigger inputs with high-transparency triggers together with their ground-truth classes to re-train f^*. The re-training reduces the effectiveness of high-transparency triggers while keeping low-transparency triggers effective.

Recall that Abs identifies triggers by maximizing abnormal activation while preserving normal neuron behavior. To adapt Imc to Abs, we integrate the cost function (Algorithm 2 in [39]) in the loss function to train f^*.

Appendix B
Additional Experiments

A. Attack

Figure 13 and 14 complement the results of attack performance evaluation on ImageNet with respect to trigger size and trigger transparency in Section IV-B.

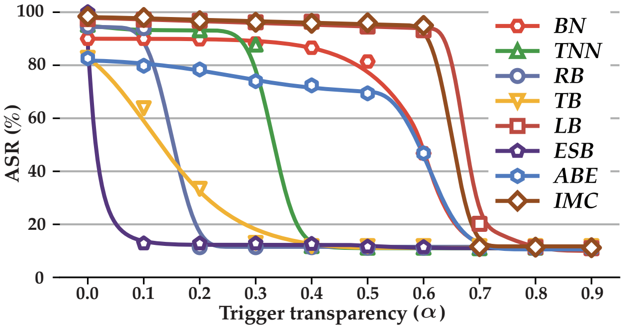

Figure 13: *ASR* with respect to trigger transparency ($|m|$ = 3×3, ImageNet).

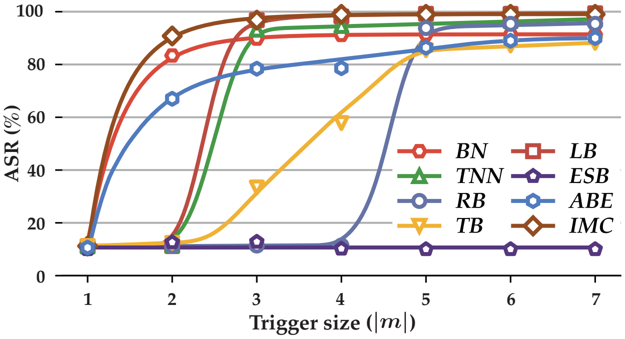

Figure 14: *ASR* with respect to trigger transparency (α = 0.3, ImageNet).

Table 23 complements the results in Table 4.

	Bn	Tnn	Rb	Tb	Lb	Esb	Abe	Imc
GTSRB	65.63	71.70	0.94	0.58	98.42	68.41	68.41	97.58
CIFAR100	64.53	89.76	42.77	23.44	97.83	0.98	67.86	98.75
VGGFace2	85.62	97.30	92.31	88.75	98.08	100.00	72.74	98.43

Table 23. Impact of data complexity on *ASR* ($|m| = 3 \times 3$ and $\alpha = 0.8$ for GTSRB and CIFAR100, $|m| = 25 \times 25$ and $\alpha = 0.0$ for VGGFace2).

B. Defense

Table 24 presents more information (F1-score, precision, recall, and accuracy), which complements Table 10.

Figure 15 and 16 shows the influence of DNN architecture and trigger definition on the performance of attack-agnostic defenses (Mp, Ar, Rs, Du).

Figure 17 illustrate the impact of DNN architecture on the performance of input filtering defenses (Neo, Strip), which complements Figure 8.

Defense	**Measure**	Bn	Tnn	Rb	Tb	Lb	Esb	Abe	Imc
Strip	F1 Score	0.12	0.21	0.47	0.39	0.91	0.18	0.13	0.95
	Precision	0.41	0.56	0.77	0.73	0.90	0.52	0.43	0.91
	Recall	0.07	0.13	0.34	0.27	0.91	0.10	0.07	0.99
	Accuracy	0.48	0.51	0.62	0.58	0.91	0.50	0.49	0.95
Neo	F1 Score	0.45	0.37	0.45	0.34	0.45	0.77	0.43	0.45
	Precision	1.00	1.00	1.00	0.35	1.00	0.96	0.90	1.00
	Recall	0.29	0.23	0.29	0.36	0.29	0.64	0.28	0.29
	Accuracy	0.65	0.62	0.65	0.36	0.65	0.81	0.63	0.65

Table 24. Additional statistics of input filtering.

Figure 18 and 19 illustrate the impact of DNN architecture and trigger definition on the performance of model-inspection defenses (Abs, Ni, Tabor, Di, Nc).

C. Unlearning

Here, we apply unlearning with the trigger recovered by Nc. We then apply Nc to re-inspect the unlearned trojan models and measure the *ASR* and *MLN* of each class before and after unlearning, with results listed in Table 25. According to Nc, the unlearning successfully cleanses all the backdoors.

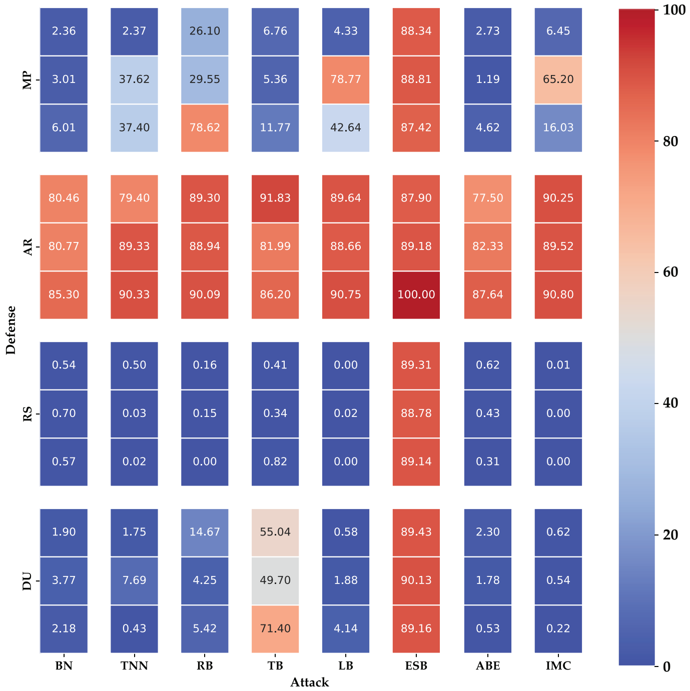

Figure 15: Impact of DNN architecture on attack-agnostic defenses (lower: ResNet18, middle: DenseNet121; upper: VGG13).

Class		0	1	2	3	4	5	6	7	8	9
Before	*ASR*	100.00	27.43	16.97	34.30	12.43	69.97	44.62	12.65	17.91	35.18
	MLN	16.00	16.00	16.00	16.00	16.00	16.00	16.00	16.00	16.00	16.00
After	*ASR*	10.05	9.88	10.10	9.95	10.00	9.43	10.15	10.07	9.87	10.03
	MLN	42.39	41.42	39.65	39.77	41.25	41.74	40.88	39.97	43.44	39.65

Table 25. Impact of unlearning using triggers detected by NC.

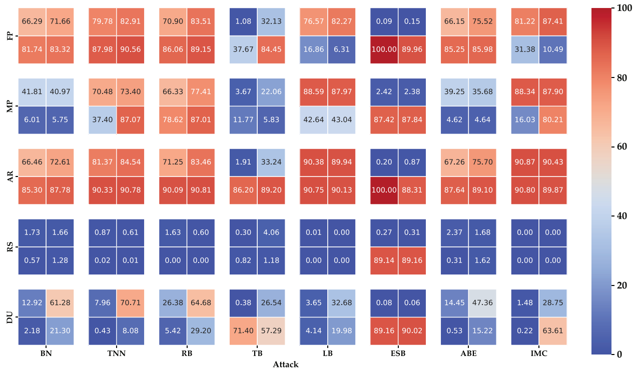

Figure 16: Impact of trigger definition on attack-agnostic defenses (left: $|m| = 3 \times 3$, right: $|m| = 6 \times 6$; lower: $\alpha = 0.0$, upper: $\alpha = 0.8$).

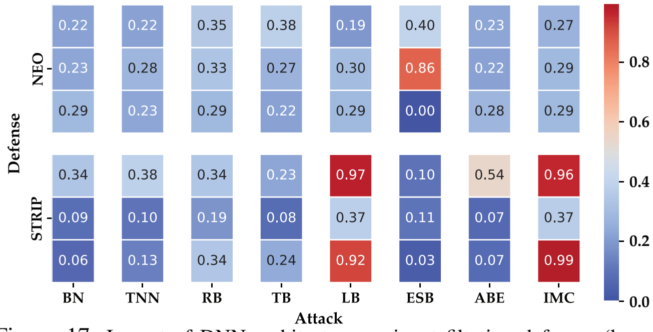

Figure 17: Impact of DNN architecture on input filtering defenses (lower: ResNet18, middle: DenseNet121; upper: VGG13).

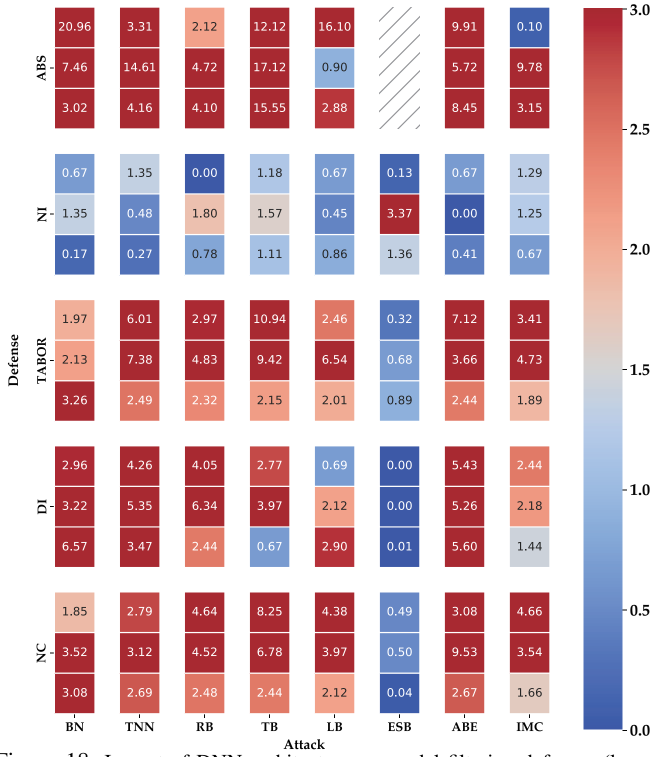

Figure 18: Impact of DNN architecture on model filtering defenses (lower: ResNet18, middle: DenseNet121; upper: VGG13; note: ESB–ABS pair is inapplicable).

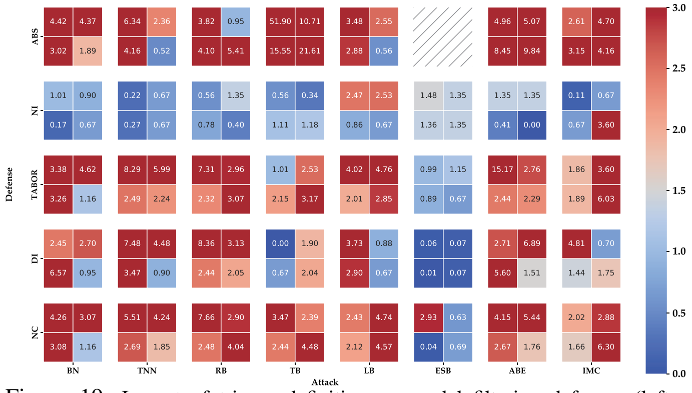

Figure 19: Impact of trigger definition on model filtering defenses (left: $|m| = 3 \times 3$, right: $|m| = 6 \times 6$; lower: $\alpha = 0.0$, upper: $\alpha = 0.8$; note: ESB–ABS pair is inapplicable).