The Pennsylvania State University
The Graduate School

UNDERSTANDING AND MITIGATING NEURAL BACKDOORS

A Dissertation in
Informatics
by
Ren Pang

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

May 2024

The dissertation of Ren Pang was reviewed and approved by the following:

Ting Wang
Associate Professor of College of Information Sciences and Technology
Dissertation Advisor
Chair of Committee

Fenglong Ma
Assistant Professor of College of Information Sciences and Technology

Xiang Zhang
Associate Professor of College of Information Sciences and Technology

Sencun Zhu
Associate Professor of Computer Science and Engineering
Outside Member

Dongwon Lee
Professor of College of Information Sciences and Technology
Director of Graduate Programs

Abstract

The rapid progress in deep learning has led to significant breakthroughs in various machine learning tasks. Despite the remarkable success of deep learning models across domains, the intensive research has produced a plethora of backdoor attacks/defenses, resulting in a constant arms race. However, previous studies have highlighted the intricate trade-offs and complexities involved, yet a fundamental understanding of the connections between different attack vectors remains elusive. Furthermore, the lack of standardized evaluation benchmarks has hindered comprehensive research from multiple critical research questions.

To address these limitations, we present three significant contributions in this dissertation. (i) We propose IMC, which enhances conventional backdoor attacks by jointly optimizing triggers and trojaned models, uncovering intriguing mutual reinforcement effects between the two attack vectors. (ii) We introduce TROJANZOO, an open-source platform designed to evaluate neural backdoor attacks and defenses holistically. Through systematic analysis, TROJANZOO reveals key insights into the design spectrum of existing attacks and defenses. (iii) We extend the scope of backdoor attacks to AutoML by introducing EVAS, a novel attack leveraging neural architecture search to discover architectures with inherent vulnerabilities. According to extensive evaluation, EVAS demonstrates high evasiveness, transferability, and robustness, raising important considerations for future defense strategies.

Table of Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I extend my sincere gratitude to my Ph.D. advisor, Prof. Ting Wang. He introduced me to the realm of computer security research and initiating the academic journey of my life. His guidance encouraged me to approach problems with a broader perspective, fostering numerous insightful discussions. Prof. Wang's supervision was pivotal to the successful completion of the projects in this dissertation, and without his support, the advancements made in the technical committee would not have been possible.

I also want to convey my appreciation to the other esteemed members of my committee, namely include Prof. Fenglong Ma and Prof. Xiang Zhang from IST, as well as Dr. Sencun Zhu from the College of Engineering. Their constructive feedback on my proposals and engaging discussions played a significant role in shaping this dissertation and preparing for my final defense.

Throughout my Ph.D. program, I received invaluable advice from my lab mates both academically and personally. Special thanks go to Xinyang Zhang, Zhaohan Xi, and Changjiang Li for their technical insights and critiques in the realm of deep learning security and model interpretability. I am also grateful to Zheng Zhang, a crucial collaborator who provided substantial support for my projects.

In conclusion, I want to express my deepest appreciation to friends, family, and everyone else who contributed to making this journey enjoyable and exciting. Their kindness and emotional support were indispensable in achieving success during the Ph.D. program.

This work was partially supported by NSF 1846151, NSF 1910546, NSF 1953813, and NSF 1953893. The findings and conclusions in this dissertation do not necessarily reflect the view of the funding agency.

Chapter 1 Introduction

The abrupt advances in deep learning have led to breakthroughs in a number of long-standing machine learning tasks (e.g., image classification [3], natural language processing [4], and even playing Go [5]), enabling scenarios previously considered strictly experimental. 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 [6]. On the upside, this "plug-and-play" paradigm greatly simplifies the development cycles [7]. On the downside, as most pre-trained DNNs are contributed by untrusted third parties [8], their lack of standardization or regulation entails profound security implications. It is now well known that deep learning systems are inherently vulnerable to adversarial manipulations, which significantly hinders their use in security-critical domains, such as autonomous driving, video surveillance, web content filtering, and biometric authentication.

Throughout my doctoral research, my colleagues and I made significant strides in comprehending and addressing neural backdoors. Within this dissertation, I aim to elucidate the core principles surrounding neural backdoors and delineate our contributions toward resolving existing challenges pertaining to the security of DL systems.

1.1 Overview of Neural Backdoors

Two primary attack vectors have been considered in the literature. (i) Adversarial inputs – typically through perturbing a benign input x, the adversary crafts an adversarial version x_* which deceives the target DNN f at inference time [9–12]. (ii) Poisoned models – during training, the adversary builds malicious functions into f, such that the

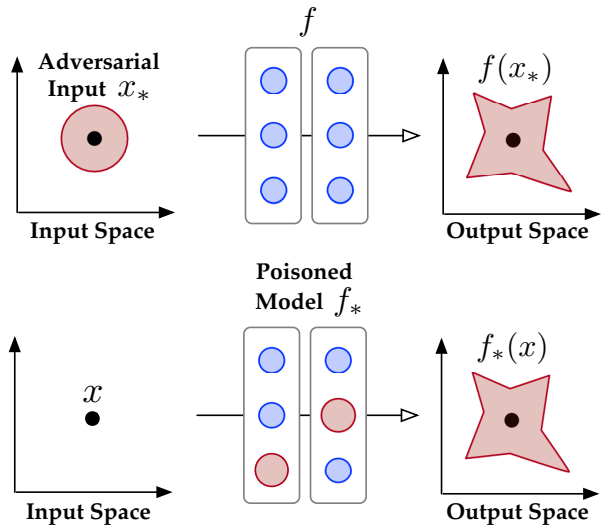

Figure 1.1: "Duality" of adversarial inputs and poisoned models.

poisoned DNN f_* misbehaves on one (or more) pre-defined input(s) x [6, 13–15]. As illustrated in Figure 1.1, the two attack vectors share the same aim of forcing the DNN to misbehave on pre-defined inputs, yet through different routes: one perturbs the input and the other modifies the model. There are attacks (e.g., backdoor attacks [16, 17]) that leverage the two attack vectors simultaneously: the adversary modifies f to be sensitive to pre-defined trigger patterns (e.g., specific watermarks) during training and then generates trigger-embedded inputs at inference time to cause the poisoned model f_* to malfunction.

Prior work has intensively studied the two attack vectors separately [6,9–15]; yet, there is still a lack of understanding about their fundamental connections. First, it remains unclear what the vulnerability to one attack implies for the other. Revealing such implications is important for developing effective defenses against both attacks. Further, the adversary may exploit the two vectors together (e.g., backdoor attacks [16, 17]), or multiple adversaries may collude to perform coordinated attacks. It is unclear how the two vectors may interact with each other and how their interactions may influence the attack dynamics. Understanding such interactions is critical for building effective defenses against coordinated attacks. Finally, studying the two attack vectors within a unified framework is essential for assessing and mitigating the holistic vulnerabilities of DNNs deployed in practice, in which multiple attacks may be launched simultaneously.

In particular, pre-trained DNNs can be exploited to launch *neural backdoor* attacks [16,18,19], one primary 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 met but to function normally otherwise, which can result in consequential damages in security-sensitive domains [20–22].

Motivated by this, intensive research has led to a plethora of attacks that craft trojan model via exploiting properties such as neural activation patterns [14, 16, 18, 23–25] and defenses that mitigate trojan models during inspection [26–31] or detect trigger inputs at inference [32–35].

1.2 Contributions

Following the fundamental background on neural backdoor security, this dissertation presents a series of contributions, each detailed in its own chapter:

- IMC **Attack**. In Chapter 3, we propose IMC, a new class of attacks that co-optimizes inputs and models simultaneously. Specifically, (i) we develop a new attack model that jointly optimizes adversarial inputs and poisoned models; (ii) with both analytical and empirical evidence, we reveal that there exist intriguing "mutual reinforcement" effects between the two attack vectors – leveraging one vector significantly amplifies the effectiveness of the other; (iii) we demonstrate that such effects enable a large design spectrum for the adversary to enhance the existing attacks that exploit both vectors (e.g., backdoor attacks), such as maximizing the attack evasiveness with respect to various detection methods; (iv) finally, we discuss potential countermeasures against such optimized attacks and their technical challenges.

- TROJANZOO. In Chapter 4, we demonstrate TROJANZOO, the first open-source platform for evaluating neural backdoor attacks/defenses in a unified, holistic, and practical manner. Thus far, focusing on the computer vision domain, it has incorporated 8 representative attacks, 14 state-of-the-art defenses, 6 attack performance metrics, 10 defense utility metrics, as well as rich tools for in-depth analysis of the attack-defense interactions. Leveraging TROJANZOO, we conduct a systematic study on the existing attacks/defenses, unveiling their complex design spectrum: both manifest intricate trade-offs among multiple desiderata (e.g., the effectiveness, evasiveness, and transferability of attacks). We further explore improving the existing attacks/defenses, leading to a number of interesting findings: (i) one-pixel triggers often suffice; (ii) training from scratch often outperforms perturbing benign models to craft trojan models; (iii) optimizing triggers and

trojan models jointly greatly improves both attack effectiveness and evasiveness; (iv) individual defenses can often be evaded by adaptive attacks; and (v) exploiting model interpretability significantly improves defense robustness.

- **AutoML Vulnerabilities**. In Chapter 5, we show that compared with their manually designed counterparts, NAS-generated models tend to suffer greater vulnerability to various malicious attacks (*e.g.*, adversarial evasion, model poisoning, and functionality stealing). Further, with both empirical and analytical evidence, we provide possible explanations for such phenomena: given the prohibitive search space and training cost, most NAS methods favor models that converge fast at early training stages; this preference results in architectural properties associated with attack vulnerability (*e.g.*, high loss smoothness and low gradient variance). Our findings not only reveal the relationships between model characteristics and attack vulnerability but also suggest the inherent connections underlying different attacks. Finally, we discuss potential remedies to mitigate such drawbacks, including increasing cell depth and suppressing skip connects.

- **Evas Attack** In Chapter 6, we present EVAS, a new attack that leverages NAS to find neural architectures with inherent backdoors and exploits such vulnerability using input-aware triggers. Compared with existing attacks, EVAS demonstrates many interesting properties: (*i*) it does not require polluting training data or perturbing model parameters; (*ii*) it is agnostic to downstream fine-tuning or even re-training from scratch; (*iii*) it naturally evades defenses that rely on inspecting model parameters or training data. With extensive evaluation on benchmark datasets, we show that EVAS features high evasiveness, transferability, and robustness, thereby expanding the adversary's design spectrum. We further characterize the mechanisms underlying EVAS, which are possibly explainable by architecture-level "shortcuts" that recognize trigger patterns.

1.3 Roadmap

The remainder of this dissertation is structured as follows. Chapter 2 reviews introduces essential concepts and relevant literature on the security of DL systems. In Chapter 3, Chapter 4, Chapter 5 and Chapter 6, we delve into the contributions outlined in § 1.2. Finally, Chapter 7 provides the concluding remarks for this dissertation.

Chapter 2
Background

2.1 Fundamentals

We begin by introducing a set of fundamental concepts and assumptions.

2.1.1 Deep Neural Networks

Deep neural networks (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. We primarily consider a predictive setting, in which a DNN f (parameterized by θ) encodes a function $f\colon \mathcal{X} \to \mathcal{Y}$. Given an input $x \in \mathcal{X}$, f predicts a nominal variable $f(x;\theta)$ ranging over a set of pre-defined classes $\mathcal{Y}$.

We consider DNNs obtained via supervised learning. To train a model f, the training algorithm uses a training set $\mathcal{D}$, of which each instance $(x, y) \in \mathcal{D} \subset \mathcal{X} \times \mathcal{Y}$ comprises an input x and its ground-truth class y. The algorithm determines the best parameter configuration θ for f via optimizing a loss function $\ell(f(x;\theta), y)$ (e.g., the cross entropy of y and $f(x;\theta)$), which is typically implemented using stochastic gradient descent or its variants [36].

2.1.2 Attack Vectors

DNNs are inherently susceptible to malicious manipulations. In particular, two primary attack vectors have been considered in the literature, namely, adversarial inputs and poisoned models.

2.1.2.1 Adversarial Inputs

Adversarial inputs are maliciously crafted samples to deceive target DNNs at inference time. An adversarial input x_* is typically generated by perturbing a benign input $x_\circ$ to change its classification to a target class t desired by the adversary (e.g., pixel perturbation [37] or spatial transformation [38]). To ensure the attack evasiveness, the perturbation is often constrained to a *feasible set* (e.g., a norm ball $\mathcal{F}_\epsilon(x_\circ) = \{x | \|x - x_\circ\|_\infty \leq \epsilon\}$). Formally, the attack is formulated as the optimization objective:

$$x_* = \arg \min_{x \in \mathcal{F}_\epsilon(x_\circ)} \ell(x, t; \theta_\circ) \tag{2.1}$$

where the loss function measures the difference between f's prediction $f(x; \theta_\circ)$ and the adversary's desired classification t.

Eq. 2.1 can be solved in many ways. For instance, `FGSM` [10] uses one-step descent along ℓ's gradient sign direction, `PGD` [37] applies a sequence of projected gradient descent steps, while `C&W` [12] solves Eq. 2.1 with iterative optimization.

2.1.2.2 Poisoned Models

Poisoned models are adversely forged DNNs that are embedded with malicious functions (i.e., misclassification of target inputs) during training.

This attack can be formulated as perturbing a benign DNN $\theta_\circ$ to a poisoned version θ_*.[1] To ensure its evasiveness, the perturbation is often constrained to a feasible set $\mathcal{F}_\delta(\theta_\circ)$ to limit the impact on non-target inputs. For instance, $\mathcal{F}_\delta(\theta_\circ) = \{\theta | \mathbb{E}_{x \in \mathcal{R}}[|f(x; \theta) - f(x; \theta_\circ)|] \leq \delta\}$ specifies that the expected difference between $\theta_\circ$ and θ_*'s predictions regarding the inputs in a reference set $\mathcal{R}$ stays below a threshold δ. Formally, the adversary attempts to optimize the objective function:

$$\theta_* = \arg \min_{\theta \in \mathcal{F}_\delta(\theta_\circ)} \mathbb{E}_{x_\circ \in \mathcal{T}} \left[\ell(x_\circ, t_{x_\circ}; \theta)\right] \tag{2.2}$$

where $\mathcal{T}$ represents the set of target inputs, $t_{x_\circ}$ denotes $x_\circ$'s classification desired by the adversary, and the loss function is defined similarly as in Eq. 2.1.

In practice, Eq. 2.2 can be solved through either polluting training data [14–16] or modifying benign DNNs [6, 17]. For instance, `StingRay` [14] generates poisoning data by perturbing benign inputs close to $x_\circ$ in the feature space; `PoisonFrog` [15] synthesizes

[1] Note that below we use $\theta_\circ$ (θ_*) to denote both a DNN and its parameter configuration. Also note that the benign model $\theta_\circ$ is independent of the target benign input $x_\circ$.

poisoning data close to $x_\circ$ in the feature space but perceptually belonging to t in the input space; while `ModelReuse` [6] directly perturbs the DNN parameters to minimize $x_\circ$'s distance to a representative input from t in the feature space.

2.1.3 Neural backdoor attacks

With the increasing use of DNN models in security-sensitive domains, the adversary is strongly incentivized to forge malicious FEs as attack vectors and lure victim users to re-use them during system development [16]. Specifically, through a malicious FE, the backdoor attack infects the target model with malicious functions desired by the adversary, which are activated once pre-defined conditions ("triggers") are present. We refer to such infected models as "trojan models". 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.

Next we present a unified attack model that subsumes most existing neural backdoor attacks and discuss the adversary's design spectrum.

Trigger mixing operator – For given trigger r, the operator $\oplus$ mixes a clean input $x_\circ \in \mathbb{R}^n$ with r to generate a trigger input $x_\circ \oplus r$. Typically, r comprises three parts: (*i*) mask $m \in \{0,1\}^n$ specifies where r is applied (*i.e.*, $x_\circ$'s i-th feature $x_{\circ i}$ 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 $p(x_\circ) \in \mathbb{R}^n$ specifies r's color intensity, which can be a constant, randomly drawn from a distribution (*e.g.*, by perturbing a template), or dependent on $x_\circ$ [39]. Formally, the trigger embedding operator is defined as:

$$x_\circ \oplus r = (1-m) \odot [(1-\alpha)x + \alpha p(x_\circ)] + m \odot x_\circ \tag{2.3}$$

where $\odot$ denotes element-wise multiplication.

Attack objectives – The trojan model satisfies that with high probability, (*i*) trigger inputs are classified to the target class desired by the adversary and (*ii*) clean input are still correctly classified. Formally, the adversary forges the malicious FE by optimizing the following objective:

$$\min_{r \in \mathcal{R}, \theta} \mathbb{E}_{(x,y)\in\mathcal{T}} \left[\ell(f_\theta(x \oplus r), t) + \lambda \ell(f_\theta(x), y)\right] \tag{2.4}$$

where $\mathcal{T}$ represents the training set, t denotes the target class, and trigger r is selected from the feasible set $\mathcal{R}$ (which constrains r's shape, transparency, and/or pattern). Intuitively,

the first and second terms describe (i) and (ii), respectively, and the hyper-parameter λ balances the two objectives.

Adversary's knowledge – If the downstream classifier h is known to the adversary, f shares the same architecture with the model $h \circ g$ used by the victim; otherwise, the adversary may resort to a surrogate classifier h^* (*i.e.*, $h^* \circ g$) or re-define the loss $\ell(f(x \oplus r), t)$ in terms of latent representations [19, 25] as $\Delta(g(x \oplus r), \phi_t)$, that is, the difference(*e.g.*, MSE loss) between $g(x \oplus r)$ and ϕ_t, where ϕ_t is the average latent representation of class t.

Malicious FE training – To optimize Eq. 2.4, one may perturb a benign FE [14,18] or train the malicious FE from scratch (details in § 4.4). To satisfy the trigger constraint, r can be fixed [16], partially defined [18] (*e.g.*, with its mask fixed), or optimized with f jointly [19].

2.1.4 Neural Architecture Search

Deep neural networks (DNNs) represent a class of ML models to learn high-level abstractions of complex data. We assume a predictive setting, in which a DNN f_θ (parameterized by θ) encodes a function $f_\theta : \mathbb{R}^n \to \mathbb{S}^m$, where n and m denote the input dimensionality and the number of classes. Given input x, $f(x)$ is a probability vector (simplex) over m classes.

In this dissertation, we mainly focus on one primary task of AutoML, neural architecture search (NAS), which searches for performant DNN architectures for given tasks [40]. Formally, let $\mathcal{D}$ be the given dataset, $\ell(\cdot, \cdot)$ be the loss function, $\mathcal{F}$ be the functional space of possible models (*i.e.*, search space), the NAS method A searches for a performant DNN f^* via minimizing the following objective:

$$f^* = \arg\min_{f \in \mathcal{F}} \mathbb{E}_{(x,y)\sim\mathcal{D}}\, \ell(f(x), y) \tag{2.5}$$

The existing NAS methods can be categorized according to their search spaces and strategies. In the following, we focus on the space of cell-based architectures [41–45], which repeat the motif of a cell structure in a pre-specified arrangement, and the strategy of differentiable NAS [43, 46, 47], which jointly optimizes the architecture and model parameters using gradient descent, due to their state-of-the-art performance and efficiency. Nevertheless, our discussion generalizes to alternative NAS frameworks.

Without loss of generality, we use *DARTS* [43] as a concrete example to illustrate differentiable NAS. At a high level, *DARTS* searches for two cell structures (*i.e.*, normal

cell and reduction cell) as the basic building blocks of the final architecture. As shown in Figure 5.1, a cell is modeled as a directed acyclic graph, in which each node $x^{(i)}$ is a latent representation and each directed edge (i, j) represents an operation $o^{(i,j)}$ applied on $x^{(i)}$ (*e.g.*, skip connect). Each node is computed based on all its predecessors:

$$x^{(j)} = \sum_{i<j} o^{(i,j)}(x^{(i)}) \tag{2.6}$$

Each cell contains n_{in} input nodes (often $n_{\text{in}} = 2$), n_{out} output nodes (often $n_{\text{out}} = 1$), and n_{mid} intermediate nodes. Each input node takes the output from a preceding cell, the output node aggregates the latent representations from intermediate nodes, while each intermediate node is connected to m preceding nodes (typically $m = n_{\text{in}}$).

To enable gradient-based optimization of the architecture, *DARTS* applies continuous relaxation on the search space. Letting $\mathcal{O}$ be the set of candidate operations, the categorical choice of an operation is reduced to a softmax over $\mathcal{O}$:

$$\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x) \tag{2.7}$$

where $\alpha_o^{(i,j)}$ represents the trainable weight of operation o. At the end of the search, a discrete architecture is obtained by replacing $\bar{o}^{(i,j)}$ with the most likely operation $\arg\max_o \alpha_o^{(i,j)}$.

The search is thus formulated as a bi-level optimization objective function:

$$\min_{\alpha} \ \mathcal{L}_{\text{val}}(\theta^*(\alpha), \alpha) \quad \text{s.t.} \quad \theta^*(\alpha) = \arg\min_{\theta} \ \mathcal{L}_{\text{trn}}(\theta, \alpha) \tag{2.8}$$

where $\mathcal{L}_{\text{trn}}$ and $\mathcal{L}_{\text{val}}$ are the training and validation losses, and $\alpha = \{\alpha^{(i,j)}\}$ and θ denote the architecture and model parameters, respectively. To handle the prohibitive cost of the nested optimization, single-step gradient descent is applied to avoid solving the inner objective exactly.

2.1.5 Other Attack Vulnerabilities

It is known that DNN models are vulnerable to a variety of attacks at both training and inference phases. Here, we highlight the following major attacks.

Adversarial evasion – At inference time, the adversary generates an adversarial input $(x + \delta)$ by modifying a begin one x with imperceptible perturbation δ, to cause the target model f to misbehave [10]. Formally, in a targeted attack, letting t be the

target class desired by the adversary, the attack crafts $(x+\delta)$ by optimizing the following objective:

$$\min_{\delta \in \mathcal{B}_\epsilon} \ell(f(x+\delta), t) \tag{2.9}$$

where $\mathcal{B}_\epsilon$ specifies the set of allowed perturbation(*e.g.*, a ℓ_∞-norm ball of radius ϵ). Eq. 2.9 is often solved using projected gradient descent [48] or general-purpose optimizers [49].

Model poisoning – The adversary aims to modify a target model f's behavior (*e.g.*, overall performance degradation or misclassification of specific inputs) via polluting its training data [50]. For instance, to cause the maximum accuracy drop, letting $\mathcal{D}_{\text{trn}}$ and $\mathcal{D}_{\text{tst}}$ be the training and testing sets and f be the target model, the attack crafts a set of poisoning inputs $\mathcal{D}_{\text{pos}}$ by optimizing the the following objective (note: the adversary may not have access to $\mathcal{D}_{\text{trn}}$, $\mathcal{D}_{\text{tst}}$, or f):

$$\begin{aligned} &\max \mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{tst}}} \ell(f_{\theta^*}(x), y) \\ &\text{s.t. } \theta^* = \arg\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{trn}}\cup\mathcal{D}_{\text{pos}}} \ell(f_\theta(x), y) \end{aligned} \tag{2.10}$$

Backdoor injection – During training, via perturbing a benign model f, the adversary forges a trojan model f_{θ^*} sensitive to a trigger pattern r^*, which is used in the downstream task by the victim; at inference time, the adversary invokes the malicious function by feeding trigger-embedded input $x + r^*$. Formally, letting $\mathcal{D}_{\text{trn}}$ be the training data and t be the target class desired by the adversary, the attack generates a trojan model parameterized by θ^* and its associated trigger r^* by optimizing the following objective:

$$\min_{r\in\mathcal{R}_\gamma,\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{trn}}}[\ell(f_\theta(x), y) + \lambda \ell(f_\theta(x+r), t)] \tag{2.11}$$

where r^* is selected from a feasible set $\mathcal{R}_\gamma$ (*e.g.*, a 3×3 patch with transparency γ), the first term enforces all clean inputs to be correctly classified, the second term ensures all trigger inputs to be misclassified into t, and the hyper-parameter λ balances the two objectives.

Functionality stealing – In functionality stealing [51], the adversary aims to construct a replicate model $\hat{f}$ (parameterized by θ^*) functionally similar to a victim model f via probing f through a black-box query interface. Notably, it is different from model stealing [52] that aims to re-construct f in terms of architectures or parameters. Formally, letting $\mathcal{D}$ be the underlying data distribution, the attack generates the query-prediction set $\mathcal{Q}$ (note: the adversary may not have the labeling of $\mathcal{D}$, has only query

access to f, and is typically constrained by the number of queries to be issued), which optimizes the following objective:

$$\begin{aligned} &\min \mathbb{E}_{x\sim\mathcal{D}}\, \ell(\hat{f}_{\theta^*}(x), f(x)) \\ &\text{s.t. } \theta^* = \arg\min_{\theta} \mathbb{E}_{(x,f(x))\sim\mathcal{Q}}\, \ell(\hat{f}_{\theta}(x), f(x)) \end{aligned} \tag{2.12}$$

Different functionality stealing attacks differ in how $\mathcal{Q}$ is constructed (*e.g.*, random or adaptive construction).

Membership inference – In membership inference [53], given input x and model's prediction $f(x)$, the adversary attempts to predict a binary variable b indicating whether x is included in f's training data: $b \leftarrow \mathcal{A}(x, f)$. The effectiveness of membership inference relies on f's performance gap with respect to the training data $\mathcal{D}_{\text{trn}}$ and testing data $\mathcal{D}_{\text{tst}}$. The adversary may exploit this performance gap by thresholding the confidence score of $f(x)$ if it is available, or estimating other signals (*e.g.*, x's distance to the nearest decision boundary) if only the label of $f(x)$ is provided [54].

2.2 Related Work

2.2.1 ML Security

With their increasing use in security-sensitive domains, DNNs are becoming the new targets of malicious manipulations [55]. Two primary attack vectors have been considered in the literature: adversarial inputs and poisoned models.

2.2.1.1 Adversarial Inputs

The existing research on adversarial inputs is divided in two campaigns.

One line of work focuses on developing new attacks against DNNs [9–12], with the aim of crafting adversarial samples to force DNNs to misbehave. The existing attacks can be categorized as untargeted (in which the adversary desires to simply force misclassification) and targeted (in which the adversary attempts to force the inputs to be misclassified into specific classes).

Another line of work attempts to improve DNN resilience against adversarial attacks by devising new training strategies (e.g., adversarial training) [56–59] or detection mechanisms [1, 60–62]. However, the existing defenses are often penetrated or circumvented by even stronger attacks [63, 64], resulting in a constant arms race between the attackers

and defenders.

2.2.1.2 Poisoned Models

The poisoned model-based attacks can be categorized according to their target inputs. In the poisoning attacks, the target inputs are defined as non-modified inputs, while the adversary's goal is to force such inputs to be misclassified by the poisoned DNNs [6, 13–15, 65]. In the backdoor attacks, specific trigger patterns (e.g., a particular watermark) are pre-defined, while the adversary's goal is to force any inputs embedded with such triggers to be misclassified by the poisoned models [16,17]. Note that compared with the poisoning attacks, the backdoor attacks leverage both adversarial inputs and poisoned models.

The existing defense methods against poisoned models mostly focus on the backdoor attacks, which, according to their strategies, can be categorized as: (i) cleansing potential contaminated data at the training stage [66], (ii) identifying suspicious models during model inspection [26, 27, 67], and (iii) detecting trigger-embedded inputs at inference time [68–71].

Despite the intensive research on adversarial inputs and poisoned models in parallel, there is still a lack of understanding about their inherent connections. This work bridges this gap by studying the two attack vectors within a unified framework and providing a holistic view of the vulnerabilities of DNNs deployed in practice.

2.2.2 Backdoor Benchmarks

Some recent studies have surveyed neural backdoor attacks/defenses (*e.g.*, [72]); yet, none of them provides benchmark implementation or empirical evaluation to explore their strengths/limitations. Compared with the rich collection of platforms for adversarial attacks/defenses (*e.g.*, CleverHans [73], DeepSec [64], and AdvBox [74]), only few platforms currently support evaluating neural backdoors. For instance, Art [75] integrates 3 attacks and 3 defenses.

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.

The work closest to ours is perhaps TrojAI [76], which is a contest platform for model-inspection defenses against neural backdoors. While compared with TrojanZoo, TrojAI provides a much larger pool of trojan models (over 10K) across different modalities (*e.g.*, vision and NLP), TrojanZoo departs from TrojAI in majors aspects and offers its unique value. (*i*) Given its contest-like setting, TrojAI is a closed platform focusing on evaluating model-inspection defenses (*i.e.*, detecting trojan models) against fixed attacks, while TrojanZoo is an open platform that provides extensible datasets, models, attacks, and defenses. Thus, TrojanZoo may serve the needs ranging from conducting comparative studies of existing attacks/defenses to exploring and evaluating new attacks/defenses. (*ii*) While TrojAI focuses on model-inspection defenses, TrojanZoo integrates four major defense categories. (*iii*) In TrojAI, for its purpose, the concrete attacks behind the trojan models are unknown, which makes it challenging to assess the strengths/limitations of given defenses with respect to different attacks, while in TrojanZoo one may directly evaluate such interactions. (*iv*) As the attacks are fixed in TrojAI, one may not evaluate adaptive attacks. (*v*) The main metric used in TrojAI is the accuracy that defenses successfully detect trojan models, while TrojanZoo provides a much richer set of metrics to characterize attacks/defenses.

2.2.3 Neural Architecture Search

The existing NAS methods can be categorized along three dimensions: search space, search strategy, and performance measure.

The search space defines the possible set of candidate models. Early NAS methods focus on the chain-of-layer structure [77], consisting of a sequence of layers. Motivated by that hand-crafted models often consist of repeated motifs, recent methods propose to search for such cell structures, including the connection topology and the corresponding operation on each connection [41–45].

The search strategy defines how to efficiently explore the pre-defined search space. Early NAS methods rely on either random search [78] or Bayesian optimization [79], which are often limited in terms of search efficiency and model complexity. More recent work mainly uses the approaches of reinforcement learning (RL) [77] or neural evolution [43, 45]. Empirically, neural evolution- and RL-based methods tend to perform comparably well [45].

The performance measure evaluates the candidate models and guides the search process. Recently, one-shot NAS has emerged as a popular performance measure. It

considers all candidate models as different sub-graphs of a super-net (*i.e.*, the one-shot model) and shares weights between candidate models [42–44]. The differentiable NAS methods considered in this dissertation belong to this category. Different one-shot methods differ in how the one-shot model is trained. For instance, *DARTS* [43] optimizes the one-shot model with continuous relaxation of the search space.

Chapter 3
IMC: Input Model Co-optimization Attack

3.1 Introduction

In this chapter, we seek to answer the following research questions.

- RQ1 – *What are the fundamental connections between adversarial inputs and poisoned models?*
- RQ2 – *What are the dynamic interactions between the two attack vectors if they are applied together?*
- RQ3 – *What are the implications of such interactions for the adversary to optimize the attack strategies?*
- RQ4 – *What are the potential countermeasures to defend against such enhanced attacks?*

3.1.1 Our Work

This work represents a solid step towards answering the key questions above. We cast adversarial inputs and poisoned models within a unified framework, conduct a systematic study of their interactions, and reveal the implications for DNNs' holistic vulnerabilities, leading to the following interesting findings.

RA1 – We develop a new attack model that jointly optimizes adversarial inputs and poisoned models. With this framework, we show that there exists an intricate "duality" relationship between the two attack vectors. Specifically, they represent different routes to achieve the same aim (i.e., misclassification of the target input): one perturbs the input at the cost of "fidelity" (whether the attack retains the original input's perceptual

quality), while the other modifies the DNN at the cost of "specificity" (whether the attack influences non-target inputs).

RA2 – Through empirical studies on benchmark datasets and in security-critical applications (e.g., skin cancer screening [80]), we reveal that the interactions between the two attack vectors demonstrate intriguing "mutual-reinforcement" effects: when launching the unified attack, leveraging one attack vector significantly amplifies the effectiveness of the other (i.e., "the whole is much greater than the sum of its parts"). We also provide analytical justification for such effects under a simplified setting.

RA3 – Further, we demonstrate that the mutual reinforcement effects entail a large design spectrum for the adversary to optimize the existing attacks that exploit both attack vectors (e.g., backdoor attacks). For instance, leveraging such effects, it is possible to enhance the attack evasiveness with respect to multiple defense mechanisms (e.g., adversarial training [37]), which are designed to defend against adversarial inputs or poisoned models alone; it is also possible to enhance the existing backdoor attacks (e.g., [16,17]) with respect to both human vision (in terms of trigger size and transparency) and automated detection methods (in terms of input and model anomaly).

RA4 – Finally, we demonstrate that to effectively defend against such optimized attacks, it is necessary to investigate the attacks from multiple complementary perspectives (i.e., fidelity and specificity) and carefully account for the mutual reinforcement effects in applying the mitigation solutions, which point to a few promising research directions.

To our best knowledge, this work represents the first systematic study of adversarial inputs and poisoned models within a unified framework. We believe our findings deepen the holistic understanding about the vulnerabilities of DNNs in practical settings and shed light on developing more effective countermeasures.[1]

3.1.2 Threat Models

We assume a threat model wherein the adversary is able to exploit both attack vectors. During training, the adversary forges a DNN embedded with malicious functions. This poisoned model is then incorporated into the target deep learning system through either system development or maintenance [6, 17]. At inference time, the adversary further generates adversarial inputs to trigger the target system to malfunction.

This threat model is realistic. Due to the increasing model complexity and training cost, it becomes not only tempting but also necessary to reuse pre-trained models [6,16,17].

[1]The source code and data are released at `https://github.com/alps-lab/imc`.

Besides reputable sources (e.g., Google), most pre-trained DNNs on the market (e.g., [8]) are provided by untrusted third parties. Given the widespread use of deep learning in security-critical domains, adversaries are strongly incentivized to build poisoned models, lure users to reuse them, and trigger malicious functions via adversarial inputs during system use. The backdoor attacks [16, 17, 25] are concrete instances of this threat model: the adversary makes DNN sensitive to certain trigger patterns (e.g., watermarks), so that any trigger-embedded inputs are misclassified at inference. Conceptually, one may regard the trigger as a universal perturbation r [81]. To train the poisoned model θ_*, the adversary samples inputs $\mathcal{T}$ from the training set $\mathcal{D}$ and enforces the trigger-embedded input $(x_\circ + r)$ for each $x_\circ \in \mathcal{T}$ to be misclassified to the target class t. Formally, the adversary optimizes the objective function:

$$\min_{r \in \mathcal{F}_\epsilon, \theta \in \mathcal{F}_\delta(\theta_\circ)} \mathbb{E}_{x_\circ \in \mathcal{T}} \left[\ell(x_\circ + r, t; \theta) \right] \tag{3.1}$$

where both the trigger and poisoned model need to satisfy the evasiveness constraints. Nonetheless, in the existing backdoor attacks, Eq. 3.1 is often solved in an ad hoc manner, resulting in suboptimal triggers and/or poisoned models. For example, `TrojanNN` [17] pre-defines the trigger shape (e.g., Apple logo) and determines its pixel values in a preprocessing step. We show that the existing attacks can be significantly enhanced within a rigorous optimization framework (details in § 3.4).

3.2 A Unified Attack Framework

Despite their apparent variations, adversarial inputs and poisoned models share the same objective of forcing target DNNs (modified or not) to misclassify pre-defined inputs (perturbed or not). While intensive research has been conducted on the two attack vectors in parallel, little is known about their fundamental connections.

3.2.1 Attack Objectives

To bridge this gap, we study the two attack vectors using *input model co-optimization* (IMC), a unified attack framework. Intuitively, within IMC, the adversary is allowed to perturb each target input $x_\circ \in \mathcal{T}$ and/or to poison the original DNN $\theta_\circ$, with the objective of forcing the adversarial version x_* of each $x_\circ \in \mathcal{T}$ to be misclassified to a target class $t_{x_\circ}$ by the poisoned model θ_*.

Formally, we define the unified attack model by integrating the objectives of Eq. 2.1, Eq. 2.2, and Eq. 3.1:

$$\min_{\theta \in \mathcal{F}_\delta(\theta_\circ)} \mathbb{E}_{x_\circ \in \mathcal{T}} \left[\min_{x \in \mathcal{F}_\epsilon(x_\circ)} \ell\left(x, t_{x_\circ}; \theta\right) \right] \tag{3.2}$$

where the different terms define the adversary's multiple desiderata:

- The loss ℓ quantifies the difference of the model prediction and the classification desired by the adversary, which represents the attack *efficacy* – whether the attack successfully forces the DNN to misclassify each input $x_\circ \in \mathcal{T}$ to its target class $t_{x_\circ}$.
- The constraint $\mathcal{F}_\epsilon$ bounds the impact of input perturbation on each target input, which represents the attack *fidelity* – whether the attack retains the perceptual similarity of each adversarial input to its benign counterpart.
- The constraint $\mathcal{F}_\delta$ bounds the influence of model perturbation on non-target inputs, which represents the attack *specificity* – whether the attack precisely directs its influence to the set of target inputs $\mathcal{T}$ only.

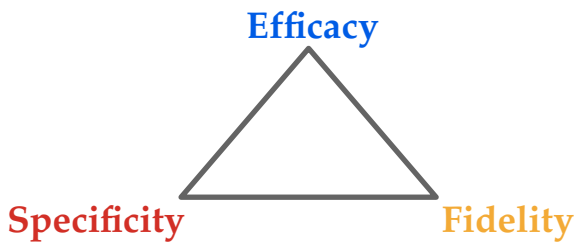

Figure 3.1: Adversary's multiple objectives.

This formulation subsumes many attacks in the literature. Specifically, (i) in the case of $\delta = 0$ and $|\mathcal{T}| = 1$, Eq. 3.2 is instantiated as the adversarial attack; (ii) in the case of $\epsilon = 0$, Eq. 3.2 is instantiated as the poisoning attack, which can be either a single target ($|\mathcal{T}| = 1$) or multiple targets ($|\mathcal{T}| > 1$); and (iii) in the case that a universal perturbation $(x - x_\circ)$ is defined for all the inputs $\{x_\circ \in \mathcal{T}\}$ and all the target classes $\{t_{x_\circ}\}$ are fixed as t, Eq. 3.2 is instantiated as the backdoor attack. Also note that this formulation does not make any assumptions regarding the adversary's capability or resource (e.g., access to the training or inference data), while it is solely defined in terms of the adversary's objectives.

Interestingly, the three objectives are tightly intertwined, forming a triangle structure, as illustrated in Figure 3.1. We have the following observations.

- It is impossible to achieve all the objectives simultaneously. To attain attack efficacy (i.e., launching a successful attack), it requires either perturbing the input (i.e., at the cost of fidelity) or modifying the model (i.e., at the cost of specificity).
- It is feasible to attain two out of the three objectives at the same time. For instance, it is trivial to achieve both attack efficacy and fidelity by setting $\epsilon = 0$ (i.e., only model perturbation is allowed).
- With one objective fixed, it is possible to balance the other two. For instance, with fixed attack efficacy, it allows to trade between attack fidelity and specificity.

Next, by casting the attack vectors of adversarial inputs and poisoned models within the IMC framework, we reveal their inherent connections and explore the dynamic interactions among the attack efficacy, fidelity, and specificity.

3.2.2 Attack Implementation

Recall that IMC is formulated in Eq. 3.2 as optimizing the objectives over both the input and model. While it is impractical to exactly solve Eq. 3.2 due to its non-convexity and non-linearity, we reformulate Eq. 3.2 to make it amenable for optimization. To ease the discussion, in the following, we assume the case of a single target input $x_\circ$ in the target set $\mathcal{T}$ (i.e., $|\mathcal{T}| = 1$), while the generalization to multiple targets is straightforward. Further, when the context is clear, we omit the reference input $x_\circ$, benign model $\theta_\circ$, and target class t to simplify the notations.

3.2.2.1 Reformulation

The constraints $\mathcal{F}_\epsilon(x_\circ)$ and $\mathcal{F}_\delta(\theta_\circ)$ in Eq. 3.2 essentially bound the fidelity and specificity losses. The fidelity loss $\ell_{\mathrm{f}}(x)$ quantifies whether the perturbed input x faithfully retains its perceptual similarity to its benign counterpart $x_\circ$ (e.g., $\|x - x_\circ\|$); the specificity loss $\ell_{\mathrm{s}}(\theta)$ quantifies whether the attack impacts non-target inputs (e.g., $\mathbb{E}_{x \in \mathcal{R}}\left[|f(x;\theta) - f(x;\theta_\circ)|\right]$). According to optimization theory [82], specifying the bounds ϵ and δ on the input and model perturbation amounts to specifying the hyper-parameters λ and ν on the fidelity and specificity losses (the adversary is able to balance different objectives by controlling λ and ν). Eq. 3.2 can therefore be re-formulated as follows:

$$\boxed{\min_{x,\theta} \ell(x;\theta) + \lambda \ell_{\mathrm{f}}(x) + \nu \ell_{\mathrm{s}}(\theta)} \tag{3.3}$$

Nonetheless, it is still difficult to directly optimize Eq. 3.3 given that the input x and the model θ are mutually dependent on each other. Note that however ℓ_{f} does not depend on θ while ℓ_{s} does not depend on x. We thus further approximate Eq. 3.3 with the following bi-optimization formulation:

$$\begin{cases} x_* = \arg\min_x \ell(x;\theta_*) + \lambda\ell_{\mathrm{f}}(x) \\ \theta_* = \arg\min_\theta \ell(x_*;\theta) + \nu\ell_{\mathrm{s}}(\theta) \end{cases} \tag{3.4}$$

3.2.2.2 Optimization

This formulation naturally leads to an optimization procedure that alternates between updating the input x and updating the model θ, as illustrated in Figure 3.2. Specifically, let $x^{(k)}$ and $\theta^{(k)}$ be the perturbed input and model respectively after the k-th iteration. The $(k+1)$-th iteration comprises two operations.

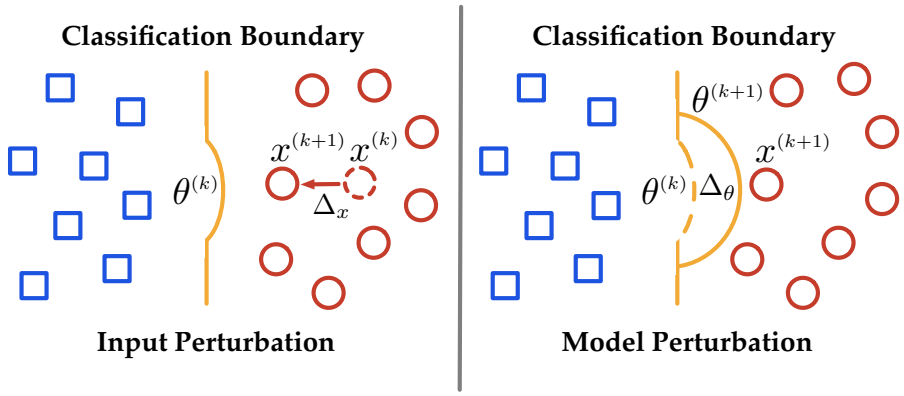

Figure 3.2: IMC alternates between two operations: (i) input perturbation to update the adversarial input x, and (ii) model perturbation to update the poisoned model θ.

Input Perturbation – In this step, with the model $\theta^{(k)}$ fixed, it updates the perturbed input by optimizing the objective:

$$x^{(k+1)} = \arg\min_x \ \ell(x;\theta^{(k)}) + \lambda\ell_{\mathrm{f}}(x) \tag{3.5}$$

In practice, this step can be approximated by applying an off-the-shelf optimizer (e.g., Adam [83]) or solved partially by applying gradient descent on the objective function. For instance, in our implementation, we apply projected gradient descent (`PGD` [37]) as the update operation:

$$x^{(k+1)} = \Pi_{\mathcal{F}_\epsilon(x_\circ)}\left(x^{(k)} - \alpha\,\mathrm{sign}\left(\nabla_x \ell\left(x^{(k)};\theta^{(k)}\right)\right)\right)$$

where Π is the projection operator, $\mathcal{F}_\epsilon(x_\circ)$ is the feasible set (i.e., $\{x | \|x - x_\circ\| \leq \epsilon\}$), and α is the learning rate.

Model Perturbation – In this step, with the input $x^{(k+1)}$ fixed, it searches for the model perturbation by optimizing the objective:

$$\theta^{(k+1)} = \arg\min_{\theta} \ \ell(x^{(k+1)}; \theta) + \nu \ell_{\mathrm{s}}(\theta) \tag{3.6}$$

In practice, this step can be approximated by running re-training over a training set that mixes the original training data $\mathcal{D}$ and m copies of the current adversarial input $x^{(k+1)}$. In our implementation, m is set to be half of the batch size.

Algorithm 1: IMC Attack

Input: benign input – $x_\circ$; benign model – $\theta_\circ$; target class – t; hyper-parameters – λ, ν
Output: adversarial input – x_*; poisoned model – θ_*

```
// initialization
```
$x^{(0)}, \theta^{(0)}, k \leftarrow x_\circ, \theta_\circ, 0$;
```
// optimization
```
while *not converged yet* **do**
  ```
  // input perturbation
  ```
  $x^{(k+1)} = \arg\min_x \ell(x; \theta^{(k)}) + \lambda \ell_{\mathrm{f}}(x)$;
  ```
  // model perturbation
  ```
  $\theta^{(k+1)} = \arg\min_\theta \ell(x^{(k+1)}; \theta) + \nu \ell_{\mathrm{s}}(\theta)$;
  $k \leftarrow k + 1$;
return $(x^{(k)}, \theta^{(k)})$;

Algorithm 1 sketches the complete procedure. By alternating between input and model perturbation, it finds approximately optimal adversarial input x_* and poisoned model θ_*. Note that designed to study the interactions of adversarial inputs and poisoned models (§ 3.3), Algorithm 1 is only one possible implementation of Eq. 3.2 under the setting of a single target input and both input and model perturbation. To implement other attack variants, one can adjust Algorithm 1 accordingly (§ 3.4). Also note that it is possible to perform multiple input (or model) updates per model (or input) update to accommodate their different convergence rates.

3.2.2.3 Analysis

Next we provide analytical justification for Algorithm 1. As Eq. 3.3 is effectively equivalent to Eq. 3.2, Algorithm 1 approximately solves Eq. 3.3 by alternating between (i) input perturbation – searching for $x_* = \arg\min_{x \in \mathcal{F}_\epsilon(x_\circ)} \ell(x; \theta_*)$ and (ii) model perturbation – searching for $\theta_* = \arg\min_{\theta \in \mathcal{F}_\delta(\theta_\circ)} \ell(x_*; \theta)$. We now show that this implementation

effectively solves Eq. 3.3 (proof deferred to Appendix A).

Proposition 1. Let $x_* \in \mathcal{F}_\epsilon(x_\circ)$ be a minimizer of the function $\min_x \ell(x;\theta)$. If x_* is non-zero, then $\nabla_\theta \ell(x_*;\theta)$ is a proper descent direction for the objective function of $\min_{x\in\mathcal{F}_\epsilon(x_\circ)} \ell(x;\theta)$.

Thus, we can conclude that Algorithm 1 is an effective implementation of the IMC attack framework. It is observed in our empirical evaluation that Algorithm 1 typically converges within less than 20 iterations (details in § 3.3).

3.3 Mutual Reinforcement Effects

Next we study the dynamic interactions between adversarial inputs and poisoned models. With both empirical and analytical evidence, we reveal that there exist intricate "mutual reinforcement" effects between the two attack vectors: (i) leverage effect – with fixed attack efficacy, at slight cost of one metric (i.e., fidelity or specificity), one can disproportionally improve the other metric; (ii) amplification effect – with one metric fixed, at minimal cost of the other, one can greatly boost the attack efficacy.

3.3.1 Study Setting

Datasets

To factor out the influence of specific datasets, we primarily use 4 benchmark datasets:

- CIFAR10 [84] – It consists of 32×32 color images drawn from 10 classes (e.g., 'airplane');
- Mini-ImageNet – It is a subset of the ImageNet dataset [3], which consists of 224×224 (center-cropped) color images drawn from 20 classes (e.g., 'dog');
- ISIC [80] – It represents the skin cancer screening task from the ISIC 2018 challenge, in which given 600×450 skin lesion images are categorized into a 7-disease taxonomy (e.g., 'melanoma');
- GTSRB [85] – It consists of color images of size ranging from 29×30 to 144×48, each representing one of 43 traffic signs.

Note that among these datasets, ISIC and GTSRB in particular represent security-sensitive tasks (i.e., skin cancer screening [80] and traffic sign recognition [86]).

DNNs

We apply ResNet18 [87] to CIFAR10, GTSRB and ImageNet and ResNet101 to ISIC as the reference DNN models. Their top-1 accuracy on the testset of each dataset is summarized in Table 3.1. Using two distinct DNNs, we intend to factor out the influence of individual DNN characteristics (e.g., network capacity).

	CIFAR10	ImageNet	ISIC	GTSRB
Model	ResNet18	ResNet18	ResNet101	ResNet18
Accuracy	95.23%	94.56%	88.18%	99.12%

Table 3.1. Accuracy of benign DNNs on reference datasets.

Attacks

Besides the IMC attack in § 3.2.2, we also implement two variants of IMC (with the same hyper-parameter setting) for comparison: (i) input perturbation only, in which IMC is instantiated as the adversarial attack (i.e., `PGD` [37]), and (ii) model perturbation only, in which IMC is instantiated as the poisoning attack. The implementation details are deferred to Appendix B.

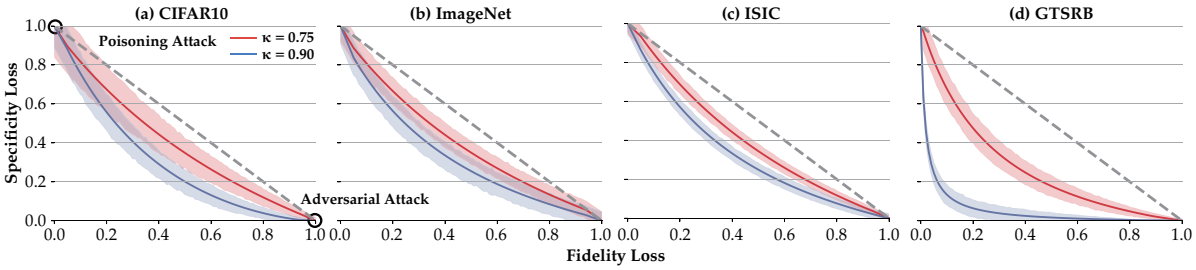

Figure 3.3: Disproportionate trade-off between attack fidelity and specificity.

Measures

We quantify the attack objectives as follows.

Efficacy – We measure the attack efficacy by the misclassification confidence, $f_t(x_*; \theta_*)$, which is the probability that the adversarial input x_* belongs to the target class t as predicted by the poisoned model θ_*. We consider the attack successful if the misclassification confidence exceeds a threshold κ.

Fidelity – We measure the fidelity loss by the L_p-norm of the input perturbation $\ell_{\mathrm{f}}(x_*) \triangleq \|x_* - x_\circ\|_p$. Following previous work on adversarial attacks [10, 12, 37], we use $p = \infty$ by default in the following evaluation.

Specificity – Further, we measure the specificity loss using the difference of the benign and poisoned models on classifying a reference set $\mathcal{R}$. Let $\mathbb{I}_z$ be the indicator function that returns 1 if z is true and 0 otherwise. The specificity loss can be defined as:

$$\ell_{\mathrm{s}}(\theta_*) \triangleq \sum_{x \in \mathcal{R}} \frac{\mathbb{I}_{f(x;\theta_\circ) \neq f(x;\theta_*)}}{|\mathcal{R}|} \tag{3.7}$$

With fixed attack efficacy κ, let (x_*, θ_*) be the adversarial input and poisoned model generated by IMC, and $\bar{x}_*$ and $\bar{\theta}_*$ be the adversarial input and poisoned model given by the adversarial and poisoning attacks respectively. Because the adversarial and poisoning attacks are special variants of IMC, we have $x_* = \bar{x}_*$ if $\theta_* = \theta_\circ$ and $\theta_* = \bar{\theta}_*$ if $x_* = x_\circ$. Thus, in the following, we normalize the fidelity and specificity losses as $\ell_{\mathrm{f}}(x_*)/\ell_{\mathrm{f}}(\bar{x}_*)$ and $\ell_{\mathrm{s}}(\theta_*)/\ell_{\mathrm{s}}(\bar{\theta}_*)$ respectively, both of which are bound to $[0, 1]$. For reference, the concrete specificity losses $\ell_{\mathrm{s}}(\bar{\theta}_*)$ (average accuracy drop) caused by the poisoning attack on each dataset are summarized in Table 3.2.

κ	CIFAR10	ImageNet	ISIC	GTSRB
0.75	0.11%	2.44%	1.53%	0.25%
0.9	0.12%	3.83%	1.62%	0.27%

Table 3.2. Specificity losses (average accuracy drop) caused by poisoning attacks on reference datasets.

3.3.2 Effect I: Leverage Effect

In the first set of experiments, we show that for fixed attack efficacy, with disproportionally small cost of fidelity, it is feasible to significantly improve the attack specificity, and vice versa.

3.3.2.1 Disproportionate Trade-off

For each dataset, we apply the adversarial, poisoning, and IMC attacks against 1,000 inputs randomly sampled from the testset (as the target set $\mathcal{T}$), and use the rest as the reference set $\mathcal{R}$ to measure the specificity loss. For each input of $\mathcal{T}$, we randomly select its target class and fix the required attack efficacy (i.e., misclassification confidence κ). By varying IMC's hyper-parameters λ and ν, we control the importance of fidelity and specificity. We then measure the fidelity and specificity losses for all the successful cases. Figure 3.3 illustrates how IMC balances fidelity and specificity. Across all the datasets and models, we have the following observations.

First, with fixed attack efficacy (i.e., $\kappa = 0.9$), by sacrificing disproportionally small fidelity (i.e., input perturbation magnitude), IMC significantly improves the attack specificity (i.e., accuracy drop on non-target inputs), compared with required by the corresponding poisoning attack. For instance, in the case of GTSRB (Figure 3.3 (d)), as the fidelity loss increases from 0 to 0.05, the specificity loss is reduced by more than 0.48.

Second, this effect is symmetric: a slight increase of specificity loss also leads to significant fidelity improvement, compared with required by the corresponding adversarial attack. For instance, in the case of CIFAR10 (Figure 3.3 (a)), as the specificity loss increases from 0 to 0.1, the specificity loss drops by 0.37.

Third, higher attack efficacy constraint brings more fidelity-specificity trade-off. Observe that across all datasets, GTSRB shows significantly larger curvature, which might be explained by the higher model accuracy(99.12%) and larger number of classes(43).

Leverage Effect

There exists an intricate fidelity-specificity trade-off. At disproportionally small cost of fidelity, it is possible to significantly improve specificity, and vice versa.

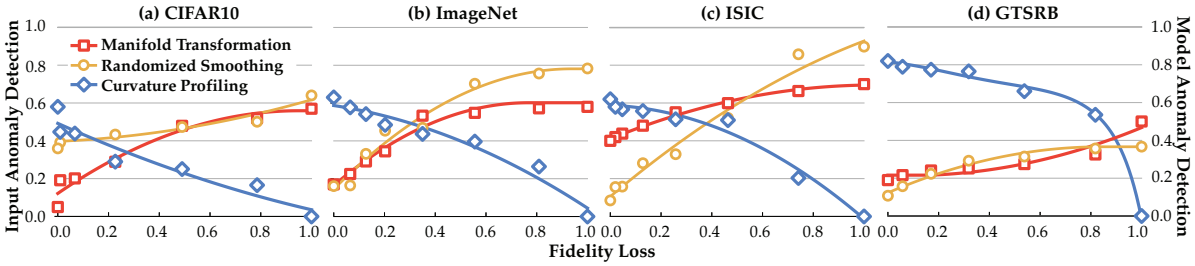

Figure 3.4: Detection rates of input anomaly (by manifold projection [1]) and model anomaly (by curvature profile [2]).

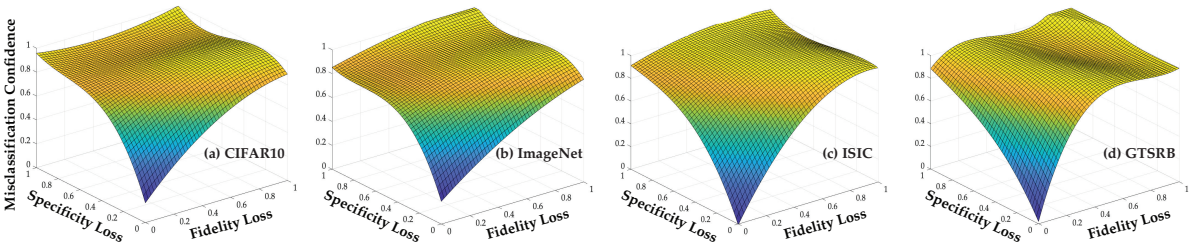

Figure 3.5: Average misclassification confidence (κ) as a function of fidelity and specificity losses.

3.3.2.2 Empirical Implications

The leverage effect has profound implications. We show that it entails a large design spectrum for the adversary to optimize the attack evasiveness with respect to various detection methods (detectors). Note that here we do not consider the adversary's adaptiveness to specific detectors but rather focus on exposing the design spectrum enabled from a detection perspective. In § 3.4, we show that IMC also allows to enhance the attacks by adapting to specific detectors. To assess IMC's evasiveness, we consider three complementary detectors.

Input Anomaly – From the input anomaly perspective, we apply manifold transformation [1] as the detector. At a high level, it employs a reformer network to project given inputs to the manifold spanned by benign inputs and a detector network to differentiate benign and adversarial inputs. Besides, we apply randomized smoothing [34] as another detector, which transforms a given DNN into a "smoothed" model and considers a given input x_* as adversarial if the probability difference of x_*'s largest and second largest classes exceeds a threshold.

Model Anomaly – From the model anomaly perspective, we apply curvature profiling [2] as the detector. Recall that the poisoning attack twists the classification boundary surrounding the target input x_*; thus, the loss function tends to change abruptly in x_*'s vicinity. To quantify this property, we compute the eigenvalues of the Hessian $H_x(x_*) = \nabla_x^2 \ell(x_*)$. Intuitively, larger (absolute) eigenvalues indicate larger curvatures of the loss function. We define the average (absolute) value of the top-k eigenvalues of $H_x(x_*)$ as x_*'s curvature profile: $\frac{1}{k}\sum_{i=1}^{k} |\lambda_i(H_x(x_*))|$, where $\lambda_i(M)$ is the i-th eigenvalue of matrix M (details in Appendix B). We compare the curvature profiles of given inputs and benign ones, and use the Kolmogorov–Smirnov statistics to differentiate the two sets.

We apply the above detectors to the adversarial inputs and poisoned models generated by IMC under varying fidelity-specificity trade-off (κ fixed as 0.75). Figure 3.4 measures the detection rates for different datasets. We have the following observations.

The detection rate of input anomaly grows monotonically with the fidelity loss (i.e., input perturbation magnitude); on the contrary, the detection rate of model anomaly drops quickly with the fidelity loss (i.e., disproportionate specificity improvement due to the leverage effect). For instance, in the case of ImageNet (Figure 3.4 (b)), as the fidelity loss varies from 0 to 0.35, the detection rate of input anomaly increases from 0.17 to 0.53 by manifold transformation and from 0.16 to 0.47 by randomized smoothing, while the detection rate of corresponding model anomaly drops from 0.63 to 0.44.

Moreover, across all the cases, IMC is able to balance fidelity and specificity, leading to high evasiveness with respect to multiple detectors simultaneously. For instance, in the case of CIFAR10 (Figure 3.4 (a)), with the fidelity loss set as 0.23, the detection rates of manifold transformation, randomized smoothing, and curvature profiling are reduced to 0.29, 0.43, and 0.29 respectively.

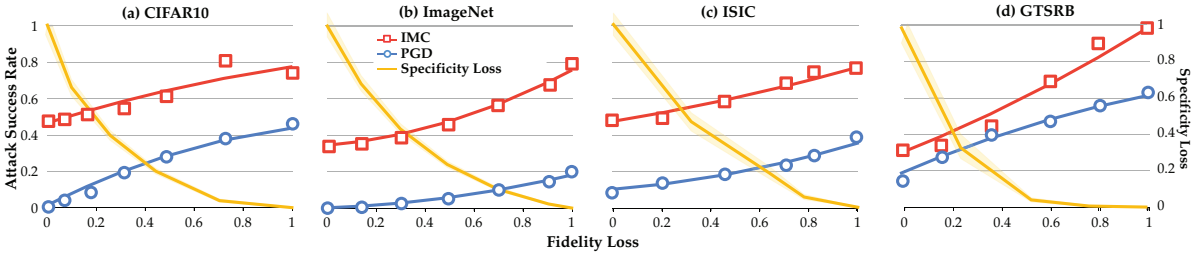

Figure 3.6: Accuracy and robustness (with respect to PGD and IMC) of adversarially re-trained models.

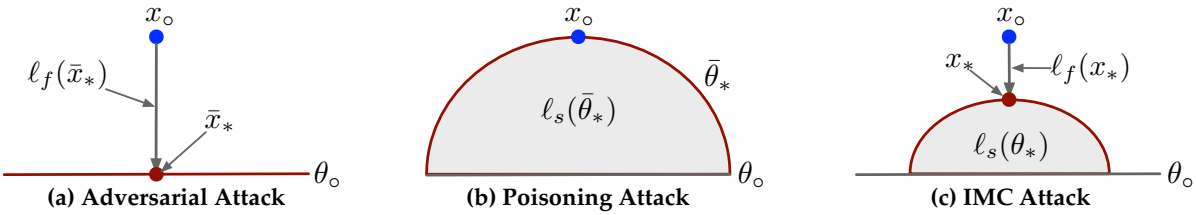

Figure 3.7: Comparison of the adversarial, poisoning, and IMC attacks under fixed attack efficacy.

3.3.3 Effect II: Amplification Effect

Next we show that the two attack vectors are able to amplify each other and attain attack efficacy unreachable by each vector alone.

3.3.3.1 Mutual Amplification

We measure the attack efficacy (average misclassification confidence) attainable by the adversarial, poisoning, and IMC attacks under varying fidelity and specificity losses. The results are shown in Figure 3.5 We have two observations.

First, IMC realizes higher attack efficacy than simply combining the adversarial and poisoning attacks. For instance, in the case of ISIC (Figure 3.5 (c)), with fidelity loss fixed as 0.2, the adversarial attack achieves κ about 0.25; with specificity loss fixed as 0.2, the poisoning attack attains κ around 0.4; while IMC reaches κ above 0.8 under this setting. This is explained by that IMC employs a stronger threat model to jointly optimize the perturbations introduced at both training and inference.

Second, IMC is able to attain attack efficacy unreachable by using each attack vector alone. Across all the cases, IMC achieves $\kappa = 1$ under proper fidelity and specificity settings, while the adversarial (or poisoning) attack alone (even with fidelity or specificity loss fixed as 1) is only able to reach κ less than 0.9.

Amplification Effect

Adversarial inputs and poisoned models amplify each other and give rise to attack efficacy unreachable by using each vector alone.

3.3.3.2 Empirical Implications

This amplification effect entails profound implications for the adversary to design more effective attacks. Here we explore to use adversarial training [37, 88], one state-of-the-art defense against adversarial attacks [63], to cleanse poisoned models. Starting with the poisoned model, the re-training iteratively updates it with adversarial inputs that deceive its current configuration (i.e., adversarial "re-training").

Dataset	Maximum Perturbation	
	PGD	IMC
CIFAR10	3×10^{-2}	2×10^{-3}
ImageNet	4×10^{-3}	1×10^{-3}
ISIC	3×10^{-2}	1×10^{-3}
GTSRB	3×10^{-2}	3×10^{-2}

Table 3.3. Maximum input perturbation magnitude for PGD and IMC.

We perform adversarial re-training on each poisoned model θ_* generated by IMC under varying fidelity-specificity trade-off (implementation details in Appendix B). We evaluate the re-trained model $\tilde{\theta}_*$ in terms of (i) the attack success rate of PGD (i.e., $\tilde{\theta}_*$'s robustness against regular adversarial attacks), (ii) the attack success rate of θ_*'s corresponding adversarial input x_* (i.e., $\tilde{\theta}_*$'s robustness against IMC), and (iii) $\tilde{\theta}_*$'s overall accuracy over benign inputs in the testset. Note that in order to work against the re-trained models, PGD is enabled with significantly higher perturbation magnitude than IMC. Table 3.3 summarizes PGD and IMC's maximum allowed perturbation magnitude (i.e., fidelity loss) for each dataset.

Observe that adversarial re-training greatly improves the robustness against PGD, which is consistent with prior work [37, 88]. Yet, due to the amplification effect, IMC retains its high attack effectiveness against the re-trained model. For instance, in the

case of ISIC (Figure 3.6 (c)), even with the maximum perturbation, PGD attains less than 40% success rate; in comparison, with two orders of magnitude lower perturbation, IMC succeeds with close to 80% chance. This also implies that adversarial re-training is in general ineffective against IMC. Also observe that by slightly increasing the input perturbation magnitude, IMC sharply improves the specificity of the poisoned model (e.g., average accuracy over benign inputs), which is attributed to the leverage effect. Note that while here IMC is not adapted to adversarial re-training, it is possible to further optimize the poisoned model by taking account of this defense during training, similar to [25].

3.3.4 Analytical Justification

We now provide analytical justification for the empirical observations regarding the mutual reinforcement effects.

3.3.4.1 Loss Measures

Without loss of generality, we consider a binary classification setting (i.e., $\mathcal{Y} = \{0, 1\}$), with $(1 - t)$ and t being the benign input $x_\circ$'s ground-truth class and the adversary's target class respectively. Let $f_t(x; \theta)$ be the model θ's predicted probability that x belongs to t. Under this setting, we quantify the set of attack objectives as follows.

Efficacy – The attack succeeds only if the adversarial input x_* and poisoned model θ_* force $f_t(x_*; \theta_*)$ to exceed 0.5 (i.e., the input crosses the classification boundary). We thus use $\kappa \triangleq f_t(x_\circ; \theta_\circ) - 0.5$ to measure the current gap between $\theta_\circ$'s prediction regarding $x_\circ$ and the adversary's target class t.

Fidelity – We quantify the fidelity loss using the L_p-norm of the input perturbation: $\ell_{\mathrm{f}}(x_*) = \|x_* - x_\circ\|_p$. For two adversarial inputs x_*, x'_*, we say $x_* < x'_*$ if $\ell_{\mathrm{f}}(x_*) < \ell_{\mathrm{f}}(x'_*)$. For simplicity, we use $p = 2$, while the analysis generalizes to other norms as well.

As shown in Figure 3.7 (a), in a successful adversarial attack (with the adversarial input $\bar{x}_*$), if the perturbation magnitude is small enough, we can approximate the fidelity loss as $x_\circ$'s distance to the classification boundary [89]: $\ell_{\mathrm{f}}(\bar{x}_*) \approx \kappa / \|\nabla_x \ell(x_\circ; \theta_\circ)\|_2$, where a linear approximation is applied to the loss function. In the following, we denote $h \triangleq \ell_{\mathrm{f}}(\bar{x}_*)$.

Specificity – Recall that the poisoned model θ_* modifies $x_\circ$'s surrounding classification boundary, as shown in Figure 3.7 (b). While it is difficult to exactly describe the classification boundaries encoded by DNNs [90], we approximate the local boundary

surrounding an input with the surface of a d-dimensional sphere, where d is the input dimensionality. This approximation is justified as follows.

First, it uses a quadratic form, which is more expressive than a linear approximation [89]. Second, it reflects the impact of model complexity on the boundary: the maximum possible curvature of the boundary is often determined by the model's inherent complexity [90]. For instance, the curvature of a linear model is 0, while a one hidden-layer neural network with an infinite number of neurons is able to model arbitrary boundaries [91]. We relate the model's complexity to the maximum possible curvature, which corresponds to the minimum possible radius of the sphere.

The boundaries before and after the attacks are thus described by two hyper-spherical caps. As the boundary before the attack is fixed, without loss of generality, we assume it to be flat for simplicity. Now according to Eq. 3.7, the specificity loss is measured by the number of inputs whose classifications are changed due to θ. Following the assumptions, such inputs reside in a d-dimensional hyper-spherical cap, as shown in Figure 3.7 (b). Due to its minuscule scale, the probability density p_{data} in this cap is roughly constant. Minimizing the specificity loss is thus equivalent to minimizing the cap volume [92], which amounts to maximizing the curvature of the sphere (or minimizing its radius). Let r be the minimum radius induced by the model. We quantify the specificity loss as:

$$\ell_{\text{s}}(\theta) = p_{\text{data}} \frac{\pi^{\frac{d-1}{2}} r^d}{\Gamma\left(\frac{d+1}{2}\right)} \int_0^{\arccos\left(1-\frac{h}{r}\right)} \sin^d(t)\,\mathrm{d}t \tag{3.8}$$

where $\Gamma(z) \triangleq \int_0^\infty t^{z-1}e^{-t}\,\mathrm{d}t$ is the Gamma function.

3.3.4.2 Mutual Reinforcement Effects

Let $\bar{x}_*, \bar{\theta}_*$ be the adversarial input and poisoned model given by the adversarial and poisoning attacks respectively, and (x_*, θ_*) be the adversarial input and poisoned model generated by IMC. Note that for fixed attack efficacy, $x_* = \bar{x}_*$ if $\theta_* = \theta_\circ$ and $\theta_* = \bar{\theta}_*$ if $x_* = x_\circ$.

Leverage Effect – We now quantify the leverage effect in the case of trading fidelity for specificity, while the alternative case can be derived similarly. Specifically, this effect is measured by the ratio of specificity "saving" versus fidelity "cost", which we term as the *leverage effect coefficient*:

$$\phi(x_*, \theta_*) \triangleq \frac{1 - \ell_{\text{s}}(\theta_*)/\ell_{\text{s}}(\bar{\theta}_*)}{\ell_{\text{f}}(x_*)/\ell_{\text{f}}(\bar{x}_*)} \tag{3.9}$$

Intuitively, the numerator is the specificity "saving", while the denominator is the fidelity "cost". We say that the trade-off is significantly disproportionate, if $\phi(x_*, \theta_*) \gg 1$, i.e., the saving dwarfs the cost. It is trivial to verify that if $\phi(x_*, \theta_*) \gg 1$ then the effect of trading specificity for fidelity is also significant $\phi(\theta_*, x_*) \gg 1$.[2]

Consider the IMC attack as shown in Figure 3.7 (c). The adversarial input x_* moves towards the classification boundary and reduces the loss by $\kappa'(\kappa' < \kappa)$. The perturbation magnitude is thus at least $\kappa'/\|\nabla_x \ell(x_\circ; \theta_\circ)\|_2$. The relative fidelity loss is given by:

$$\ell_{\mathrm{f}}(x_*)/\ell_{\mathrm{f}}(\bar{x}_*) = \kappa'/\kappa \tag{3.10}$$

Below we use $z = \kappa'/\kappa$ for a short notation.

Meanwhile, it is straightforward to derive that the height of the hyper-spherical cap is $(1 - z)h$. The relative specificity loss is thus:

$$\ell_{\mathrm{s}}(\theta_*)/\ell_{\mathrm{s}}(\bar{\theta}_*) = \frac{\int_0^{\arccos\left(1-\frac{h}{r}+z\frac{h}{r}\right)} \sin^d(t)\,\mathrm{d}t}{\int_0^{\arccos\left(1-\frac{h}{r}\right)} \sin^d(t)\,\mathrm{d}t} \tag{3.11}$$

Instantiating Eq. 3.9 with Eq. 3.10 and Eq. 3.11, the leverage effect of trading fidelity for specificity is defined as:

$$\phi(x_*, \theta_*) = \frac{\int_{\arccos\left(1-\frac{h}{r}+z\frac{h}{r}\right)}^{\arccos\left(1-\frac{h}{r}\right)} \sin^d(t)\,\mathrm{d}t}{z\int_0^{\arccos\left(1-\frac{h}{r}\right)} \sin^d(t)\,\mathrm{d}t} \tag{3.12}$$

The following proposition justifies the effect of trading fidelity for specificity (proof in Appendix A). A similar argument can be derived for trading specificity for fidelity.

Proposition 2. The leverage effect defined in Eq. 3.12 is strictly greater than 1 for any $0 < z < 1$.

Intuitively, to achieve fixed attack efficacy (κ), with a slight increase of fidelity loss $\ell_{\mathrm{f}}(x_*)$, the specificity loss $\ell_{\mathrm{s}}(\theta_*)$ is reduced super-linearly.

Figure 3.8 evaluates this effect as a function of relative fidelity loss under varying setting of h/r. Observe that the effect is larger than 1 by a large margin, especially for small fidelity loss κ'/κ, which is consistent with our empirical observation: with little fidelity cost, it is possible to significantly reduce the specificity loss.

[2] If $(1 - x)/y \gg 1$ then $(1 - y)/x \gg 1$ for $0 < x, y < 1$.

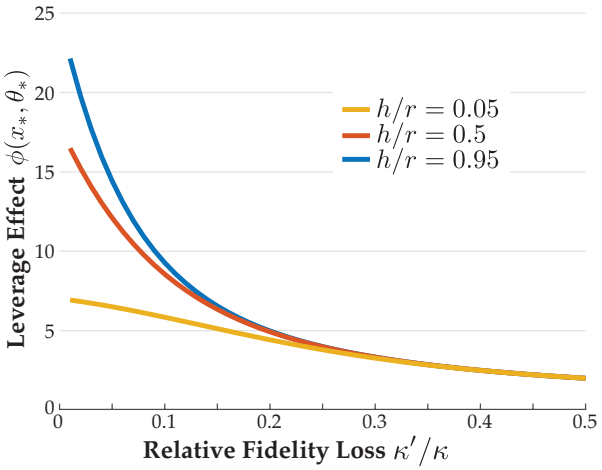

Figure 3.8: Leverage effect with respect to the relative fidelity loss z and the minimum radius r (with $d = 50$).

Amplification Effect – From Proposition 2, we can also derive the explanation for the amplification effect.

Consider an adversarial input x_* that currently achieves attack efficacy κ' with relative fidelity loss κ'/κ. Applying the poisoned model θ_* with relative specificity loss $(1 - \kappa'/\kappa)/\phi(x_*, \theta_*)$, the adversary is able to attain attack efficacy κ. In other words, the poisoned model θ_* "amplifies" the attack efficacy of the adversarial input x_* by κ/κ' times, with cost much lower than required by using the adversarial attack alone to reach the same attack efficacy (i.e., $1 - \kappa'/\kappa$), given that $\phi(x_*, \theta_*) \gg 1$ in Proposition 2.

3.4 IMC-Optimized Attacks

In this section, we demonstrate that IMC, as a general attack framework, can be exploited to enhance existing attacks with respect to multiple metrics. We further discuss potential countermeasures against such optimized attacks and their technical challenges.

3.4.1 Attack Optimization

3.4.1.1 Basic Attack

We consider `TrojanNN` [17], a representative backdoor attack, as the reference attack model. At a high level, `TrojanNN` defines a specific pattern (e.g., watermark) as the trigger and enforces the poisoned model to misclassify all the inputs embedded with this trigger. As it optimizes both the trigger and poisoned model, `TrojanNN` enhances other backdoor attacks (e.g., `BadNet` [16]) that employ fixed trigger patterns.

Specifically, the attack consists of three steps. (i) First, the trigger pattern is partially defined in an ad hoc manner; that is, the watermark shape (e.g., square) and embedding position are pre-specified. (ii) Then, the concrete pixel values of the trigger are optimized to activate neurons rarely activated by benign inputs, in order to minimize the impact on benign inputs. (iii) Finally, the model is re-trained to enhance the effectiveness of the trigger pattern.

Note that within `TrojanNN`, the operations of trigger optimization and model re-training are executed independently. It is thus possible that after re-training, the neurons activated by the trigger pattern may deviate from the originally selected neurons, resulting in suboptimal trigger patterns and/or poisoned models. Also note that `TrojanNN` works without access to the training data; yet, to make fair comparison, in the following evaluation, `TrojanNN` also uses the original training data to construct the backdoors.

3.4.1.2 Enhanced Attacks

We optimize `TrojanNN` within the IMC framework. Compared with optimizing the trigger only [93], IMC improves `TrojanNN` in terms of both attack effectiveness and evasiveness. Specifically, let r denote the trigger. We initialize r with the trigger pre-defined by `TrojanNN` and optimize it using the co-optimization procedure. To this end, we introduce a mask m for the given benign input $x_\circ$. For $x_\circ$'s i-th dimension (pixel), we define $m[i] = 1 - p$ (p is the transparency setting) if i is covered by the watermark and $m[i] = 0$ otherwise. Thus the perturbation operation is defined as $x_* = \psi(x_\circ, r; m) = x_\circ \odot (1 - m) + r \odot m$, where $\odot$ denotes element-wise multiplication. We reformulate the backdoor attack in Eq. 3.3 as follows:

$$\min_{\theta,r,m} \ \mathbb{E}_{x_\circ \in \mathcal{T}} \left[\ell(\psi(x_\circ, r; m), t; \theta)\right] + \lambda \ell_{\mathrm{f}}(m) + \nu \ell_{\mathrm{s}}(\theta) \tag{3.13}$$

where we define the fidelity loss in terms of m. Typically, $\ell_{\mathrm{f}}(m)$ is defined as m's L_1 norm and $\ell_{\mathrm{s}}(\theta)$ is the accuracy drop on benign cases similar to Eq. 3.7.

Algorithm 2 sketches the optimization procedure of Eq. 3.13. It alternates between optimizing the trigger and mask (line 4) and optimizing the poisoned model (line 5). Specifically, during the trigger perturbation step, we apply the Adam optimizer [83]. Further, instead of directly optimizing r which is bounded by $[0, 1]$, we apply change-of-variable and optimize over a new variable $w_r \in (-\infty, +\infty)$, such that $r = (\tanh(w_r)+1)/2$ (the same trick is also applied on m). Note that Algorithm 2 represents a general optimization framework, which is adaptable to various settings. For instance, one may

specify all the non-zero elements of m to share the same transparency or optimize the transparency of each element independently (details in § 3.4.2 and § 3.4.3). In the following, we term the enhanced `TrojanNN` as `TrojanNN`*.

Algorithm 2: `TrojanNN`* Attack

Input: initial trigger mask – $m_\circ$; benign model – $\theta_\circ$; target class – t; hyper-parameters – λ, ν

Output: trigger mask – m; trigger pattern – r, poisoned model – θ_*

`// initialization`

$\theta^{(0)}, k \leftarrow \theta_\circ, 0$;

$m^{(0)}, r^{(0)} \leftarrow m_0, \texttt{TrojanNN}(m_0)$;

`// optimization`

while *not converged yet* **do**

  `// trigger perturbation`

  $r^{(k+1)}, m^{(k+1)} \leftarrow \arg\min_{r,m} \mathbb{E}_{x_\circ \in \mathcal{T}} \left[\ell(\psi(x_\circ, r; m), t; \theta^{(k)})\right] + \lambda \ell_{\mathrm{f}}(m)$;

  `// model perturbation`

  $\theta^{(k+1)} \leftarrow \arg\min_{\theta} \mathbb{E}_{x_\circ \in \mathcal{T}} \left[\ell(\psi(x_\circ, r^{(k)}; m^{(k)}), t; \theta)\right] + \nu \ell_{\mathrm{s}}(\theta)$;

  $k \leftarrow k + 1$;

return $(m^{(k)}, r^{(k)}, \theta^{(k)})$;

3.4.2 Optimization against Human Vision

We first show that `TrojanNN`* is optimizable in terms of its evasiveness with respect to human vision. The evasiveness is quantified by the size and transparency (or opacity) of trigger patterns. Without loss of generality, we use square-shaped triggers. The trigger size is measured by the ratio of its width over the image width.

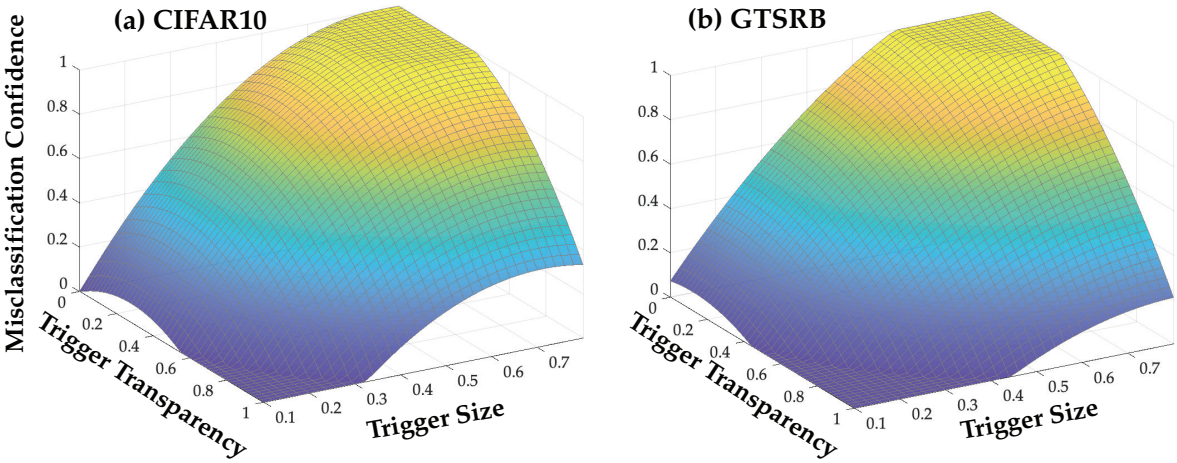

Figure 3.9: Attack efficacy of `TrojanNN`* as a function of trigger size and transparency.

Figure 3.9 illustrates `TrojanNN`*'s attack efficacy (average misclassification confidence of trigger-embedded inputs) under varying evasiveness constraints. Observe that the

efficacy increases sharply as a function of the trigger size or opacity. Interestingly, the trigger size and opacity also demonstrate strong mutual reinforcement effects: (i) leverage - for fixed attack efficacy, by a slight increase in opacity (or size), it significantly reduces the size (or opacity); (ii) amplification - for fixed opacity (or size), by slightly increasing size (or opacity), it greatly boosts the attack efficacy.

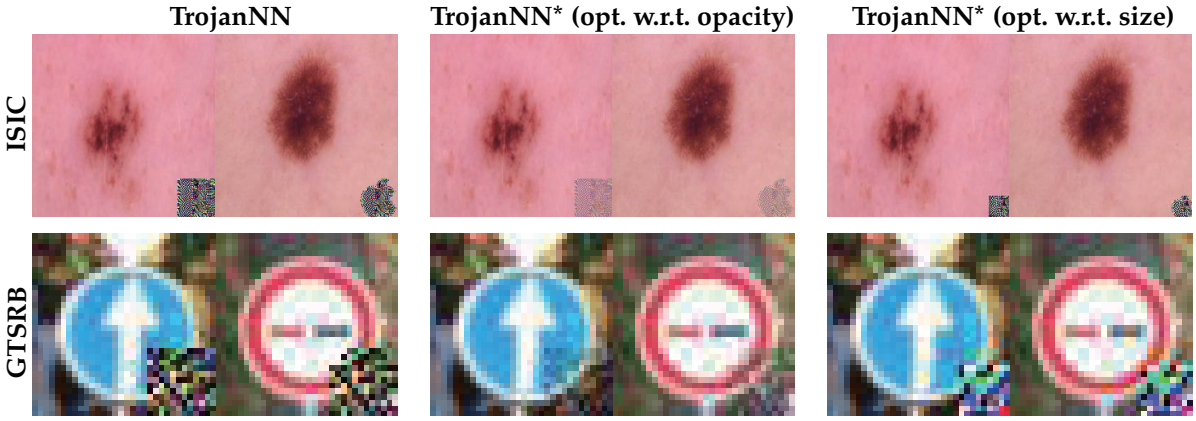

Figure 3.10: Sample triggers generated by `TrojanNN` (a), `TrojanNN`* optimizing opacity (b) and optimizing size (c).

To further validate the leverage effect, we compare the triggers generated by `TrojanNN` and `TrojanNN`*. Figure 3.10 shows sample triggers given by `TrojanNN` and `TrojanNN`* under fixed attack efficacy (with $\kappa = 0.95$). It is observed that compared with `TrojanNN`, `TrojanNN`* significantly increases the trigger transparency (under fixed size) or minimizes the trigger size (under fixed opacity).

To further validate the amplification effect, we measure the attack success rate (ASR) of `TrojanNN` and `TrojanNN`* under varying evasiveness constraints (with $\kappa = 0.95$), with results shown in Figure 3.11 and 3.12. It is noticed that across all the datasets, `TrojanNN`* outperforms `TrojanNN` by a large margin under given trigger size and opacity. For instance, in the case of GTSRB (Figure 3.11 (d)), with trigger size and transparency fixed as 0.4 and 0.7, `TrojanNN`* outperforms `TrojanNN` by 0.39 in terms of ASR; in the case of CIFAR10 (Figure 3.12 (a)), with trigger size and transparency fixed as 0.3 and 0.2, the ASRs of `TrojanNN`* and `TrojanNN` differ by 0.36.

We can thus conclude that leveraging the co-optimization framework, `TrojanNN`* is optimizable with respect to human detection without affecting its attack effectiveness.

3.4.3 Optimization against Detection Methods

In this set of experiments, we demonstrate that `TrojanNN`* is also optimizable in terms of its evasiveness with respect to multiple automated detection methods.

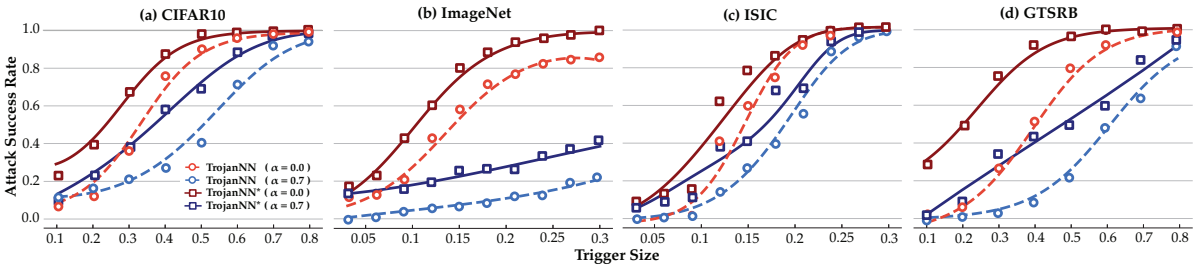

Figure 3.11: ASR of `TrojanNN` and `TrojanNN`* as functions of trigger size.

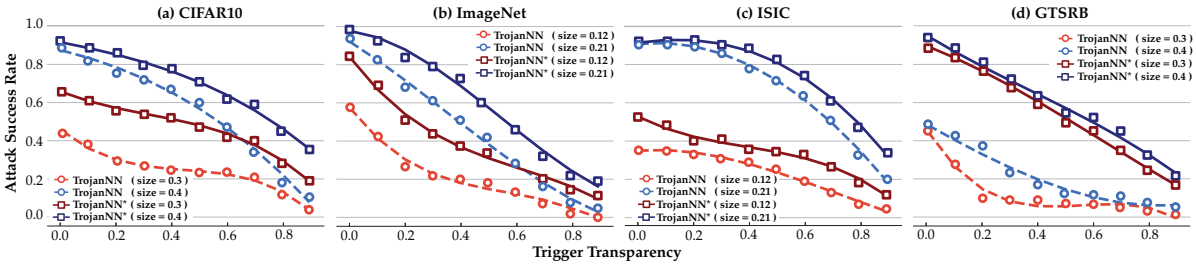

Figure 3.12: ASR of `TrojanNN` and `TrojanNN`* as functions of trigger transparency.

3.4.3.1 Backdoor Detection

The existing backdoor detection methods can be roughly classified in two categories based on their application stages and detection targets. The first class is applied at the model inspection stage and aims to detect suspicious models and potential backdoors [26, 27, 67]; the other class is applied at inference time and aims to detect trigger-embedded inputs [68–71]. In our evaluation, we use NeuralCleanse [26] and STRIP [70] as the representative methods of the two categories. In Appendix C, we also evaluate `TrojanNN` and `TrojanNN`* against ABS [27], another state-of-the-art backdoor detector.

NeuralCleanse – For a given DNN, NeuralCleanse searches for potential triggers in every class. Intuitively, if a class is embedded with a backdoor, the minimum perturbation (measured by its L_1-norm) necessary to change all the inputs in this class to the target class is abnormally smaller than other classes. Empirically, after running the trigger search algorithm over 1,600 randomly sampled inputs for 10 epochs, a class with its minimum perturbation normalized by median absolute deviation exceeding 2.0 is considered to contain a potential backdoor with 95% confidence.

STRIP – For a given input, STRIP mixes it up with a benign input using equal weights, feeds the mixture to the target model, and computes the entropy of the prediction vector (i.e., self-entropy). Intuitively, if the input is embedded with a trigger, the mixture is still dominated by the trigger and tends to be misclassified to the target class, resulting in relatively low self-entropy; otherwise, the self-entropy tends to be higher. To reduce

variance, for a given input, we average its self-entropy with respect to 8 randomly sampled benign inputs. We set the positive threshold as 0.05 and measure STRIP's effectiveness using F-1 score.

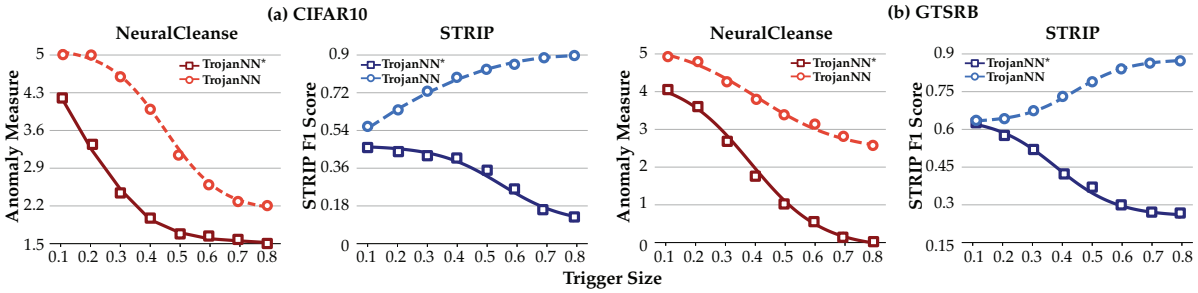

Figure 3.13: Detection of `TrojanNN` and `TrojanNN`* by NeuralCleanse and STRIP on CIFAR10 and GTSRB.

3.4.3.2 Attack Optimization

We optimize `TrojanNN`* in terms of its evasiveness with respect to both NeuralCleanse and STRIP. Both detectors aim to detect anomaly under certain metrics, which we integrate into the loss terms in Algorithm 2.

Specifically, NeuralCleanse searches for potential trigger with minimum L_1-norm, which is related to the mask m. We thus instantiate the fidelity loss $\ell_{\mathrm{f}}(m)$ as m's L_1-norm and optimize it during the trigger perturbation step. To normalize $\ell_{\mathrm{f}}(m)$ to an appropriate scale, we set the hyper-parameter λ as the number of pixels covered by the trigger. Meanwhile, STRIP mixes adversarial and benign inputs and computes the self-entropy of the mixtures, which highly depends on the model's behaviors. We thus instantiate the specificity loss $\ell_{\mathrm{s}}(\theta)$ as $\mathbb{E}_{x,x' \in \mathcal{R}}\left[-H(f(\frac{x_*}{2} + \frac{x'}{2}; \theta))\right]$, in which we randomly mix up an adversarial input x_* (via perturbing a benign input x) and another benign input x' and maximize the self-entropy of their mixture.

3.4.3.3 Detection Evasiveness

We apply the above two detectors to detect `TrojanNN` and `TrojanNN`*, with results summarized in Figure 3.13. We have the following observations. First, the two detectors are fairly effective against `TrojanNN`. In comparison, `TrojanNN`* demonstrates much higher evasiveness. For instance, in the case of GTSRB (Figure 3.13 (b)), with trigger size fixed as 0.4, the anomaly measures of `TrojanNN`* and `TrojanNN` by NeuralCleanse differ by over 2, while the F-1 scores on `TrojanNN`* and `TrojanNN` by STRIP differ by more than 0.3. We thus conclude that `TrojanNN`* is optimizable in terms of evasiveness

with respect to multiple detection methods simultaneously.

3.4.4 Potential Countermeasures

Now we discuss potential mitigation against IMC-optimized attacks and their technical challenges. It is shown above that using detectors against adversarial inputs or poisoned models independently is often insufficient to defend against IMC-optimized attacks, due to the mutual reinforcement effects. One possible solution is to build ensemble detectors that integrate individual ones and detect IMC-optimized attacks based on both input and model anomaly.

To assess the feasibility of this idea, we build an ensemble detector against `TrojanNN`* via integrating NeuralCleanse and STRIP. Specifically, we perform the following detection procedure: (i) applying NeuralCleanse to identify the potential trigger, (ii) for a given input, attaching the potential trigger to a benign input, (iii) mixing this benign input up with the given input under varying mixture weights, (iv) measuring the self-entropy of these mixtures, and (v) using the standard deviation of the self-entropy values to distinguish benign and trigger-embedded inputs.

Intuitively, if the given input is trigger-embedded, the mixture combines two trigger-embedded inputs and is thus dominated by one of the two triggers, regardless of the mixture weight, resulting in a low deviation of self-entropy. In comparison, if the given input is benign, the mixture is dominated by the trigger only if the weight is one-sided, resulting in a high deviation of self-entropy.

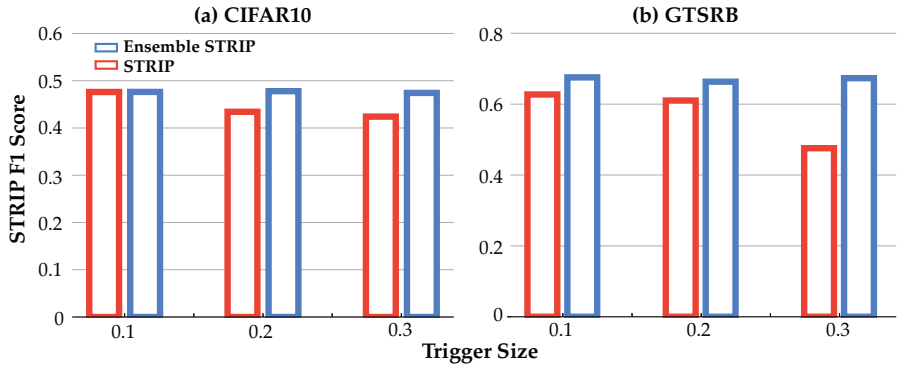

Figure 3.14: Detection of basic and ensemble STRIP against `TrojanNN`* on CIFAR10 and GTSRB.

We compare the performance of the basic and ensemble STRIP against `TrojanNN`*. As shown in Figure 3.14, the ensemble detector performs slightly better across all the cases, implying the effectiveness of the ensemble approach. However, the improvement is marginal (less than 0.2), especially in the case of small-sized triggers. This may be

explained by the inherent challenges of defending against IMC-optimized attacks: due to the mutual reinforcement effects, TrojanNN* attains high attack efficacy with minimal input and model distortion; it thus requires to carefully account for such effects in order to design effective countermeasures.

3.5 Conclusion

This work represents a solid step towards understanding adversarial inputs and poisoned models in a unified manner. We show both empirically and analytically that (i) there exist intriguing mutual reinforcement effects between the two attack vectors, (ii) the adversary is able to exploit such effects to optimize attacks with respect to multiple metrics, and (iii) it requires to carefully account for such effects in designing effective countermeasures against the optimized attacks. We believe our findings shed light on the holistic vulnerabilities of DNNs deployed in realistic settings.

This work also opens a few avenues for further investigation. First, besides the targeted, white-box attacks considered in this dissertation, it is interesting to study the connections between the two vectors under alternative settings (e.g., untargeted, black-box attacks). Second, enhancing other types of threats (e.g., latent backdoor attacks) within the input-model co-optimization framework is a direction worthy of exploration. Finally, devising a unified robustness metric accounting for both vectors may serve as a promising starting point for developing effective countermeasures.

Chapter 4
TrojanZoo: Unified Evaluation of Neural Backdoors

4.1 Introduction

With the rapid development of new attacks/defenses, a number of open questions have emerged:

RQ_1 – *What are the strengths and limitations of different attacks/defenses?*

RQ_2 – *What are the best practices (e.g., optimization strategies) to operate them?*

RQ_3 – *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 under-explored due to the following challenges.

Non-holistic evaluations – Most studies conduct evaluation with a fairly limited set of attacks/defenses, resulting in incomplete assessment. For instance, it is unknown whether STRIP [32] is effective against the newer ABE attack [94]. Further, the evaluation often uses 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 attack performance, which is insufficient to describe the attack's ability of trading between these two metrics.

Non-unified platforms – Due to the lack of unified benchmarks, different attacks/defenses are often evaluated under inconsistent settings, leading to non-comparable conclusions. For instance, TNN [18] and LB [25] are evaluated with distinct trigger definitions (*i.e.*, shape, size, and transparency), datasets, and DNNs, making it difficult to directly compare their assessment.

Non-adaptive attacks – The evaluation of the existing defense [26, 29, 31, 32] 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 unified, holistic, and practical manner. Our contributions are summarized in the following three aspects.

Platform – To our best knowledge, TrojanZoo represents the first open-source platform designed for evaluating neural backdoor attacks/defenses. To date, TrojanZoo has incorporated 8 representative attacks, 14 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. Further, TrojanZoo implements a rich set of tools for in-depth analysis of the attack-defense interactions, including measuring feature-space similarity, tracing neural activation patterns, and comparing attribution maps.

Assessment – Leveraging TrojanZoo, we conduct a systematic study on the existing attacks/defenses, unveiling the complex design spectrum for the adversary and the defender. Different attacks manifest trade-offs among effectiveness, evasiveness, and transferability. For instance, weaker attacks (*i.e.*, lower *ASR*) tend to show higher transferability. Meanwhile, different defenses demonstrate trade-offs among robustness, utility-preservation, and detection accuracy. For instance, while generally effective against a variety of attacks, model sanitization [31, 37] also incur a significant accuracy drop. Such observations also imply the importance of using comprehensive metrics to evaluate neural backdoor attacks/defenses.

Exploration – We further explore improving existing attacks/defenses, leading to a number of previously unknown findings including (*i*) one-pixel triggers often suffice (over 95% *ASR*); (*ii*) training from scratch often outperforms perturbing benign models to forge trojan models; (*iii*) leveraging DNN architectures (*e.g.*, skip connects) in optimizing trojan models improves the attack effectiveness; (*iv*) most individual defenses are vulnerable to adaptive attacks; and (v) exploiting model interpretability significantly improves defense robustness. We envision that the TrojanZoo platform and 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.

4.2 Platform

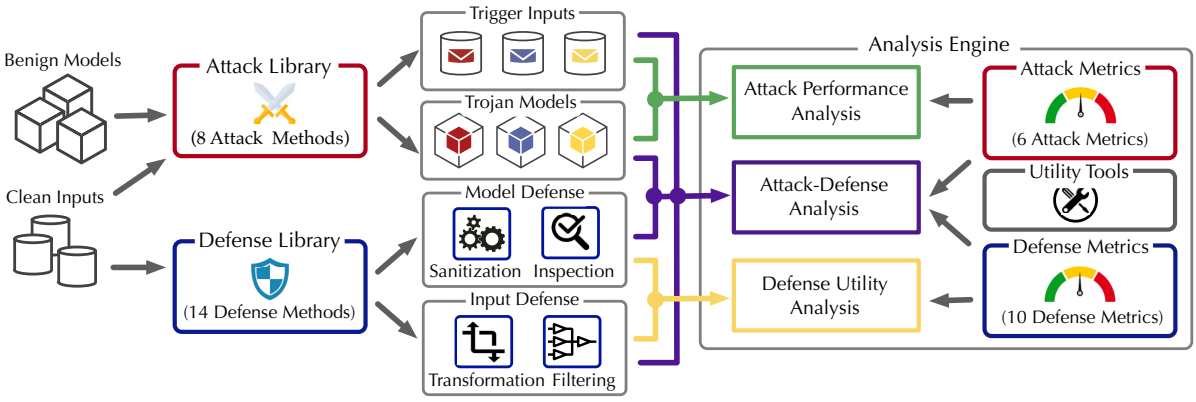

Figure 4.1: Overall system design of TrojanZoo.

As illustrated in Figure 4.1, TrojanZoo comprises three major components: (*i*) the attack library integrates a set of representative attacks that, for given benign models and clean inputs, are able to generate trojan models and trigger inputs; (*ii*) the defense library integrates a set of state-of-the-art defenses that are able to provide model- and input-level protection against trojan models and trigger inputs; and (*iii*) the analysis engine, equipped with attack performance metrics, defense utility metrics, and feature-rich utility tools, is able to conduct unified and holistic evaluation across different attacks/defenses.

In its current implementation, TrojanZoo has incorporated 9 attacks, 14 defenses, 6 attack performance metrics, and 10 defense utility metrics, which we systemize as follows.

4.2.1 Attack Library

While neural backdoor attacks can be characterized from a number of aspects, here we focus on 4 key design choices by the adversary that directly impact attack performance. Table 4.1 summarizes the representative neural backdoor attacks currently implemented in TrojanZoo, which are characterized along the above 4 dimensions. More specifically,

Non-optimization – The attack simply solves Eq. 2.4 under pre-defined triggers (*i.e.*, shape, transparency, and pattern) without optimization for other desiderata.

– BadNet (Bn) [16], as the representative, pre-defines trigger r, generates trigger inputs $\{(x \oplus r, t)\}$, and crafts the trojan model f^* by re-training a benign model f with such data.

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

Attack	Architecture Modifiability	Trigger Optimizability	Fine-tuning Survivability	Defense Adaptivity
Bn [16]	○	○	○	○
Esb [95]	●	○	○	○
Tnn [18]	○	◐	○	○
Rb [96]	○	◐	○	○
Tb [23]	○	◐	○	○
Lb [25]	○	○	●	○
Abe [94]	○	○	○	●
Imc [19]	○	●	●	●

Table 4.1. Summary of representative neural backdoor attacks currently implemented in TrojanZoo (● – full optimization, ◐ – partial optimization, ○ – no optimization)

defenses (*e.g.*, model specification checking).

– Embarrassingly-Simple-Backdoor (Esb) [95], as the representative, 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 – whether the attack uses a fixed, pre-defined 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).

– TrojanNN (Tnn) [18] 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.

– Reflection-Backdoor (Rb) [96] 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.

– Targeted-Backdoor (Tb) [23] 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).

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.

– Latent Backdoor (Lb) [25] accounts for the impact of downstream fine-tuning

by optimizing g with respect to latent representations rather than final predictions. Specifically, it instantiates Eq. 2.4 with the following loss function: $\ell(g(x \oplus r), t) = \Delta(g(x \oplus r), \phi_t)$, where Δ measures the difference of two latent representations and ϕ_t denotes the average representation of class t, defined as $\phi_t = \arg\min_\phi \mathbb{E}_{(x,t)\in\mathcal{T}}[g(x)]$.

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.

– Adversarial-Backdoor-Embedding (ABE) [94] accounts for possible defenses in forging g^*. In solving Eq. 2.4, 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 $\Delta(d \circ g(x), b(x))$, where $b(x)$ encodes whether x is trigger or clean, while g^* and d are trained using an adversarial learning framework [97].

Multi-optimization – whether the attack is optimizable with respect to multiple objectives listed above.

– Input-Model Co-optimization (IMC) [19] is motivated by the mutual-reinforcement effect between r and f^*: optimizing one amplifies the effectiveness of the other. Instead of solving Eq. 2.4 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).

4.2.2 Attack Performance Metrics

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

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

$$\text{Attack Success Rate } (ASR) = \frac{\#\text{ successful trials}}{\#\text{ total trials}} \tag{4.1}$$

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.

Neural Backdoor Defense	Category	Design Rationale
Randomized-Smoothing (Rs) [34]	Input Reformation	$\mathcal{A}$'s fidelity (x's and x^*'s surrounding class boundaries)
Down-Upsampling (Du) [98]		$\mathcal{A}$'s fidelity (x's and x^*'s high-level features)
Manifold-Projection (Mp) [99]		$\mathcal{A}$'s fidelity (x's and x^*'s manifold projections)
Activation-Clustering (Ac) [33]	Input Filtering	distinct activation patterns of $\{x\}$ and $\{x^*\}$
Spectral-Signature (Ss) [100]		distinct activation patterns of $\{x\}$ and $\{x^*\}$ (spectral space)
STRIP (Strip) [32]		distinct self-entropy of x and x^* mixtures with clean inputs
NEO (Neo) [35]		sensitivity of f^*'s prediction to trigger perturbation
Adversarial-Retraining (Ar) [37]	Model Sanitization	$\mathcal{A}$'s fidelity (x's and x^*'s surrounding class boundaries)
Fine-Pruning (Fp) [31]		$\mathcal{A}$'s use of neurons rarely activated by clean inputs
NeuralCleanse (Nc) [26]	Model Inpsection	abnormally small perturbation from other classes to t in f
DeepInspect (Di) [28]		abnormally small perturbation from other classes to t in f^*
TABOR (Tabor) [29]		abnormally small perturbation from other classes to t in f
NeuronInspect (Ni) [30]		distinct explanations of f and f^* w.r.t. clean inputs
ABS (Abs) [27]		$\mathcal{A}$'s use of neurons elevating t's prediction

Table 4.2. 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)

Clean accuracy drop (*CAD*) – which measures the difference of the classification accuracy of benign and trojan models; *CAD* measures 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 [19], 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 $\mathcal{N}_\mathrm{c}$ and $\mathcal{N}_\mathrm{t}$, the top-k active neurons with respect to clean and trigger inputs, respectively,

and calculate their jaccard index:

$$\text{Neuron Separation Ratio } (NSR) = 1 - \frac{|\mathcal{N}_\text{t} \cap \mathcal{N}_\text{c}|}{|\mathcal{N}_\text{t} \cup \mathcal{N}_\text{c}|} \tag{4.2}$$

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

4.2.3 Defense Library

The existing defenses against neural backdoors, according to their strategies, can be classified into 4 major categories, as summarized in Table 4.2. Notably, we focus on the setting of transfer learning or outsourced training, which precludes certain other defenses such as purging poisoning training data [101]. Next, we detail the 14 representative defenses currently implemented in TrojanZoo.

Input reformation – which, before feeding an incoming input to the model, 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 attack $\mathcal{A}$, that is, $\mathcal{A}$ tends to retain the perceptual similarity of a clean input x and its trigger counterpart x^*.

– Randomized-Smoothing (Rs) [34] 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).

– Down-Upsampling (Du) [98] 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.

– Manifold-Projection (Mp) [99] exploits the premise that $\mathcal{A}$ retains the similarity of x and x^* in terms of their projections to the data manifold. Thus, it trains an autoencoder to learn an approximate manifold, which projects x^* to the manifold.

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.

– Activation-Clustering (Ac) [33] 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.

– Spectral-Signature (Ss) [100] exploits the similar property in the spectral space.

– Strip [32] 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 [35] 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.

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.

– Adversarial-Retraining (AR) [37] treats trigger inputs as one type of adversarial inputs and applies adversarial training over the pre-trained model to improves its robustness to backdoor attacks.

– Fine-Pruning (FP) [31] 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 – which determines whether f is a trojan model and, if so, recovers the target class and the potential trigger, at the model checking stage.

– NeuralCleanse (NC) [26] 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.

– DeepInspect (DI) [28] follows a similar pipeline but uses a generative network to generate trigger candidates.

– TABOR [29] extends NC by adding a new regularizer to control the trigger search space.

– NeuronInspect (NI) [30] 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 [27] 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.

4.2.4 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 clean inputs. 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.

$$\text{True Positive Rate } (TPR) = \frac{\#\text{ detected trigger inputs}}{\#\text{ total trigger inputs}} \tag{4.3}$$

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.*, [26–29]) 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 :

$$\text{Mask Jaccard Similarity } (MJS) = \frac{|O(m^o) \cap O(m^r)|}{|O(m^o) \cup O(m^r)|} \tag{4.4}$$

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

Average running time (*ART*) – which measures $\mathcal{D}$'s overhead. 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.

4.3 Assessment

Leveraging TrojanZoo, we conduct a systematic assessment of the existing attacks and defenses and unveil their complex design spectrum: both attacks and defenses tend to manifest intricate trade-offs among multiple desiderata. We begin by describing the setting of the evaluation.

4.3.1 Experimental Setting

Dataset	# Class	# Dimension	Model	ACC
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	10	224×224		92.40%
VGGFace2	20	224×224		90.77%

Table 4.3. *ACC* of benign models over different datasets.

Datasets – In the evaluation, we primarily use 5 datasets: CIFAR10 [84], CIFAR100 [84], ImageNet [3], GTSRB [85], and VGGFace2 [102], with their statistics summarized in Table 4.3.

Models – We consider 3 representative DNN models: VGG [103], ResNet [87], and DenseNet [104]. Using models of distinct architectures (*e.g.*, residual blocks versus skip connections), we factor out the influence of individual model characteristics. By default, we assume the downstream classifier comprising one fully-connected layer with softmax activation (1Fcn). We also consider other types of classifiers, including Bayes, SVM, and Random Forest. The *ACC* of benign models is summarized in Table 4.3.

Attacks, Defenses, and Metrics – In the evaluation, we exemplify with 9 attacks in Table 4.1 and 12 defenses in Table 4.2, and measure them using all the metrics in § 4.2.2 and § 4.2.4. 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.

4.3.2 Attack Evaluation

We evaluate the existing attacks under the vanilla setting (without defenses), aiming to understand the impact of various design choices on the attack performance. Overall, different attacks manifest intricate trade-offs among *effectiveness*, *evasiveness*, and *transferability*, as detailed below.

4.3.2.1 Effectiveness vs. Evasiveness (Trigger)

We start with the effectiveness-evasiveness trade-off. Intuitively, the effectiveness measures whether the trigger inputs are successfully misclassified into the target class, while the evasiveness measures whether the trigger inputs and trojan models are distinguishable from their normal counterparts. Here, we first consider the evasiveness of triggers.

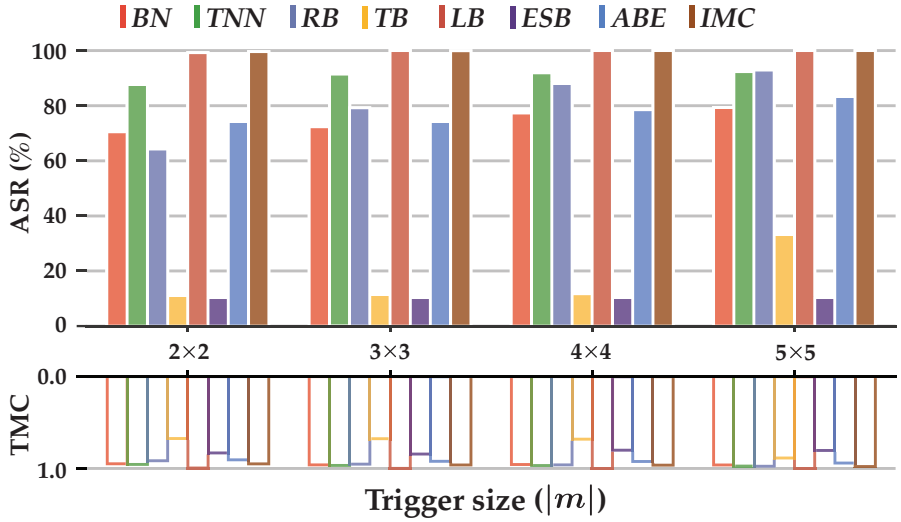

Figure 4.2: *ASR* and *TMC* with respect to trigger size ($\alpha = 0.8$).

Trigger size – Recall that the trigger definition comprises mask m, transparency α, and pattern p. We measure how the attack effectiveness 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 4.2 plots the attack success rate (*ASR*) and trojan misclassification confidence (*TMC*) of various attacks under varying $|m|$ (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 with over 30% growth. This may be attributed to its additional constraints: Rb defines the trigger to be the reflection of another image; thus, increasing $|m|$ may improve its perturbation spaces. Compared with other attacks, Tb and Esb perform poorly because Tb aims to force

inputs with random triggers to be misclassified while ESB is unable to account for trigger transparency during training. Also observe that the *TMC* of most attacks remains close to 1.0 regardless of $|m|$.

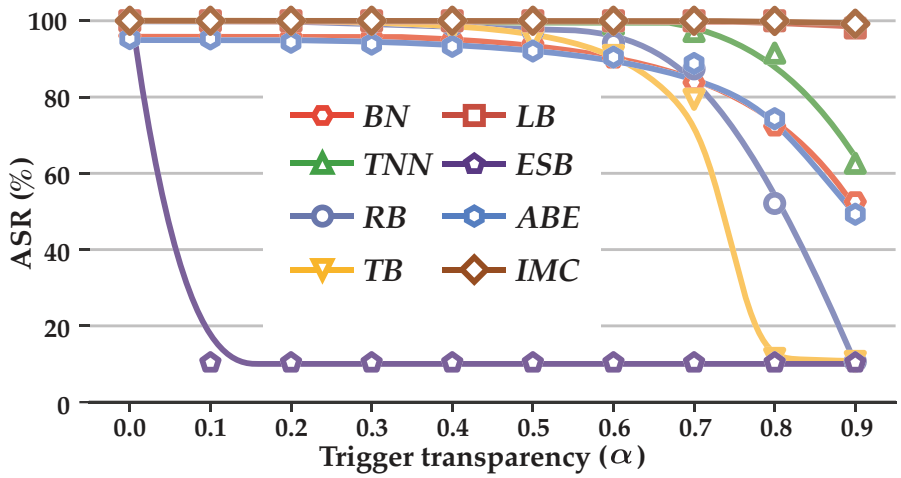

Figure 4.3: *ASR* with respect to trigger transparency ($|m| = 3{\times}3$).

Trigger transparency – Under the same setting, we evaluate the impact of trigger transparency α. Figure 4.3 plots the *ASR* of various attacks as a function of α ($|m| = 3{\times}3$).

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.

Data complexity – The trade-off between attack effectiveness and trigger evasiveness is especially evident for complex data. We compare the *ASR* and *TMC* of given attacks on different datasets, with results in Table 4.4.

We observe that the class-space size (the number of classes) negatively affects the attack effectiveness. 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).

Attack	CIFAR10	CIFAR100	ImageNet	
	$\|m\|=3,\ \alpha=0.8$	$\|m\|=3,\ \alpha=0.8$	$\|m\|=3,\ \alpha=0$	$\|m\|=7,\ \alpha=0.8$
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 4.4. Impact of data complexity on *ASR* and *TMC*.

Remark 1 – *There exists a trade-off between attack effectiveness and trigger evasiveness (in terms of transparency), which is especially evident for complex data.*

4.3.2.2 Effectiveness vs. Evasiveness (Model)

Further, we consider the evasiveness of trojan models, which is measured by their difference from benign models in terms of classifying clean inputs. One intriguing property of the attacks is the trade-off between maximizing the attack effectiveness with respect to trigger inputs and minimizing the influence over clean inputs. Here, we characterize this trade-off via varying the fraction of trigger inputs in the 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 4.4 visualizes the normalized *CAD*-*ASR* trade-off. Observe that the curves of all the attacks manifest strong convexity, indicating the "leverage" effects [19]: 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 2 – *The trade-off between attack effectiveness and model evasiveness demonstrates strong "leverage" effects.*

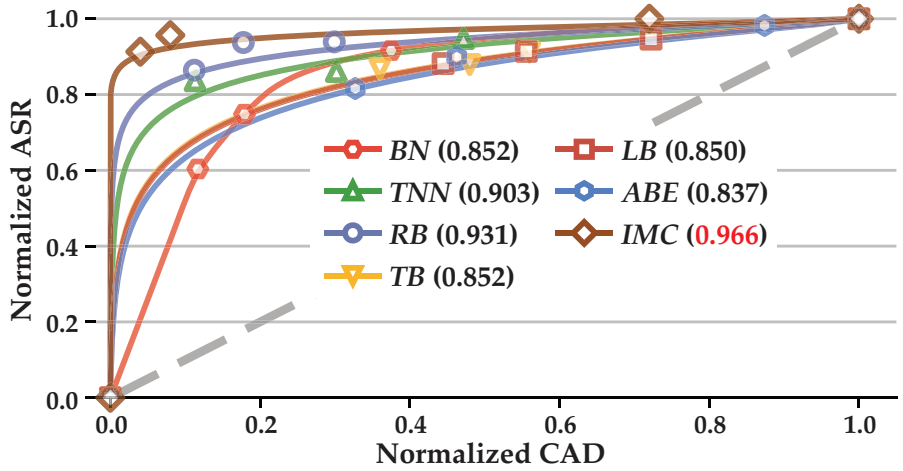

Figure 4.4: Trade-off between attack effectiveness and model evasiveness ($|m| = 3 \times 3$, $\alpha = 0.8$).

4.3.2.3 Effectiveness vs. Transferability

Next, we evaluate the transferability of different attacks to the downstream tasks. We consider two scenarios: (*i*) the pre-training and downstream tasks share the same dataset; and (*ii*) the downstream task uses a different dataset.

Transferability (classifier) – In (*i*), we focus on evaluating the impact of downstream-classifier selection and fine-tuning strategy on the attacks. We consider 5 different classifiers (1/2 fully-connected layer, Bayes, SVM, and Random Forest) and 3 fine-tuning strategies (none, partial tuning, and full tuning). Notably, the adversary is unaware of such settings.

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 4.5. Impact of fine-tuning and downstream-classifier selection.

Table 4.5 compares the *ASR* of 5 attacks with respect to varying downstream classifiers and fine-tuning strategies. Observe that fine-tuning has a large impact on attack effectiveness. For instance, the *ASR* of TNN drops by 62.5% from partial- to full-tuning. Yet, LB and IMC are less sensitive to fine-tuning, due to their optimization strategies. Also, note that the attack performance seems agnostic to the downstream classifier.

This may be explained by that the downstream classifier in practice tends to manifest "pseudo-linearity" [6].

Transferability (data) – In (*ii*), we focus on evaluating the transferability of the attacks across different datasets.

Transfer Setting	Attack				
	BN	TNN	RB	LB	IMC
C → C	94.5 (0.99)	100.0 (1.0)	100.0 (1.0)	100.0 (1.0)	100.0 (1.0)
C → I	8.4 (0.29)	7.8 (0.29)	8.6 (0.30)	8.2 (0.30)	9.4 (0.32)
I → I	90.0 (0.98)	95.2 (0.99)	94.6 (0.98)	97.4 (0.99)	98.4 (1.0)
I → C	77.0 (0.84))	26.9 (0.72)	11.0 (0.38)	10.0 (0.38)	14.3 (0.48)

Table 4.6. *ASR* and *TMC* of transfer attacks across CIFAR10 (C) and ImageNet (I) ($|m| = 3\times3$, $\alpha = 0.0$).

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

We evaluate the effectiveness of transferring attacks across two datasets, CIFAR10 and ImageNet, with results summarized in Table 4.6. 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) [80]; 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 poor transferability.

Remark 3 – *Most attacks transfer across classifiers; however, weaker attacks demonstrate higher transferability across datasets.*

4.3.3 Defense Evaluation

As the defenses from different categories bear distinct objectives (*e.g.*, detecting trigger inputs versus cleansing trojan models), below we evaluate each defense category separately.

4.3.3.1 Robustness vs. Utility

As input transformation and model sanitization mitigate backdoors in an attack-agnostic manner, while input filtering and model inspection have no direct influence on clean accuracy, we focus on evaluating attack-agnostic defenses to study the trade-off between robustness and utility preservation.

Robustness – With the no-defense (vanilla) case as reference, we compare different defences in terms of attack rate deduction (*ARD*) and trojan misclassification confidence (*TMC*), with results shown in Table 4.7. We have the following observations: (*i*) MP and AR are the most robust methods in the categories of input transformation and model sanitization, respectively. (*ii*) FP seems robust 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.

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

Utility – We now measure the impact of defenses on the accuracy of classifying clean inputs. Table 4.8 summarizes the results. With the vanilla setting as the baseline, most defenses tend to negatively affect clean accuracy, yet with varying impact. For instance, across all the cases, FP attains the least *CAD* across all the cases, mainly due to its fine-tuning; 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. Notably, comparing with Table 4.7, while MP and AR seem generically effective against all the attacks, they also suffer over 10% *CAD*, indicating the trade-off between robustness and utility preservation.

Remark 4 – *The design of attack-agnostic defenses faces the trade-off between robustness and utility preservation.*

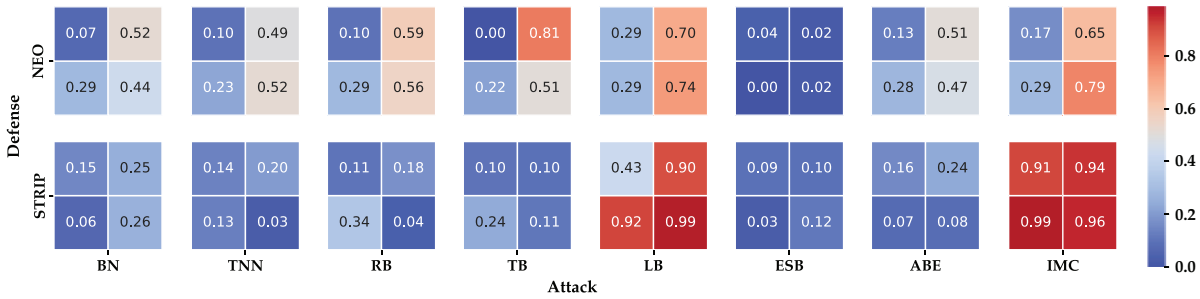

Figure 4.5: TPR of NEO and STRIP under varying trigger definition (left: $|m| = 3 \times 3$, right: $|m| = 6 \times 6$; lower: $\alpha = 0.0$, upper: $\alpha = 0.8$).

4.3.3.2 Detection Accuracy of Different Attacks

We evaluate the effectiveness of input filtering by measuring its accuracy in detecting trigger inputs.

Detection accuracy – For each attack, we randomly generate 100 pairs of trigger-clean inputs and measure the true positive (*TPR*) and false positive (*FPR*) rates of STRIP and NEO, two input filtering methods. To make comparison, we fix *FPR* as 0.05 and report *TPR* in Table 4.9.

We have the following findings. (*i*) STRIP is particularly effective 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 4.1). (*ii*) NEO is effective against most attacks to a limited

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 4.9. TPR of NEO and STRIP (FPR = 0.05, $\alpha = 0.0$).

extent (less than 0.3 *TPR*), but especially effective against ESB (over 0.6 *TPR*), due to its requirement for recognizing the trigger pattern precisely to overwrite the model prediction.

Impact of trigger definition – We also evaluate the impact of trigger definition on input filtering, with results in Figure 4.5. With fixed trigger transparency, NEO constantly attains higher *TPR* under larger triggers; in comparison, STRIP seems less sensitive but also less effective under larger triggers. This is attributed to the difference of their detection rationale: given input x, NEO searches for the "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.

Remark 5 – *The design of input filtering defenses needs to balance the detection accuracy with respect to different attacks.*

Defense	Attack							
	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 4.10. *AIV* of clean models and trojan models by various attacks.

4.3.3.3 Detection Accuracy vs. Recovery Capability

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

Detection Accuracy – Given defense $\mathcal{D}$ and model f, we measure the anomaly index value (*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.

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 4.11. *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).

The results are shown in Table 4.10. We observe: (*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), explainable by its trigger-model co-optimization strategy that 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 4.7).

Recovery Capability – 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 the recovered and injected triggers, with results shown in Table 4.11. While the ground-truth trigger has $MLN = 9$ ($\alpha = 0.0$, $|m| = 3\times3$), most defenses recover triggers of varying *MLN* and non-zero *MJS*, indicating that they recover triggers different from, yet overlapping with, the injected ones. In contrast to Table 4.10, NC and TABOR outperform ABS in trigger recovery, which may be explained by that while ABS relies on the most abnormal neuron to recover the trigger, the actual trigger may be embedded into multiple neurons. This may also be corroborated by that ABS attains the highest *MJS* on LB and IMC, which tend to generate triggers embedded in a few neurons (Table 4.9).

Remark 6 – *The design of model-inspection defenses faces the trade-off between the accuracy of detecting trojan models and the effectiveness of recovering trigger patterns.*

4.4 Exploration

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

4.4.1 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 § 4.3.2 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 4.12. Note that the trigger definition is inapplicable to ESB, due to its requirement for trigger size.

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

Interestingly, with single-pixel triggers, most attacks attain *ASR* comparable with the cases of larger triggers (*cf.* Figure 4.2). This implies the existence of universal, single-pixel perturbation [81] with respect to trojan models (but not clean models!), highlighting the mutual-reinforcement effects between trigger inputs and trojan models [19].

Remark 7 – *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. 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 4.13 compares *ASR* under the settings of regular and random triggers.

Trigger	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 4.13. Comparison of regular and random triggers.

Except for 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 8 – *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 [18]. Here, we validate this guidance by measuring the *NSR* (§ 4.2.2) of benign and trojan models before and after Fp, as shown in Table 4.14.

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 4.14. *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 4.1) tend to have higher *NSR* (*e.g.*, Imc). We thus conclude that the neuron-separation heuristic is in general valid.

Remark 9 – *The separation between the neurons activated by clean and trigger inputs is an indicator of attack effectiveness.*

4.4.2 Attack – Optimization

We now examine the optimization strategies used by the existing 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.

Table 4.15 compares the *ASR* of trojan models generated using the two strategies. Except for 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

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

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 10 – *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.

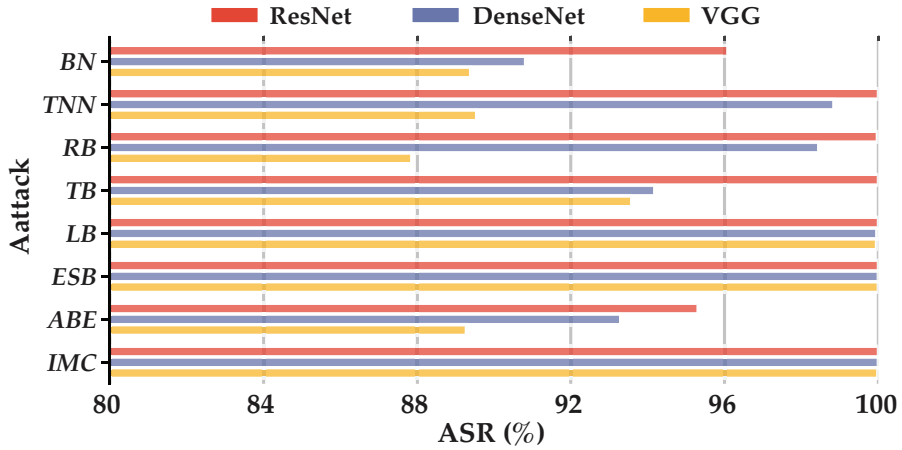

Figure 4.6: Impact of DNN architecture on attack efficacy.

We first compare the attack performance on three DNN models, VGG, ResNet, and DenseNet, with results shown in Figure 4.6. 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.*, dense 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.

We then consider the skip-connect structures and attempt to improve the gradient

backprop in training trojan models. In such networks, gradients propagate through both skip-connects and residual blocks. By setting the weights of gradients from skip-connects or residual blocks, it amplifies the gradient update towards inputs or model parameters [105]. 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 4.7.

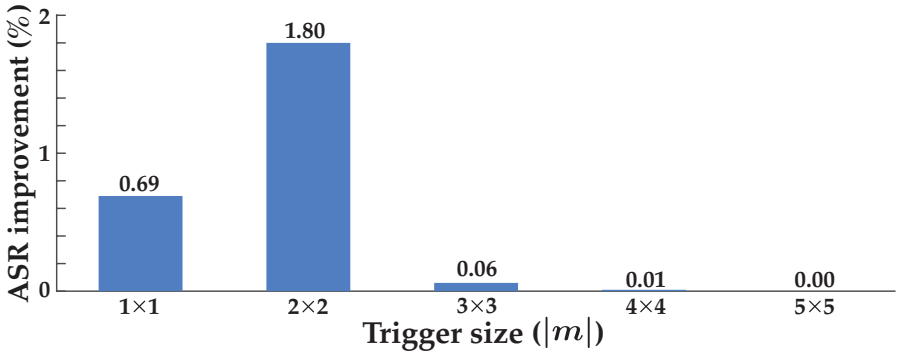

Figure 4.7: *ASR* improvement by reducing skip-connection gradients ($\alpha = 0.9$).

Observe that by reducing the skip-connection gradients, it marginally improves the *ASR* of IMC especially for small triggers (*e.g.*, $|m| = 2\times2$). We consider searching for the optimal γ to maximize attack efficacy as our ongoing work.

Remark 11 – *It is feasible to exploit skip-connect 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 4.16. Impact of mixing strategies on attack efficacy ($\alpha = 0.0$, $\lambda = 0.01$).

We observe in Table 4.16 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 an overly small learning rate. In comparison, batch-level mixing asserts every poisoning instance and its clean version must share the same batch, making the model focus more on the trigger as the classification evidence of target class.

Remark 12 – *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 4.8 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.

Remark 13 – *Optimizing the trigger pattern and the trojan model jointly leads to more effective attacks.*

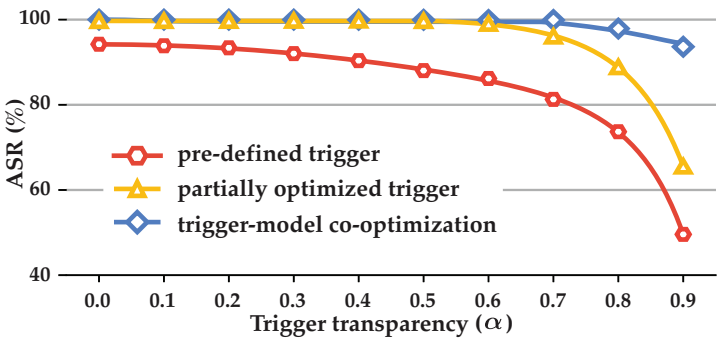

Figure 4.8: Impact of trigger optimization.

4.4.3 Defense – Evadability

RQ$_8$: *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 4.2. Specifically, we adapt IMC to each defense.

We compare the efficacy of non-adaptive and adaptive IMC, as shown in Figure 4.9. Observe that across all the cases, the adaptive IMC significantly outperforms the non-adaptive one. For instance, under $|m| = 6\times6$, 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 14 – *Most existing defenses are potentially evadable by adaptive attacks.*

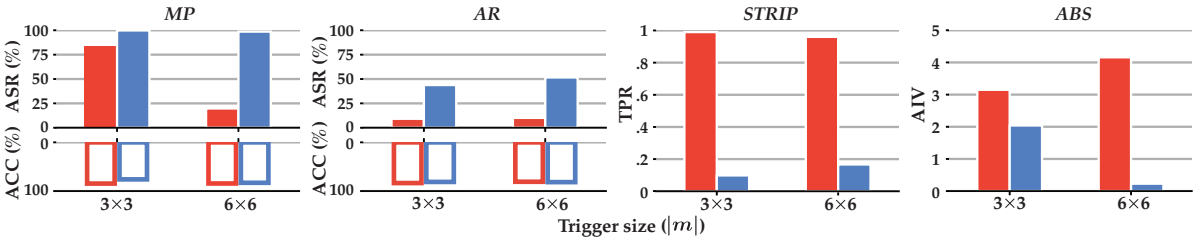

Figure 4.9: Performance of non-adaptive and adaptive IMC against representative defenses ($\alpha = 0.0$).

4.4.4 Defense – Interpretability

RQ$_9$: *Does interpretability help mitigate backdoor attacks?* – The interpretability of DNNs explain how they make predictions for given inputs [106, 107]. Recent studies [108, 109] 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 [106] of each input with respect to both models and ground and target classes, with an example shown in Figure 4.10.

We measure the difference (ℓ_1-norm normalized by image size) of attribution maps of clean and trigger inputs. Observe in Table 4.17 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 [110].

Remark 15 – *It seems promising to exploit model interpretability to enhance defense robustness.*

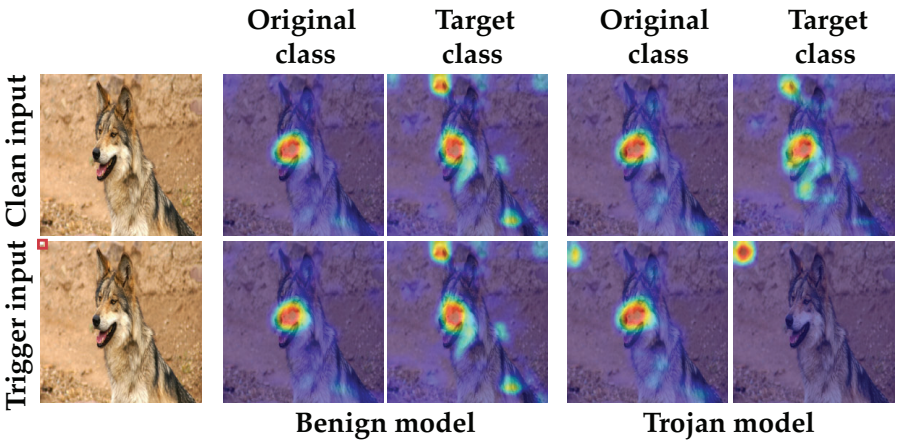

Figure 4.10: Sample attribution maps of clean and trigger inputs with respect to benign and trojan models ($\alpha = 0.0$, ImageNet).

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

Table 4.17. Heatmap difference of clean and trigger inputs ($\alpha = 0.0$).

4.5 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.

Chapter 5
The Security Risks of AutoML

5.1 Introduction

Automated Machine Learning (AutoML) represents a new paradigm of applying ML techniques in real-world settings. For given tasks, AutoML automates the pipeline from raw data to deployable ML models, covering model design [40], optimizer selection [111], and parameter tuning [112]. The use of AutoML greatly simplifies the development of ML systems and propels the trend of ML democratization. Many IT giants have unveiled their AutoML frameworks, such as Microsoft Azure AutoML, Google Cloud AutoML, and IBM Watson AutoAI.

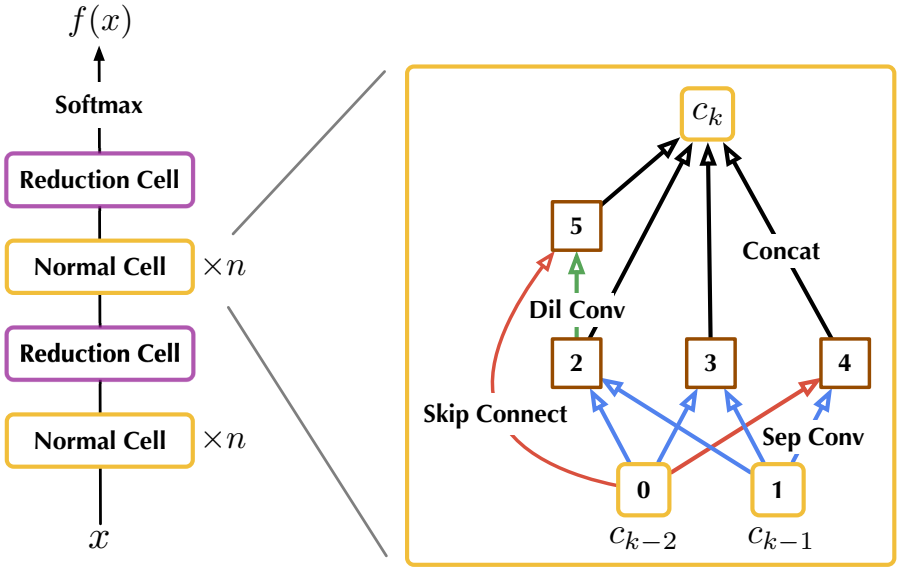

Figure 5.1: Cell-based neural architecture search.

In this dissertation, we focus on one primary task of AutoML, Neural Architecture

Search (NAS), which aims to find performant deep neural network (DNN) architectures[1] tailored to given tasks. For instance, as illustrated in Figure 5.1, cell-based NAS constructs a model by repeating the motif of a cell structure following a pre-specified template, wherein a cell is a topological combination of operations (*e.g.*, 3×3 convolution). With respect to the given task, NAS optimizes both the topological structure and the operation assignment. It is shown that in many tasks, NAS finds models that remarkably outperform manually designed ones [42, 43, 46, 47].

Yet, in contrast to the intensive research on improving the capabilities of NAS, its security implications are fairly unexplored. As ML systems are becoming the new targets for malicious attacks [55], the lack of understanding about the potential risks of NAS is highly concerning, given its surging popularity in security-sensitive applications. Specifically,

RQ1 – *Does NAS introduce new weaknesses, compared with the conventional ML practice?*

RQ2 – *If so, what are the possible root causes of such vulnerability?*

RQ3 – *Further, how would ML practitioners mitigate such drawbacks in designing and operating NAS?*

The answers to these key questions are crucial for the use of NAS in security-sensitive domains (*e.g.*, cyber-security, finance, and healthcare).

Our work – This work represents a solid initial step towards answering such questions.

A1 - First, through an extensive empirical study of 10 representative NAS methods, we show that compared with their manually designed counterparts, NAS-generated models tend to suffer greater vulnerability to various malicious manipulations such as adversarial evasion [48,49], model poisoning [50], backdoor injection [16,18], functionality stealing [51], and label-only membership inference [54]. The findings suggest that NAS is likely to incur larger attack surfaces, compared with the conventional ML practice.

A2 - Further, with both empirical and analytical evidence, we provide possible explanations for the above observations. Intuitively, due to the prohibitive search space and training cost, NAS tends to prematurely evaluate the quality of candidate models before their convergence. This practice favors models that converge fast at early training stages, resulting in architectural properties that facilitate various attacks (*e.g.*, high loss smoothness and low gradient variance). Our analysis not only reveals the relationships between model characteristics and attack vulnerability but also suggests the inherent

[1] In the following, when the context is clear, we use the terms of "model" and "architecture" exchangeably.

connections underlying different attacks.

A3 - Finally, we discuss potential remedies. Besides post-NAS mitigation (*e.g.*, adversarial training [48]), we explore in-NAS strategies that build attack robustness into the NAS process, such as increasing cell depth and suppressing skip connects. We show that while such strategies mitigate the vulnerability to a certain extent, they tend to incur non-trivial costs of search efficiency and model performance. We deem understanding the fundamental trade-off between model performance, attack robustness and search efficiency as an important topic for further investigation.

Contributions – To our best knowledge, this work represents the first study on the potential risks incurred by NAS (and AutoML in general) and reveals its profound security implications. Our contributions are summarized as follows.

– We demonstrate that compared with conventional ML practice, NAS tends to introduce larger attack surfaces with respect to a variety of attacks, which raises severe concerns about the use of NAS in security-sensitive domains.

– We provide possible explanations for such vulnerability, which reveal the relationships between architectural properties (*i.e.*, gradient smoothness and gradient variance) and attack vulnerability. Our analysis also hints at the inherent connections underlying different attacks.

– We discuss possible mitigation to improve the robustness of NAS-generated models under both in-situ and ex-situ settings. This discussion suggests the necessity of improving the current practice of designing and operating NAS, pointing to several research directions.

5.2 Measurement

To investigate the security risks incurred by NAS, we empirically compare the vulnerability of NAS-generated and manually designed models to the aforementioned attacks.

5.2.1 Experimental Setting

We first introduce the setting of the empirical evaluation.

Datasets – In the evaluation, we primarily use 3 datasets that have been widely used to benchmark NAS performance in recent work [42–44, 46, 118]: CIFAR10 [84] – it consists of 32×32 color images drawn from 10 classes (*e.g.*, 'airplane'); CIFAR100 – it is essentially the CIFAR10 dataset but divided into 100 fine-grained classes; ImageNet – it

	Architecture	CIFAR10	CIFAR100	ImageNet32
Manual Architecture	*BiT* [113]	96.6%	80.6%	72.1%
	DenseNet [104]	96.7%	80.7%	73.6%
	DLA [114]	96.5%	78.0%	70.8%
	ResNet [87]	96.6%	79.9%	67.1%
	ResNext [115]	96.7%	80.4%	67.4%
	VGG [103]	95.1%	73.9%	62.3%
	WideResNet [116]	96.8%	81.0%	73.9%
NAS Architecture	*AmoebaNet* [45]	96.9%	78.4%	74.8%
	DARTS [43]	97.0%	81.7%	76.6%
	DrNAS [47]	96.9%	80.4%	75.6%
	ENAS [42]	96.8%	79.1%	74.0%
	NASNet [41]	97.0%	78.8%	73.0%
	PC-DARTS [117]	96.9%	77.4%	74.7%
	PDARTS [118]	97.1%	81.0%	75.8%
	SGAS [46]	97.2%	81.2%	76.8%
	SNAS [44]	96.9%	79.9%	75.5%
	Random [119]	96.7%	78.6%	72.2%

Table 5.1. Accuracy of representative NAS-generated and manually designed models on benchmark datasets.

is a subset of the ImageNet dataset [120], downsampled to images of size 32×32 in 60 classes.

NAS methods – We consider 10 representative cell-based NAS methods, which cover a variety of search strategies: (1) *AmoebaNet* [45] applies an evolutionary approach to generate candidate models; (2) *DARTS* [43] is the first differentiable method using gradient descent to optimize both architecture and model parameters; (3) *DrNAS* [47] formulates differentiable NAS as a Dirichlet distribution learning problem; (4) *ENAS* [42] reduces the search cost via parameter sharing among candidate models; (5) *NASNet* [41] searches for cell structures transferable across different tasks by re-designing the search space; (6) *PC-DARTS* [117] improves the memory efficiency by restricting operation selection to a subset of edges; (7) *PDARTS* [118] gradually grows the number of cells to reduce the gap between the model depth at the search and evaluation phases; (8) *SGAS* [46] selects the operations in a greedy, sequential manner; (9) *SNAS* [44] reformulates reinforcement learning-based NAS to make it differentiable; and (10) *Random* [119] randomly samples candidate models from the pre-defined search space.

NAS search space – We define the default search space similar to *DARTS* [43], which consists of 10 operations including: *skip-connect*, 3×3 *max-pool*, 3×3 *avg-pool*, 3×3 *sep-conv*, 5×5 *sep-conv*, 7×7 *sep-conv*, 3×3 *dil-conv*, 5×5 *dil-conv*, $1 \times 7 - 7 \times 1$ *conv*, and *zero*.

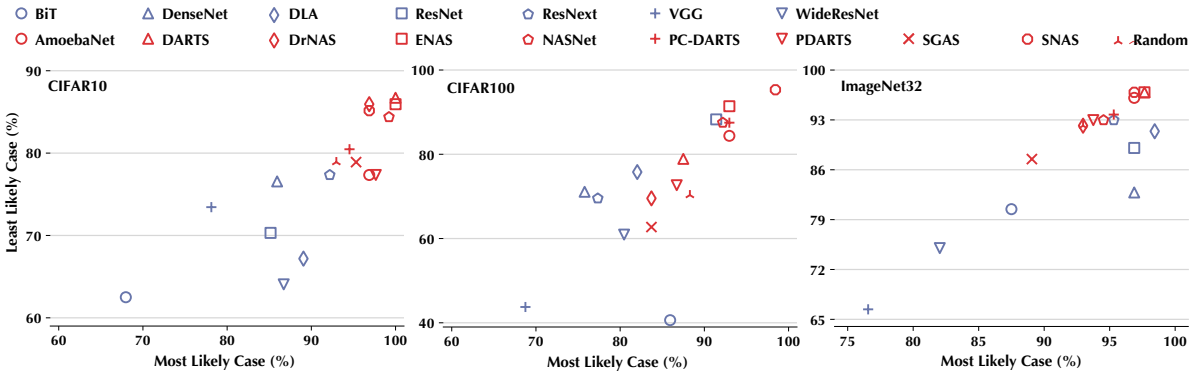

Figure 5.2: Performance of adversarial evasion (PGD) against NAS and manual models under the least and most likely settings.

Manual models – For comparison, we use 7 representative manually designed models that employ diverse architecture designs: (1) *BiT* [113] uses group normalization and weight standardization to facilitate transfer learning; (2) *DenseNet* [104] connects all the layers via skip connects; (3) *DLA* [114] applies deep aggregation to fuse features across layers; (4) *ResNet* [87] uses residual blocks to facilitate gradient back-propagation; (5) *ResNext* [115] aggregates transformations of the same topology; (6) *VGG* [103] represents the conventional deep convolution structures; and (7) *WideResNet* [116] decrease the depth and increases the width of *ResNet*.

Training – All the models are trained using the following setting: epochs = 600, batch size = 96, optimizer = SGD, gradient clipping threshold = 5.0, initial learning rate = 0.025, and learning rate scheduler = Cosine annealing. The accuracy of all the models on the benchmark datasets is summarized in Table 5.1. Observe that the NAS models often outperform their manual counterparts.

5.2.2 Experimental Results

Next, we empirically compare the vulnerability of NAS-generated and manually designed models to various attacks.

Adversarial evasion – We exemplify with the projected gradient descent (`PGD`) attack [48]. Over each dataset, we apply the attack on a set of 1,000 inputs randomly

sampled from the test set and measure the attack success rate as:

$$\text{Attack Success Rate (ASR)} = \frac{\text{\# Successful trials}}{\text{\# Total trials}} \tag{5.1}$$

A trial is marked as successful if it is classified as its target class within maximum iterations.

Let $f_c(x)$ be the probability that model f assigns to class c with respect to input x. To assess the full spectrum of vulnerability, we consider both "difficult" and "easy" cases for the adversary. Specifically, given input x, we rank the output classes c's according to their probabilities $f_c(x)$ as $c_1, c_2, \ldots, c_m$, where c_1 is x's current classification; the difficult case refers to that the adversary aims to change x's classification to the least likely class c_m, while the easy case refers to that the adversary aims to change x's classification to the second most likely class c_2.

Figure 5.2 illustrates the attack effectiveness against both NAS and manual models. We have the following observations. First, across all the datasets, the NAS models seem more vulnerable to adversarial evasion. For instance, on CIFAR10, the attack attains over 90% and 75% *ASR* against the NAS models in the most and least likely cases, respectively. Second, compared with the manual models, the *ASR* of NAS models demonstrates more evident clustered structures, implying their similar vulnerability. Finally, the vulnerability of NAS models shows varying patterns on different datasets. For instance, the measures of NAS models show a larger variance on CIFAR100 compared with CIFAR10 and ImageNet (especially in the least likely case), which may be explained by that its larger number of classes results in more varying "degree of difficulty" for the attack.

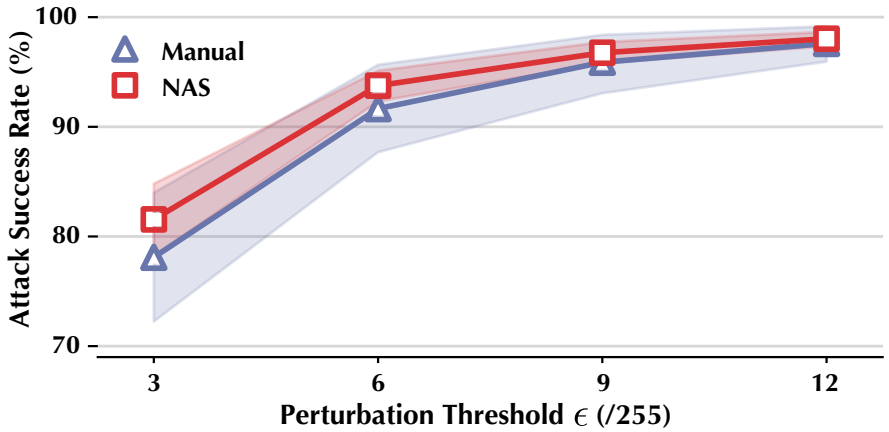

Figure 5.3: Impact of perturbation threshold (ϵ) on the vulnerability of different models with respect to `PGD` on CIFAR10.

We also evaluate the impact of perturbation threshold (ϵ) on the attack vulnerability. Figure 5.3 shows the *ASR* of untargeted PGD as a function of ϵ against different models on CIFAR10 (with perturbation step $\alpha = \epsilon/3$). We have the following observations. First, across different settings, the manual models consistently outperform the NAS models in terms of robustness. Second, this vulnerability gap gradually decreases with ϵ, as the *ASR* on both NAS and manual models approaches 100%. Third, compared with the manual models, the measures of NAS models show a smaller variance, indicating the commonality of their vulnerability.

Further, by comparing the sets of adversarial examples to which different models are vulnerable, we show the commonality and difference of their vulnerability. We evaluate PGD ($\epsilon = 4/255$) against different models on CIFAR10 in the least likely case. For each model, we collect the set of adversarial examples successfully generated from 1,000 random samples. Figure 5.4 plots the distribution of inputs with respect to the number of successfully attacked models.

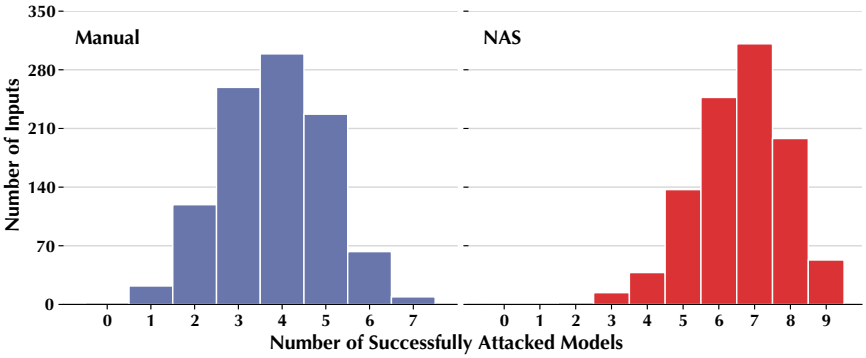

Figure 5.4: Distribution of inputs with respect to the number of successfully attacked models (PGD with $\epsilon = 4/255$ on CIFAR10).

Overall, PGD generates more successful adversarial examples against the NAS models than the manual models. Moreover, there are more inputs that lead to successful attacks against multiple NAS models. For instance, over 300 inputs lead to successful attacks against 7 NAS models; in contrast, the number is less than 10 in the case of manual models. We may thus conclude that the vulnerability of NAS models to adversarial evasion seems fairly similar, pointing to potential associations with common causes.

We also consider alternative adversarial evasion attacks other than PGD. We use natural evolutionary strategies (NES) [121], a black-box attack in which the adversary has only query access to the target model f and generates adversarial examples using a derivative-free optimization approach. Specifically, at each iteration, it generates n_{query} symmetric data points in the vicinity of current input x by sampling from a normal

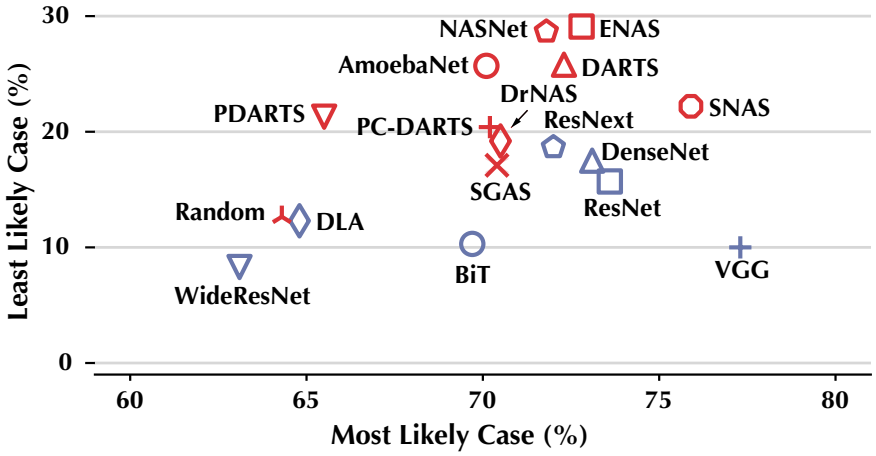

Figure 5.5: Performance of adversarial evasion (NES) against NAS and manual models under the least and most likely settings.

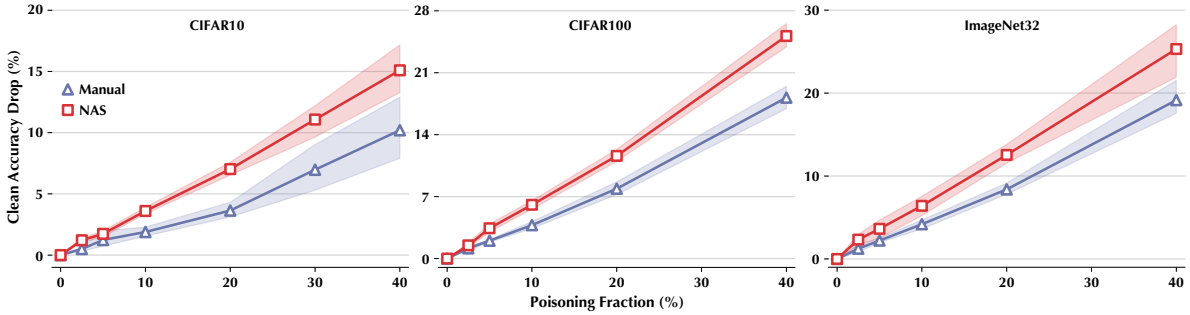

Figure 5.6: Performance of model poisoning against NAS and manually designed models under varying poisoning fraction p_{pos}.

distribution, retrieves their predictions from f, and estimates the gradient $\hat{g}(x)$ as:

$$\hat{g}(x) = \frac{1}{\sigma n_{\text{query}}} \sum_{j=1}^{\lceil n_{\text{query}}/2 \rceil} (f(x + \sigma u_j) - f(x - \sigma u_j)) u_j \tag{5.2}$$

where each sample u_j is sampled from the standard normal distribution $\mathcal{N}(0, I)$, and σ is the sampling variance.

We evaluate the vulnerability of different models to NES under the same setting of Figure 5.2 (with $n_{\text{query}} = 400$) on CIFAR10, with results shown in Figure 5.5. In general, the NAS models show higher vulnerability to NES, especially in the least likely case, indicating that the vulnerability gap between NAS and manual models also generalizes to black-box adversarial evasion attacks.

Model poisoning – In this set of experiments, we evaluate the impact of poisoning attacks on the performance of NAS and manual models. We assume that a fraction p_{pos} of the training data is polluted by randomly changing the class of each input. We

measure the performance of various models with respect to varying poisoning fraction p_{pos}, in comparison with the case of clean training data (*i.e.*, $p_{\text{pos}} = 0$). We define the metric of clean accuracy drop:

$$\begin{aligned} &\text{Clean Accuracy Drop (CAD)} \\ &= \text{Acc. of original model} - \text{Acc. of polluted model} \end{aligned} \tag{5.3}$$

Figure 5.6 compares the *CAD* of different models as p_{pos} increases from 0% to 40%. The results are average over the families of NAS and manual models. We have the following observations. First, as expected, larger p_{pos} causes more performance degradation on all the models. Second, with fixed p_{pos}, the NAS models suffer more significant accuracy drop. For instance, on CIFAR100, with p_{pos} fixed as 20%, the *CAD* of NAS models is 4% higher than the manual models. Further, the *CAD* gap between NAS and manual models enlarges as p_{pos} increases.

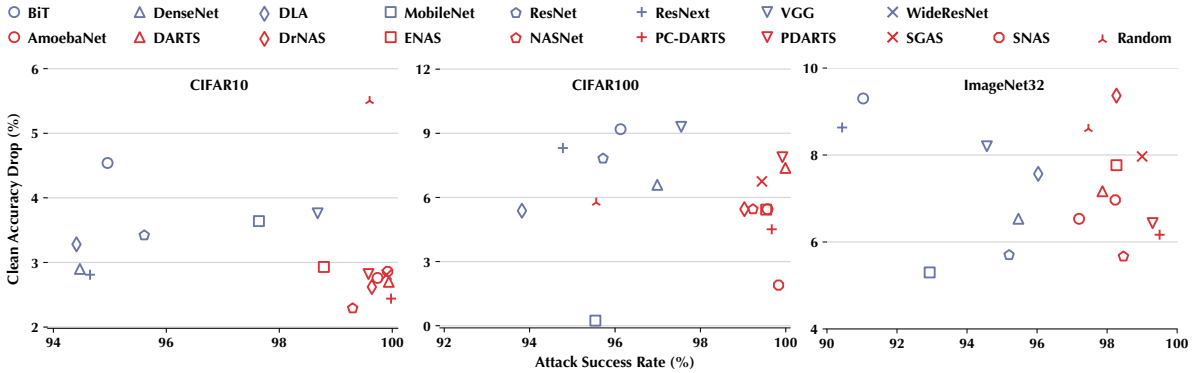

Figure 5.7: Performance of backdoor injection (TrojanNN) against NAS and manually designed models.

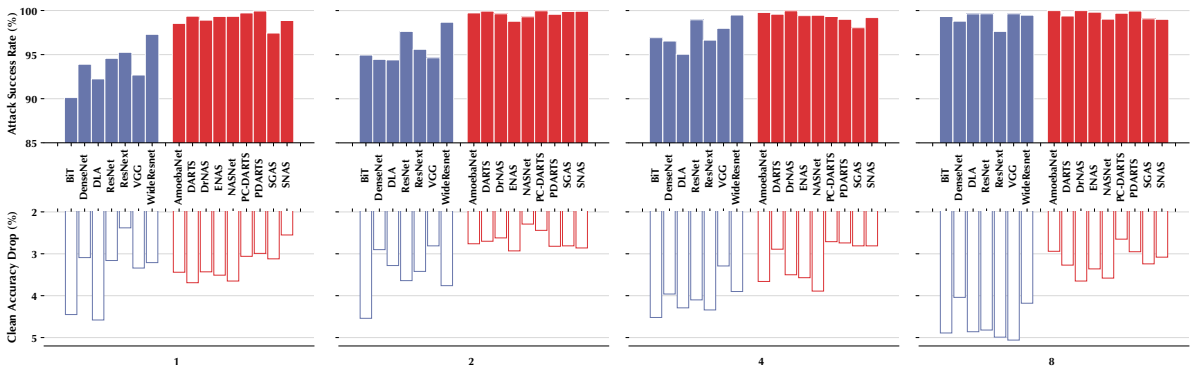

Figure 5.8: Impact of the number of target neurons (n_{neuron}) on the vulnerability of different models with respect to `TrojanNN` on CIFAR10.

Backdoor injection – Next, we compare the vulnerability of NAS and manual models to neural backdoor attacks [16, 18, 122]. Recall that in backdoor injection, the

adversary attempts to forge a trojan model f^* (typically via perturbing a benign model f) that is sensitive to a specific trigger but behaves normally otherwise. We thus measure the attack effectiveness using two metrics: attack success rate (*ASR*), which is the fraction of trigger-embedded inputs successfully classified by f^* to the target class desired by the adversary; clean accuracy drop (*CAD*), which is the accuracy difference of f^* and f on clean inputs.

We consider `TrojanNN` [18], a representative backdoor attack, as the reference attack model. By optimizing both the trigger r and trojan model f^*, `TrojanNN` enhances other backdoor attacks (*e.g.*, `BadNet` [16]) that employ fixed triggers. Figure 5.7 plots the *ASR* and *CAD* of all the models, in which the results are average over 1,000 inputs randomly sampled from each testing set. Observe that the attack seems more effective against the NAS models across all the datasets. For instance, on CIFAR10, the attack achieves close to 100% *ASR* on most NAS models with *CAD* below 3%. Further, similar to adversarial evasion and model poisoning, the measures of most NAS models (except *Random*) are fairly consistent, indicating their similar vulnerability. Recall that *Random* samples models from the search space; thus, the higher vulnerability of NAS models is likely to be associated with their particular architectural properties.

We further evaluate the impact of the number of target neurons (n_{neuron}) in `TrojanNN`. Recall that `TrojanNN` optimizes the trigger with respect to n_{neuron} target neurons. Figure 5.8 plots the *ASR* and *CAD* of `TrojanNN` against different models under varying setting n_{neuron}. First, across all the settings of n_{neuron}, `TrojanNN` consistently attains more effective attacks (*i.e.*, higher *ASR* and lower *CAD*) on the NAS models than the manual models. Second, as n_{neuron} varies from 1 to 4, the difference of *ASR* between NAS and manual models decreases, while the difference of *CAD* tends to increase. This may be explained as follows: optimizing triggers with respect to more target neurons tends to lead to more effective attacks (*i.e.*, higher *ASR*) but also result in a larger impact on clean inputs (*i.e.*, higher *CAD*). However, this trade-off is less evident on the NAS models, implying their higher capabilities to fit both poisoning and clean data.

From the experiments above, we may conclude that compared with manual models, NAS models tend to be more vulnerable to backdoor injection attacks, especially under more restricted settings (*e.g.*, fewer target neurons).

Functionality stealing – We now evaluate how various models are subject to functionality stealing, in which each model f as a black box only allowing query access: given input x, f returns its prediction $f(x)$. The adversary attempts to re-construct a functionally similar model f^* based on the query-prediction pairs $\{(x, f(x))\}$.

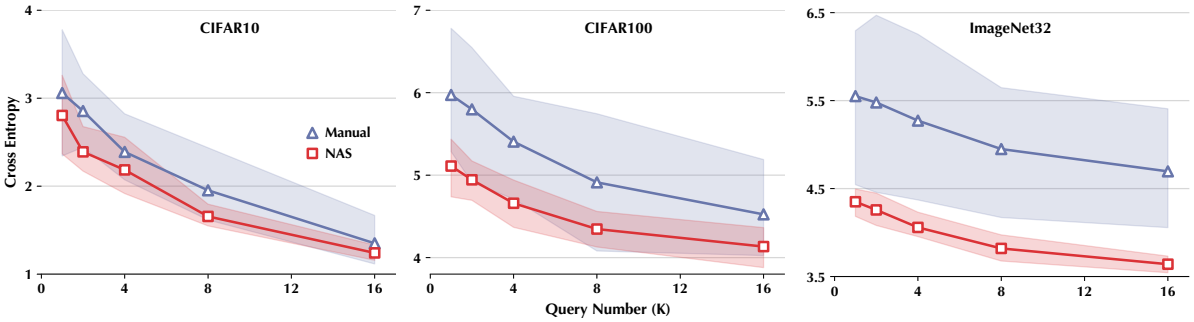

Figure 5.9: Performance of functionality stealing against NAS and manually designed models under the victim architecture-aware setting.

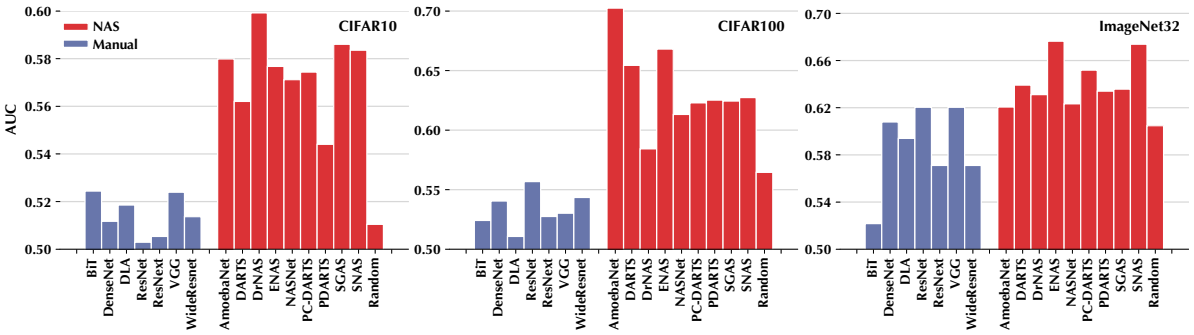

Figure 5.10: Performance of label-only membership inference attacks against NAS and manually designed models.

We consider two scenarios: (*i*) f and f^* share the same architecture; and (*ii*) the adversary is unaware of f's architecture and instead uses a surrogate architecture in f^*. We apply Knockoff [51], a representative functionality stealing attack that adaptively generates queries to probe f to re-construct f^*. We evaluate the attack using the average cross entropy (ACE) of f's and f^*'s predictions on the testing set, with lower cross entropy indicating more effective stealing.

Victim f \ Replicate f^*		Manual		NAS	
		ResNet	*DenseNet*	*DARTS*	*ENAS*
Manual	*ResNet*	1.286	1.509	1.377	1.455
	DenseNet	1.288	1.245	1.231	1.381
NAS	*DARTS*	1.272	1.115	1.172	1.125
	ENAS	1.259	1.050	1.115	1.151

Table 5.2. Performance of functionality stealing against NAS and manual models under the victim architecture-agnostic setting.

Figure 5.9 summarizes the attack effectiveness under the victim architecture-aware setting. Across all the datasets, the attack achieves smaller ACE on the NAS models with much lower variance, in comparison with the manual models. This implies that most

NAS models share similar vulnerability to functionality stealing. We further consider the victim architecture-agnostic setting. For each pair of models, we assume one as f and the other as f^*, and measure the attack effectiveness. The results on CIFAR10 (with the query number fixed as 8K) are summarized in Table 5.2. Observe that with the replicate model f^* fixed, the NAS models as the victim model f result in lower ACE, implying that it tends to be easier to steal the functionality of NAS models, regardless of the architecture of the replicate model.

Membership inference – Recall that in membership inference, the adversary attempts to infer whether the given input x appears in the training set of the target model f. The inference accuracy serves as an indicator of f's privacy leakages. Next, we conduct membership inference attacks on various models to assess their privacy risks.

There are two possible scenarios: (*i*) f's prediction $f(x)$ contains the confidence score $f_c(x)$ of each class c; and (*ii*) $f(x)$ contains only the label $c^* = \arg\max_c f_c(x)$. As (*i*) can be mitigated by removing the confidence scores in $f(x)$ [53], here, we focus on (*ii*). Under the class-only setting, we apply the decision boundary-based attack [54], which determines x's membership (in the training data) by estimating its distance to the nearest decision boundary using label-only adversarial attacks (*e.g.*, HopSkipJump [123]). In each case, we evaluate the attack over 2,000 inputs, half randomly sampled from the training set and the other half from the testing set, and measure the attack effectiveness using the area under the ROC curve (*AUC*), with the estimated distance as the control of false and true positive rates.

Figure 5.10 compares the attack performance against different models. Notably, the attack achieves higher *AUC* scores on the NAS models. For instance, the average scores on the NAS and manual models on CIFAR100 differ by more than 0.05, while the scores on the manual models are close to random guesses (*i.e.*, 0.5). Moreover, most NAS models (except *Random*) show similar vulnerability. Also, note that the manual models seem more vulnerable on ImageNet, which may be explained as follows: compared with CIFAR10 and CIFAR100, ImageNet is a more challenging dataset (see Table 5.1); the models thus tend to overfit the training set more aggressively, resulting in their higher vulnerability to membership inference.

Remark 1 – *Compared with their manually designed counterparts, NAS-generated models tend to be more vulnerable to various malicious manipulations.*

5.3 Analysis

The empirical evaluation in § 5.2 reveals that compared with manually designed models, NAS-generated models tend to be more vulnerable to a variety of attacks. Next, we provide possible explanations for such phenomena.

5.3.1 Architectural Properties of Trainability

We hypothesize that the greater vulnerability of NAS models stems from their key design choices.

Popular NAS methods often evaluate the performance of a candidate model prematurely before its full convergence during the search. For instance, *DARTS* [43] formulate the search as a bi-level optimization problem, in which the inner objective optimizes a given model; to save the computational cost, instead of solving this objective exactly, it approximates the solution using a single training step, which is far from its full convergence. Similar techniques are applied in other popular NAS methods (*e.g.*, [42, 45]). As the candidate models are not evaluated on their performance at convergence, NAS tends to favor models with higher "trainability" – those converge faster during early stages – which result in candidate models demonstrating the following key properties:

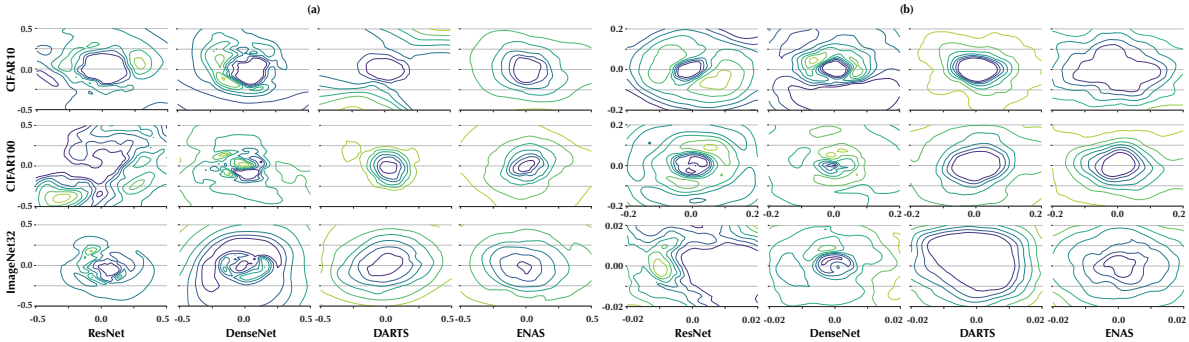

Figure 5.11: Loss contours of NAS-generated models (*DARTS*, *ENAS*) and manually designed ones (*ResNet*, *DenseNet*) in (a) parameter space and (b) input space.

High loss smoothness – *The loss landscape of NAS models tends to be smooth, while the gradient provides effective guidance for optimization. Therefore, NAS models are amenable to training using simple, first-order optimizers.*

Low gradient variance – *The gradient of NAS models with respect to the given distribution tends to have low variance. Therefore, the stochastic gradient serves as a reliable estimate of the true gradient, making NAS models converge fast.*

Note that the loss smoothness captures the geometry of the loss function in the parameter space (or the input space), while the gradient variance measures the difference between the gradients with respect to different inputs. While related, the former dictates whether a model is easy to train if the gradient direction is known and the latter dictates whether it is easy to estimate the gradient direction reliably.

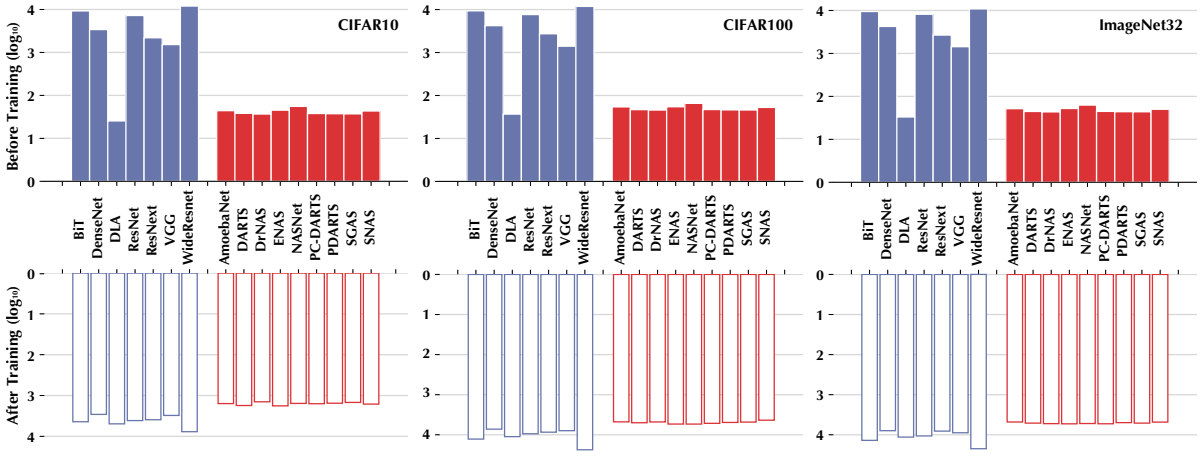

Figure 5.12: Gradient variance of NAS-generated and manually designed models before and after training.

Next, we empirically validate the above hypotheses by comparing the gradient smoothness and variance of NAS-generated and manually designed models.

Loss smoothness – A loss function $\mathcal{L}$ is said to have L-Lipschitz ($L > 0$) continuous gradient with respect to θ if it satisfies $\|\nabla\mathcal{L}(\theta) - \nabla\mathcal{L}(\theta')\| \leq L\|\theta - \theta'\|$ for any θ, θ'. The constant L controls $\mathcal{L}$'s smoothness. While it is difficult to directly measure L of given model f, we explore its loss contour [124], which quantifies the impact of parameter perturbation on $\mathcal{L}$. Specifically, we measure the loss contour of model f as follows:

$$\Gamma(\alpha, \beta) = \mathcal{L}(\theta^* + \alpha d_1 + \beta d_2) \tag{5.4}$$

where θ^* denotes the local optimum, d_1 and d_2 are two random, orthogonal directions as the axes, and α and β represent the perturbation steps along d_1 and d_1, respectively. Notably, the loss contour effectively approximates the loss landscape in a two-dimensional space [125].

Figure 5.11(a) visualizes the loss contours of NAS (*DARTS* and *ENAS*) and manual (*ResNet* and *DenseNet*) models across different datasets. Observe that the NAS models tend to demonstrate a flatter loss landscape. Similar phenomena are observed with respect to other models. This observation may explain why the gradient of NAS models gives more effective guidance for minimizing the loss function, leading to their higher trainability.

Further, for the purpose of the analysis in § 5.3, we extend the loss smoothness in the parameter space to the input space. We have the following result to show their fundamental connections.

Theorem 1. If the loss function $\mathcal{L}$ has L-Lipschitz continuous gradient with respect to θ and the weight matrix of each layer of the model is normalized [126], then $\mathcal{L}$ has $L/\sqrt{n}$-Lipschitz continuous gradient with respect to the input, where n is the input dimensionality.

Empirically, we define f's loss contour with respect to a given input-class pair (x, y) as follows:

$$\Gamma_{(x,y)}(\alpha, \beta) = \ell(f(x + \alpha d_1 + \beta d_2), y) \tag{5.5}$$

where d_1 and d_2 are two random, orthogonal directions in the input space. Figure 5.11(b) visualizes the loss contours of NAS and manual models in the vicinity of randomly sampled inputs. It is observed that NAS models also demonstrate higher loss smoothness in the input space, compared with the manual models.

Gradient variance – Meanwhile, the variance of the gradient with respect to inputs sampled from the underlying distribution quantifies the noise level of the gradient estimate used by stochastic training methods (*e.g.*, SGD) [127]. Formally, let g be the stochastic gradient. We define the gradient variance as follows (where the expectation is taken with respect to the given distribution):

$$\mathrm{Var}(g) = \mathbb{E}\left[\|g - \mathbb{E}[g]\|_2^2\right] \tag{5.6}$$

Assuming g is an unbiased estimate of the true gradient, $\mathrm{Var}(g)$ measures g's expected deviation from the true gradient. Smaller $\mathrm{Var}(g)$ implies lower noise level, thereby more stable updating of the model parameters θ.

In Figure 5.12, we measure the gradient variance of various models before training (with Kaiming initialization [128]) and after training is complete. It is observed in all the cases that at initialization, the gradient variance of NAS models is more than two

orders of magnitude smaller than the manual models and then grows gradually during the training; in comparison, the gradient variance of manual models does not change significantly before and after training. This observation may explain why the stochastic gradient of NAS models gives a reliable estimate of the true gradient, making them converge fast at early training phases.

5.3.2 Explanations of Attack Vulnerability

We now discuss how the vulnerability of NAS models to various attacks can be attributed to the properties of high loss smoothness and low gradient variance.

Adversarial evasion – The vulnerability to adversarial evasion is mainly attributed to the sensitivity of model prediction $f(x)$ to the perturbation of input x. Under the white-box setting, the adversary typically relies on the gradient to craft the adversarial input x^*. For instance, `PGD` [48] crafts x^* by iteratively updating the input using the following rule:

$$x_{t+1} = \Pi_{x+\mathcal{B}_\epsilon} \left(x_t + \alpha \operatorname{sign} \left(\nabla_x \ell(f(x_t), y) \right) \right) \tag{5.7}$$

where x_t is the perturbed input after the t-th iteration, Π denotes the projection operator, $\mathcal{B}_\epsilon$ represents the allowed set of perturbation (parameterized by ϵ), and α is the perturbation step. Apparently, the attack effectiveness relies on whether the gradient $\nabla_x \ell(f(x_t), y)$ is able to provide effective guidance for perturbing x_t.

As shown in § 5.2.2, compared with the manual models, due to the pursuit of higher trainability, the NAS models often demonstrate a smoother loss landscape wherein the gradient at each point represents effective optimization direction; thus, the NAS models tend to be more vulnerable to gradient-based adversarial evasion. Notably, this finding also corroborates existing studies (*e.g.*, [10]) on the fundamental tension between designing "linear" models that are easier to train and designing "nonlinear" models that are more resistant to adversarial evasion.

The similar phenomena observed in the case of black-box attacks (*e.g.*, NES) may be explained as follows: to perform effective perturbation, black-box attacks often rely on indirect gradient estimation, while the high loss smoothness and low gradient variance of NAS models lead to more accurate and efficient (with fewer queries) gradient estimation.

Model poisoning – The vulnerability to model poisoning can be attributed to the sensitivity of model training to the poisoning data in the training set. Here, we analyze how the property of low gradient variance impacts this sensitivity.

For a given dataset $\mathcal{D}$, let $\mathcal{L}(\theta)$ be the loss of a model f_θ parameterized by θ with

respect to $\mathcal{D}$:

$$\mathcal{L}(\theta) \triangleq \frac{1}{|\mathcal{D}|}\sum\nolimits_{(x,y)\in\mathcal{D}} \ell(f_\theta(x), y) \tag{5.8}$$

Further, let θ^* represent f's (local) optimum with respect to $\mathcal{D}$. With θ initialized as θ_0, consider T-step SGD updates with the t-th step update as:

$$\theta_{t+1} = \theta_t - \alpha_t g_t \tag{5.9}$$

where α_t is the step size and g_t is the gradient estimate. We have the following result describing the convergence property of θ_t $(t = 1, \ldots, T)$.

Theorem 2 ([127]). Assuming that (i) $\mathcal{L}(\theta)$ is continuous and differentiable, with its gradient bounded by Lipschitz constant L, (ii) the variance of gradient estimate g_t $(t = 1, \ldots, T)$ is bounded by σ^2, and (iii) θ_t is selected as the final parameters with probability proportional to $2\alpha_t - L\alpha_t^2$. Then, the output parameters $\theta_{\bar{t}}$ satisfies:

$$\mathbb{E}\left[\mathcal{L}(\theta_{\bar{t}}) - \mathcal{L}(\theta^*)\right] \leq \frac{\|\theta_0 - \theta^*\|^2 + \sigma^2 \sum_{t=1}^{T} \alpha_t^2}{\sum_{t=1}^{T}(2\alpha_t - L\alpha_t^2)} \tag{5.10}$$

where the expectation is defined with respect to the selection of $\bar{t}$ and the gradient variance.

Intuitively, Theorem 2 describes the properties that impact the fitting of model f to the given dataset $\mathcal{D}$. As shown in § 5.2.2, compared with the manual models, the NAS models tend to have both higher loss smoothness (*i.e.*, smaller L) and lower gradient variance (*i.e.*, smaller σ). Therefore, the NAS models tend to fit $\mathcal{D}$ more easily. Recall that in model poisoning, $\mathcal{D}$ consists of both clean data $\mathcal{D}_{\text{trn}}$ and poisoning data $\mathcal{D}_{\text{pos}}$, fitting to $\mathcal{D}$ more tightly implies more performance drop over the testing data, which may explain the greater vulnerability of NAS models to model poisoning.

Backdoor injection – Recall that in backdoor injection, the adversary forges a trojan model f^* that is sensitive to a trigger pattern r such that any input x, once embedded with r, tends to be misclassified to a target class t: $f^*(x + r) = t$. To train f^*, the adversary typically pollutes the training data $\mathcal{D}_{\text{trn}}$ with trigger-embedded inputs.

Intuitively, this attack essentially exploits the attack vectors of adversarial evasion that perturbs x at inference time and model poisoning that pollutes $\mathcal{D}_{\text{trn}}$ at training time. Therefore, the vulnerability of NAS models to both attack vectors naturally results in their vulnerability to backdoor injection. Due to the space limitations, we omit the detailed analysis here.

Functionality stealing – Recall that in functionality stealing (*e.g.*, Knockoff [51]), the adversary (adaptively) generates queries to probe the victim model f to replicate a functionally similar one f^*. For instance, Knockoff encourages queries that are certain by f, diverse across different classes, and disagreed by f^* and f.

The effectiveness of such attacks depends on f's loss landscape with respect to the underlying distribution; intuitively, the complexity of the loss landscape in the input space implies the hardness of fitting f^* to f based on a limited number of queries. Thus, given their high loss smoothness, the NAS models tend to be more vulnerable to functionality stealing.

Membership inference – It is shown in § 5.2 that the NAS models seem more vulnerable to membership inference, especially under the label-only setting in which only the prediction labels are accessible. The adversary thus relies on signals such as input x's distance to its nearest decision boundary $\text{dist}(x, f(x))$; intuitively, if x appears in the training set, $\text{dist}(x, f(x))$ is likely to be below a certain threshold. Concretely, the HopSkipJump attack [123] is employed in [54] to estimate $\text{dist}(x, f(x))$ via iteratively querying f to find point x_t on the decision boundary using bin search, walking along the boundary using the estimated gradient at x_t, and finding point x_{t+1} to further reduce the distance to x, which is illustrated in Figure 5.13.

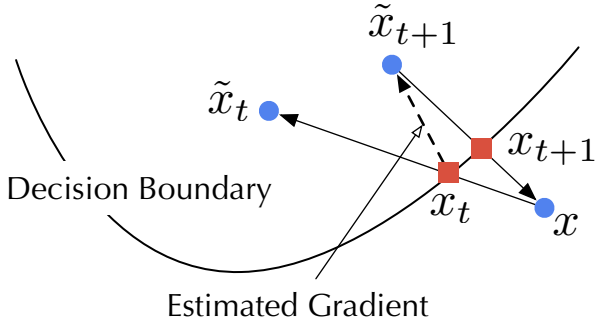

Figure 5.13: Illustration of the HopSkipJump attack.

The effectiveness of this attack hinges on (*i*) the quality of estimated gradient and (*ii*) the feasibility of descending along the decision boundary. For the NAS models, the gradient estimate tends to be more accurate due to the low gradient variance, while the decision boundary tends to be smoother due to the high loss smoothness, which may explain the greater vulnerability of NAS models to label-only membership inference attacks.

Remark 2 – *The high loss smoothness and low gradient variance of NAS-generated models may account for their greater vulnerability to various attacks.*

5.3.3 Connections of Various Attacks

It is shown above that the vulnerability of NAS models to various attacks may be explained by their high loss smoothness and low gradient variance, which bears an intriguing implication: different attacks may also be inherently connected via these two factors.

Specifically, most existing attacks involve input or model perturbation. For instance, adversarial evasion, regardless of the white- or black-box setting, iteratively computes (or estimates) the gradient and performs perturbation accordingly; backdoor injection optimizes the trigger and model jointly, requiring to estimate, based on the gradient, how the model responds to the updated trigger.

The effectiveness of such attacks thus highly depends on (*i*) how to estimate the gradient at each iteration and (*ii*) how to use the gradient estimate to guide the input or model perturbation. Interestingly, gradient variance and loss smoothness greatly impact (*i*) and (*ii*), respectively: low gradient variance enables the adversary to accurately estimate the gradient, while high loss smoothness allows the adversary to use such estimate to perform effective perturbation.

Remark 3 – *The effectiveness of various attacks is inherently connected through loss smoothness and gradient variance.*

5.4 Discussion

In § 5.2 and § 5.3, we reveal the relationships between the trainability of NAS-generated models and their vulnerability to various attacks, two key questions remain: (*i*) what are the architectural patterns associated with such vulnerability? and (*iii*) what are the potential strategies to remedy the vulnerability incurred by the current NAS practice? In this section, we explore these two questions and further discuss the limitations of this work.

5.4.1 Architectural Weaknesses

As shown in § 5.3, the vulnerability of NAS models is potentially related to their high loss smoothness and low gradient variance, which stem from the preference for models of higher trainability. We now discuss how such preference is reflected in the concrete architectural patterns, which we examine from two aspects, namely, topology selection

and operation selection.

Architecture	Cell Depth	Cell Width	# Skip connects
AmoebaNet	4	$3c$	2
DARTS	3	$3c$	3
DrNAS	4	$2c$	1
ENAS	2	$5c$	2
NASNet	2	$5c$	1
PC-DARTS	2	$4c$	1
PDARTS	4	$2c$	2
SGAS	3	$2c$	1
SNAS	2	$4c$	4

Table 5.3. The cell depth and width, and the number of skip connects of representative NAS-generated models (the width of each intermediate node is assumed to be c).

Topology selection – Recent studies [129] suggest that in cell-based NAS, the preference for models with faster convergence often results in wide, shallow cell structures. As shown in Figure 5.1, the cell depth is defined as the number of connections along the longest path from the input nodes to the output node; the width of each intermediate node is defined as the number of channels for convolution operators or the number of features for linear operators, while the cell width is defined as the total width of intermediate nodes connected to the input nodes. Table 5.3 summarizes the cell depth and width of NAS models used in our evaluation. It is observed that the cell structures of most NAS models are both shallow (with an average depth of 2.8) and wide (with an average width of $3.3c$), where the width of each intermediate node is assumed to be c.

It is shown in [129] that under similar settings (*i.e.*, the same number of nodes and connections), wide and shallow cells tend to demonstrate higher trainability. This observation is also corroborated by the recent theoretical studies on the convergence of wide neural networks [130]: neural networks of infinite width tend to evolve as linear models using gradient descent optimization.

Operation selection – The preference for higher trainability also impacts the selection of operations (*e.g.*, 3×3 convolution versus skip connection) on the connections within the cell structure, and typically favors skip connects over other operations.

Recall that differential NAS methods [43, 46, 47] typically apply continuous relaxation on the search space to enable direct gradient-based optimization. The operation on each connection is modeled as a softmax of all possible operations $\mathcal{O}$ and discretized by selecting the most likely one $\arg\max_{o\in\mathcal{O}}\alpha_o$. It is shown in [131] that in well-optimized

models, the weight of skip connection α_{skip} often exceeds other operations, leading to its higher chance of being selected. This preference takes effect in our context, as NAS models tend to converge fast at early training stages. Table 5.3 summarizes the number of skip connects in each cell of representative NAS models. Observe that most NAS models have more than one skip connection in each cell.

The operation of skip connection is originally designed to enable back-propagation in DNNs [87, 104]. As a side effect, accurate gradient estimation also facilitates attacks that exploit gradient information [105]. Thus, the over-use of skip connects in NAS models also partially accounts for their vulnerability to such attacks.

Remark 4 – *NAS-generated models often feature wide and shallow cell structures as well as overuse of skip connects.*

5.4.2 Potential Mitigation

We now discuss potential mitigation to remedy the vulnerability incurred by the NAS practice. We consider enhancing the robustness of NAS models under both post-NAS and in-NAS settings. In post-NAS mitigation, we explore using existing defenses against given attacks to enhance NAS models, while with in-NAS mitigation, we explore building attack robustness into the NAS process directly.

Post-NAS mitigation – As a concrete example, we apply adversarial training [88, 132], one representative defense against adversarial evasion, to enhance the robustness of NAS models. Intuitively, adversarial training improves the robustness of given model f by iteratively generating adversarial inputs with respect to its current configuration and updating f to correctly classify such inputs.

Figure 5.14 compares the effectiveness of adversarial training on various models over CIFAR10. For each model, we measure its accuracy (in terms of accuracy drop from before adversarial training) and robustness (in terms of the success rate of the untargeted `PGD` attack). Observe that a few NAS models (*e.g.*, *DARTS*) show accuracy and robustness comparable with manual models, while the other NAS models (*e.g.*, *DrNAS*) underperform in terms of both accuracy and robustness, which may be explained by their diverse architectural patterns associated with adversarial training (*e.g.*, dense connections, number of convolution operations, and cell sizes) [133]. This disparity also implies that adversarial training may not be a universal solution for improving the robustness of all the NAS models.

In-NAS mitigation – We further investigate how to build attack robustness into

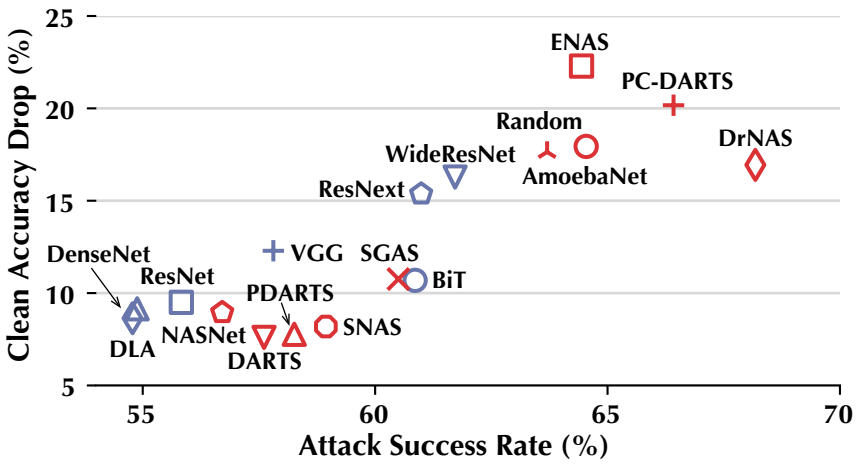

Figure 5.14: Effectiveness of adversarial training on various models over CIFAR10.

the NAS process directly. Motivated by the analysis in § 5.4.1, we explore two potential strategies.

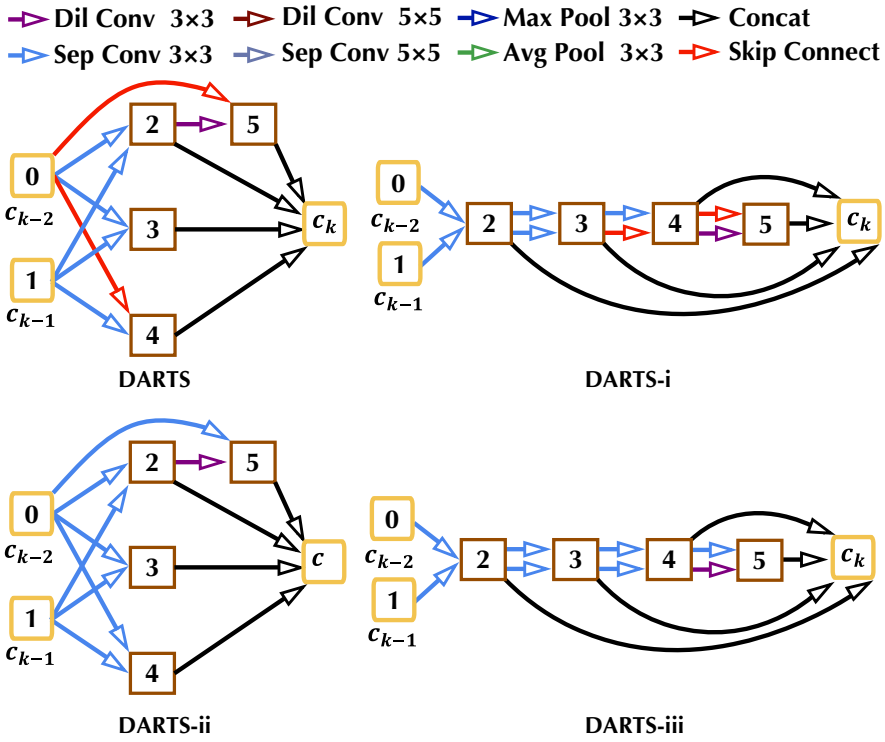

Figure 5.15: Illustration of cell structures of *DARTS*, *DARTS-i*, *DARTS-ii*, and *DARTS-iii*.

(*i*) Increasing cell depth – As the vulnerability of NAS models tends to be associated with their wide and shallow cell structures, we explore increasing their cell depth. To this end, we may re-wire existing NAS models or modify the performance measure of candidate models. For the latter case, we may increase the number of training epochs

before evaluation. For instance, *DARTS*, without fully optimizing model parameters θ with respect to architecture parameters α, uses a single-step gradient descent ($n_{\text{step}} = 1$) to approximate the solution [43]. We improve the approximation by increasing the number of training steps (*e.g.*, $n_{\text{step}} = 5$) at the cost of additional search time.

(*ii*) Suppressing skip connects – As the vulnerability of NAS models is also associated with skip connects, we explore purposely reducing their overuse. To this end, we may replace the skip connects in existing NAS models with other operations (*e.g.*, convolution) or modify their likelihood of being selected in the search process. For the latter case, at each iteration, we may multiply the weight of skip connection α_{skip} by a coefficient $\gamma \in (0, 1)$ in Eq. 2.7.

We evaluate the effectiveness of such strategies within the *DARTS* framework. Let *DARTS-i*, *DARTS-ii*, and *DARTS-iii* be the variants of *DARTS* after applying the strategies of (*i*), (*ii*), and (*i*) and (*ii*) combined. Figure 5.15 compares their cell structures. Notably, *DARTS-i* features a cell structure deeper than *DARTS* (5 versus 2), while *DARTS-ii* and *DARTS-iii* substitute the skip connects in *DARTS* and *DARTS-i* with 3×3 convolution, respectively.

Architecture	Evasion		Backdoor		Membership
	ASR (M)	ASR (L)	ASR	CAD	AUC
DARTS	100.0%	86.7%	99.9%	2.7%	0.562
DARTS-i	88.3%	72.7%	90.4%	4.6%	0.534
DARTS-ii	93.0%	75.0%	98.8%	3.0%	0.531
DARTS-iii	82.0%	65.6%	84.2%	4.6%	0.527

Table 5.4. Vulnerability of *DARTS* and its variants to adversarial evasion (M - most likely case, L - least likely case), backdoor injection, and membership inference on CIFAR10.

Table 5.4 compares their vulnerability to adversarial evasion, backdoor injection, and membership inference on CIFAR10. The experimental setting is identical to that in § 5.2. Observe that both strategies may improve the robustness of NAS models against these attacks. For instance, combining both strategies in *DARTS-iii* reduces the *AUC* score of membership inference from 0.562 to 0.527. Similar phenomena are observed in the case of model extraction attacks. As shown in Figure 5.16, increasing the cell depth significantly augments the robustness against model extraction, while suppressing skip connects further improves it marginally.

Yet, such strategies seem to have a negative impact on the robustness against model poisoning. As shown in Figure 5.17, both strategies, especially increasing the cell depth, tends to exacerbate the attack vulnerability. This may be explained by that while more

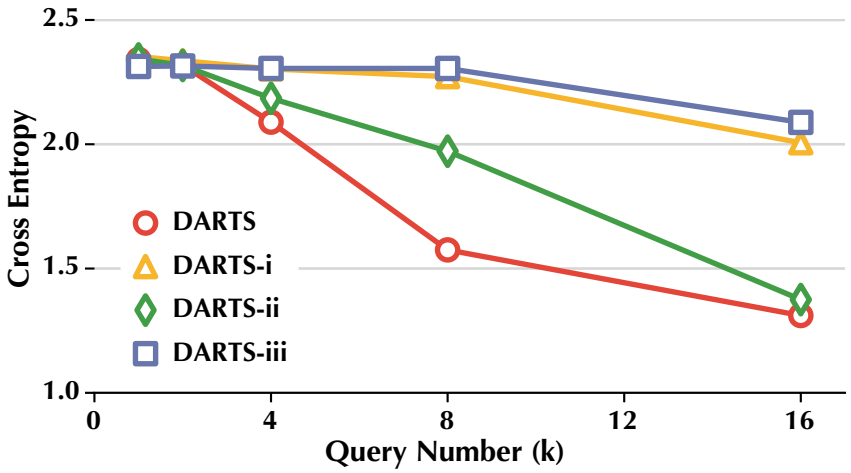

Figure 5.16: Vulnerability of *DARTS* and its variants to model extraction on CIFAR10.

difficult to fit the poisoning data, it is also more difficult to fit deeper structures to the clean data, which results in a significant accuracy drop. This may also explain why the backdoor injection attack has higher *CAD* on *DARTS-i* and *DARTS-iii* as shown in Table 5.4. The observation also implies a potential trade-off between the robustness against different attacks in designing NAS models.

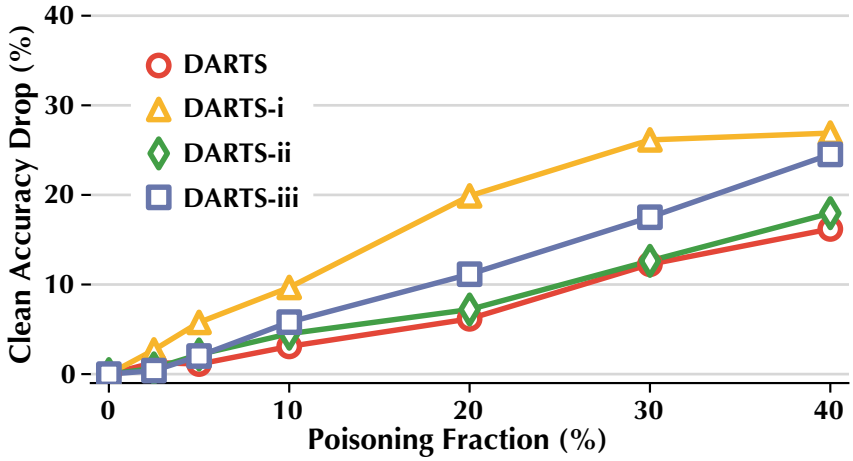

Figure 5.17: Vulnerability of *DARTS* and its variants to model poisoning on CIFAR10.

Remark 5 – *Simply increasing cell depth and/or suppressing skip connects may only partially mitigate the vulnerability of NAS-generated models.*

5.4.3 Limitations

Next, we discuss the limitations of this work.

Alternative NAS frameworks – In this work, we mainly consider the cell-based search space adopted by recent NAS methods [42, 43, 47, 134, 135], while other methods have considered the global search space (*e.g.*, chain-of-layer structures) [77, 136]. Further, while we focus on the differentiable search strategy, there are other strategies including random search [78], Bayesian optimization [79], and reinforcement learning [41, 77, 137]. We consider exploring the vulnerability of models generated by alternative NAS frameworks as our ongoing research.

Other trainability metrics – In this work, we only consider loss smoothness and gradient variance as two key factors impacting the trainability (and vulnerability) of NAS models. There are other trainability metrics (*e.g.*, condition number of neural tangent kernel [138]) that are potentially indicative of attack vulnerability as well.

Robustness, accuracy, and search efficiency – It is revealed that the greater vulnerability incurred by NAS is possibly associated with the preference for models that converge fast at early training phases (*i.e.*, higher trainability). It is however unclear whether this observation implies fundamental conflicts between the factors of robustness, accuracy, and search efficiency; if so, is it possible to find an optimal balance between them? We consider answering these questions critical for designing and operating NAS in practical settings.

5.5 Conclusion

This work represents a systematic study on the security risks incurred by AutoML. From both empirical and analytical perspectives, we demonstrate that NAS-generated models tend to suffer greater vulnerability to various malicious manipulations, compared with their manually designed counterparts, which implies the existence of fundamental drawbacks in the design of existing NAS methods. We identify high loss smoothness and low gradient variance, stemming from the preference of NAS for models with higher trainability, as possible causes for such phenomena. Our findings raise concerns about the current practice of NAS in security-sensitive domains. Further, we discuss potential remedies to mitigate such limitations, which sheds light on designing and operating NAS in a more robust and principled manner.

Chapter 6
EVAS: Exploitable and Vulnerable Arch Search

6.1 Introduction

As a new paradigm of applying ML techniques in practice, automated machine learning (AutoML) automates the pipeline from raw data to deployable models, which covers model design, optimizer selection, and parameter tuning. The use of AutoML greatly simplifies the ML development cycles and propels the trend of ML democratization. In particular, neural architecture search (NAS), one primary AutoML task, aims to find performant deep neural network (DNN) arches[1] tailored to given datasets. In many cases, NAS is shown to find models remarkably outperforming manually designed ones [42, 43, 46].

In contrast to the intensive research on improving the capability of NAS, its security implications are largely unexplored. As ML models are becoming the new targets of malicious attacks [55], the lack of understanding about the risks of NAS is highly concerning, given its surging popularity in security-sensitive domains [139]. Towards bridging this striking gap, we pose the intriguing yet critical question:

Is it possible for the adversary to exploit NAS to launch previously improbable attacks?

This work provides an affirmative answer to this question. We present *exploitable and vulnerable arch search* (EVAS), a new backdoor attack that leverages NAS to find neural arches with inherent, exploitable vulnerability. Conventional backdoor attacks typically embed the malicious functions ("backdoors") into the space of model parameters. They often assume strong threat models, such as polluting training data [16, 18, 122] or

[1] In the following, we use "arch" for short of "architecture".

perturbing model parameters [140, 141], and are thus subject to defenses based on model inspection [26, 27] and data filtering [70]. In EVAS, however, as the backdoors are carried in the space of model arches, even if the victim trains the models using clean data and operates them in a black-box manner, the backdoors are still retained. Moreover, due to its independence of model parameters or training data, EVAS is naturally robust against defenses such as model inspection and input filtering.

To realize EVAS, we define a novel metric based on neural tangent kernel [138], which effectively indicates the exploitable vulnerability of a given arch; further, we integrate this metric into the NAS-without-training framework [138, 142]. The resulting search method is able to efficiently identify candidate arches without requiring model training or backdoor testing. To verify EVAS's empirical effectiveness, we evaluate EVAS on benchmark datasets and show: (*i*) EVAS successfully finds arches with exploitable vulnerability, (*ii*) the injected backdoors may be explained by arch-level "shortcuts" that recognize trigger patterns, and (*iii*) EVAS demonstrates high evasiveness, transferability, and robustness against defenses. Our findings show the feasibility of exploiting NAS as a new attack vector to implement previously improbable attacks, raise concerns about the current practice of NAS in security-sensitive domains, and point to potential directions to develop effective mitigation.

6.2 Evas

Next, we present EVAS, a new backdoor attack leveraging NAS to find neural arches with exploitable vulnerability. We begin by introducing the threat model.

6.2.1 Threat Model

A backdoor attack injects a hidden malicious function ("backdoor") into a target model [143]. The backdoor is activated once a pre-defined condition ("trigger") is present, while the model behaves normally otherwise. In a predictive task, the backdoor is often defined as classifying a given input to a class desired by the adversary, while the trigger can be defined as a specific perturbation applied to the input. Formally, given input x and trigger $r = (m, p)$ in which m is a mask and p is a pattern, the trigger-embedded input is defined as:

$$\tilde{x} = x \odot (1 - m) + p \odot m \tag{6.1}$$

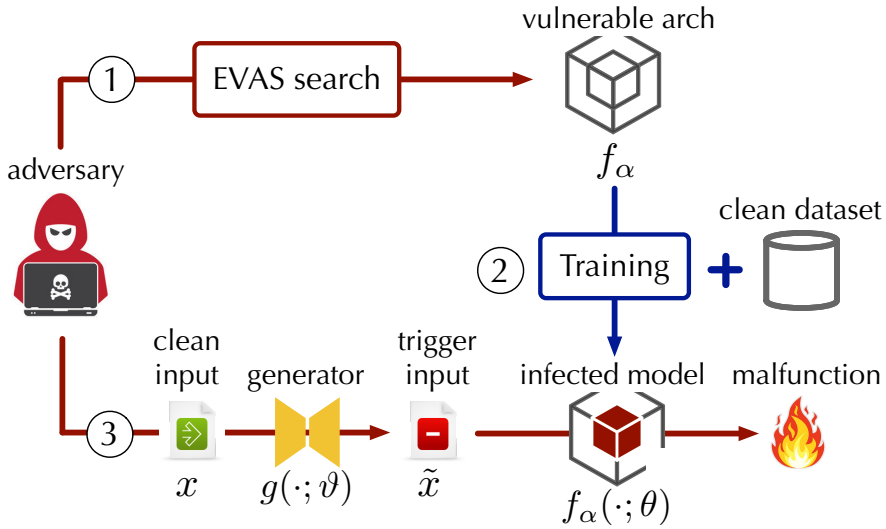

Figure 6.1: Attack framework of EVAS. (*1*) The adversary applies NAS to search for arches with exploitable vulnerability; (*2*) such vulnerability is retained even if the models are trained using clean data; (*3*) the adversary exploits such vulnerability by generating trigger-embedded inputs.

Let f be the backdoor-infected model. The backdoor attack implies that for given input-label pair (x, y), $f(x) = y$ and $f(\tilde{x}) = t$ with high probability, where t is the adversary's target class.

The conventional backdoor attacks typically follow two types of threat models: (*i*) the adversary directly trains a backdoor-embedded model, which is then released to and used by the victim user [18, 122, 140]; or (*ii*) the adversary indirectly pollutes the training data or manipulate the training process [16, 141] to inject the backdoor into the target model. As illustrated in Figure 6.1, in EVAS, we assume a more practical threat model in which the adversary only releases the exploitable arch to the user, who may choose to train the model from scratch using clean data or apply various defenses (*e.g.*, model inspection or data filtering) before or during using the model. We believe this represents a more realistic setting: due to the prohibitive computational cost of NAS, users may opt to use performant model arches provided by third parties, which opens the door for the adversary to launch the EVAS attack.

However, realizing EVAS represents non-trivial challenges including (*i*) how to define the trigger patterns? (*ii*) how to define the exploitable, vulnerable arches? and (*iii*) how to search for such arches efficiently? Below we elaborate on each of these key questions.

6.2.2 Input-Aware Triggers

Most conventional backdoor attacks assume universal triggers: the same trigger is applied to all the inputs. However, universal triggers can be easily detected and mitigated by current defenses [26, 27]. Moreover, it is shown that implementing universal triggers at the arch level requires manually designing "trigger detectors" in the arches and activating such detectors using poisoning data during training [144], which does not fit our threat model.

Instead, as illustrated in Figure 6.1, we adopt input-aware triggers [145], in which a trigger generator g (parameterized by ϑ) generates trigger r_x specific to each input x. Compared with universal triggers, it is more challenging to detect or mitigate input-aware triggers. Interestingly, because of the modeling capacity of the trigger generator, it is more feasible to implement input-aware triggers at the arch level. For simplicity, below we use $\tilde{x} = g(x; \vartheta)$ to denote both generating trigger r_x for x and applying r_x to x to generate the trigger-embedded input $\tilde{x}$.

6.2.3 Exploitable arches

In EVAS, we aim to find arches with backdoors exploitable by the trigger generator, which we define as the following optimization problem.

Specifically, let α and θ respectively denote f's arch and model parameters. We define f's training as minimizing the following loss:

$$\mathcal{L}_{\mathsf{trn}}(\theta, \alpha) \triangleq \mathbb{E}_{(x,y)\sim\mathcal{D}}\ell(f_\alpha(x;\theta), y) \tag{6.2}$$

where f_α denotes the model with arch fixed as α and $\mathcal{D}$ is the underlying data distribution. As θ is dependent on α, we define:

$$\theta_\alpha \triangleq \arg\min_\theta \mathcal{L}_{\mathsf{trn}}(\theta, \alpha) \tag{6.3}$$

Further, we define the backdoor attack objective as:

$$\mathcal{L}_{\mathsf{atk}}(\alpha, \vartheta) \triangleq \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\ell(f_\alpha(x;\theta_\alpha), y) + \lambda\ell(f_\alpha(g(x;\vartheta);\theta_\alpha), t)\right] \tag{6.4}$$

where the first term specifies that f works normally on clean data, the second term specifies that f classifies trigger-embedded inputs to target class t, and the parameter λ balances the two factors. Note that we assume the testing data follows the same

distribution $\mathcal{D}$ as the training data.

Overall, we consider an arch α^* having exploitable vulnerability if it is possible to find a trigger generator ϑ^*, such that $\mathcal{L}_{\mathsf{atk}}(\alpha^*, \vartheta^*)$ is below a certain threshold.

6.2.4 Search without Training

Searching for exploitable archs by directly optimizing Eq. 6.4 is challenging: the nested optimization requires recomputing θ (*i.e.*, re-training model f) in $\mathcal{L}_{\mathsf{trn}}$ whenever α is updated; further, as α and ϑ are coupled in $\mathcal{L}_{\mathsf{atk}}$, it requires re-training generator g once α is changed.

Motivated by recent work [142, 146–148] on NAS using easy-to-compute metrics as proxies (without training), we present a novel method of searching for exploitable arches based on *neural tangent kernel* (NTK) [149] without training the target model or trigger generator. Intuitively, NTK describes model training dynamics by gradient descent [130,149,150]. In the limit of infinite-width DNNs, NTK becomes constant, which allows closed-form statements to be made about model training. Recent work [151, 152] shows that NTK serves as an effective predictor of model "trainability" (*i.e.*, how fast the model converges at early training stages). Formally, considering model f (parameterized by θ) mapping input x to a probability vector $f(x;\theta)$ (over different classes), the NTK is defined as the product of the Jacobian matrix:

$$\Theta(x,\theta) \triangleq \left[\frac{\partial f(x;\theta)}{\partial \theta}\right]\left[\frac{\partial f(x;\theta)}{\partial \theta}\right]^{\intercal} \tag{6.5}$$

Let $\lambda_{\min}$ ($\lambda_{\max}$) be the smallest (largest) eigenvalue of the empirical NTK $\hat{\Theta}(\theta) \triangleq \mathbb{E}_{(x,y)\sim\mathcal{D}}\Theta(x,\theta)$. The condition number $\kappa \triangleq \lambda_{\max}/\lambda_{\min}$ serves as a metric to estimate model trainability [138], with a smaller conditional number indicating higher trainability.

In our context, we consider the trigger generator and the target model as an end-to-end model and measure the empirical NTK of the trigger generator under randomly initialized θ:

$$\hat{\Theta}(\vartheta) \triangleq \mathbb{E}_{(x,y)\sim\mathcal{D},\theta\sim P_{\theta_\circ}}\left[\frac{\partial f(g(x;\vartheta);\theta)}{\partial \vartheta}\right]\left[\frac{\partial f(g(x;\vartheta;\theta)}{\partial \vartheta}\right]^{\intercal} \tag{6.6}$$

where $P_{\theta_\circ}$ represents the initialization distribution of θ. Here, we emphasize that the measure should be independent of θ's initialization.

Intuitively, $\hat{\Theta}(\vartheta)$ measures the trigger generator's trainability with respect to a

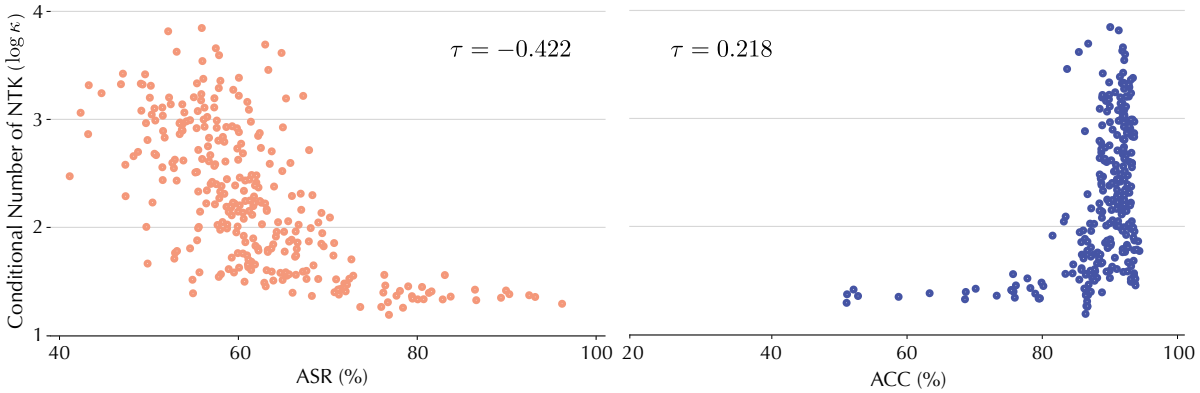

Figure 6.2: The conditional number of NTK versus the model performance (ACC) and vulnerability (ASR).

randomly initialized target model. The generator's trainability indicates the easiness of effectively generating input-aware triggers, implying the model's vulnerability to input-aware backdoor attacks. To verify the hypothesis, on the CIFAR10 dataset, we measure $\hat{\Theta}(\vartheta)$ with respect to 900 randomly generated arches as well as the model accuracy (ACC) on clean inputs and the attack success rate (ASR) on trigger-embedded inputs. Specifically, for each arch α, we first train the model f_α to measure ACC and then train the trigger generator g with respect to f_α on the same dataset to measure ASR, with results shown in Figure 6.2. Observe that the conditional number of $\hat{\Theta}(\vartheta)$ has a strong negative correlation with ASR, with a smaller value indicating higher attack vulnerability; meanwhile, it has a limited correlation with ACC, with most of the arches having ACC within the range from 80% to 95%.

Leveraging the insights above, we present a simple yet effective algorithm that searches for exploitable arches without training, which is a variant of regularized evolution [142, 153]. As sketched in Algorithm 3, it starts from a candidate pool $\mathcal{A}$ of n arches randomly sampled from a pre-defined arch space; at each iteration, it samples a subset $\mathcal{A}'$ of m arches from $\mathcal{A}$, randomly mutates the best candidate (*i.e.*, with the lowest score), and replaces the oldest arch in $\mathcal{A}$ with this newly mutated arch. In our implementation, the score function is defined as the condition number of Eq. 6.6; the arch space is defined to be the NATS-Bench search space [119], which consists of 5 atomic operators {*none*, *skip connect*, *conv* 1×1, *conv* 3×3, and *avg pooling* 3×3}; and the mutation function is defined to be randomly substituting one operator with another.

Algorithm 3: EVAS Attack

Input: n – pool size; m – sample size; score – score function; sample – subset sampling function; mutate – arch mutation function;

Output: exploitable arch

$\mathcal{A}, \mathcal{S}, \mathcal{T} = [], [], []$; `// candidate archs, scores, timestamps`

for $i \leftarrow 1$ **to** n **do**

  $\mathcal{A}[i] \leftarrow$ randomly generated arch, $\mathcal{S}[i] \leftarrow \text{score}(\mathcal{A}[i])$, $\mathcal{T}[i] \leftarrow 0$;

$best \leftarrow 0$;

while *maximum iterations not reached yet* **do**

  $i \leftarrow \arg\min_{k \in \text{sample}(\mathcal{A}, m)} \mathcal{S}[k]$; `// best candidate`

  $j \leftarrow \arg\max_{k \in \mathcal{A}} \mathcal{T}[k]$; `// oldest candidate`

  $\mathcal{A}[j] \leftarrow \text{mutate}(\mathcal{A}[i])$; `// mutate candidate`

  $\mathcal{S}[j] \leftarrow \text{score}(\mathcal{A}[j])$; `// update score`

  $\mathcal{T} \leftarrow \mathcal{T} + 1, \mathcal{T}[j] \leftarrow 0$; `// update timestamps`

  if $\mathcal{S}[j] < \mathcal{S}[best]$ **then** $best \leftarrow j$;

return $\mathcal{A}[best]$;

6.3 Evaluation

We conduct an empirical evaluation of EVAS on benchmark datasets under various scenarios. The experiments are designed to answer the following key questions: (*i*) *does it work?* – we evaluate the performance and vulnerability of the arches identified by EVAS; (*ii*) *how does it work?* – we explore the dynamics of EVAS search as well as the characteristics of its identified arches; and (*ii*) *how does it differ?* – we compare EVAS with conventional backdoors in terms of attack evasiveness, transferability, and robustness.

6.3.1 Experimental Setting

Datasets. In the evaluation, we primarily use three datasets that have been widely used to benchmark NAS methods [42–44, 46, 118]: CIFAR10 [84], which consists of 32×32 color images drawn from 10 classes; CIFAR100, which is similar to CIFAR10 but includes 100 finer-grained classes; and ImageNet16, which is a subset of the ImageNet dataset [3] down-sampled to images of size 16×16 in 120 classes.

Search space. We consider the search space defined by NATS-Bench ([154]), which consists of 5 operators {*none*, *skip connect*, *conv* 1×1, *conv* 3×3, and *avg pooling* 3×3} defined among 4 nodes, implying a search space of 15,625 candidate arches.

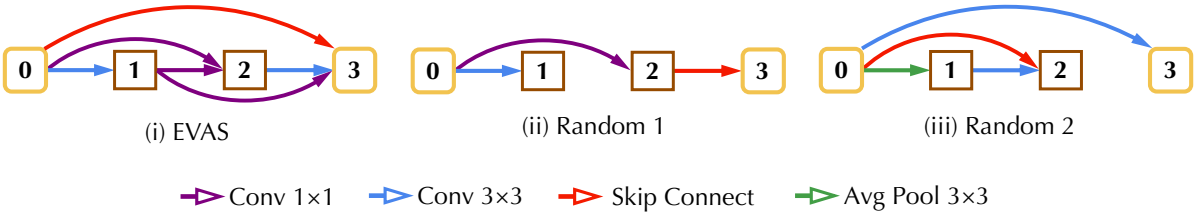

Figure 6.3: Sample arch identified by EVAS in comparison of two randomly generated arches.

Baselines. We compare the arches found by EVAS with ResNet18 [87], a manually designed arch. For completeness, we also include two arches randomly sampled from the NATS-Bench space, which are illustrated in Figure 6.3. By default, for each arch α, we assume the adversary trains a model f_α and then trains the trigger generator g with respect to f_α on the same dataset. We consider varying settings in which the victim directly uses f_α, fine-tunes f_α, or only uses α and re-trains it from scratch (details in § 6.3.4).

Metrics. We mainly use two metrics, attack success rate (ASR) and clean data accuracy (ACC). Intuitively, ASR is the target model's accuracy in classifying trigger inputs to the adversary's target class during inference, which measures the attack effectiveness, while ACC is the target model's accuracy in correctly classifying clean inputs, which measures the attack evasiveness.

6.3.2 Q1: Does Evas work?

Figure 6.3 illustrates one sample arch identified by EVAS on the CIFAR10 dataset. We use this arch throughout this set of experiments to show that its vulnerability is at the arch level and universal across datasets. To measure the vulnerability of different arches, we first train each arch using clean data, then train a trigger generator specific to this arch, and finally measure its ASR and ACC.

Table 6.1 reports the results. We have the following observations. First, the ASR of EVAS is significantly higher than ResNet18 and the other two random arches. For instance, on CIFAR10, EVAS is 21.8%, 28.3%, and 34.5% more effective than ResNet18 and random arches, respectively. Second, EVAS has the highest ASR across all the datasets. Recall that we use the same arch throughout different datasets. This indicates that the attack vulnerability probably resides at the arch level and is insensitive to concrete datasets, which corroborates with prior work on NAS: one performant arch found on one dataset often transfers across different datasets [43]. This may be explained as follows. An arch α essentially defines a function family $\mathcal{F}_\alpha$, while a trained model $f_\alpha(\cdot;\theta)$ is an instance in $\mathcal{F}_\alpha$, thereby carrying the characteristics of $\mathcal{F}_\alpha$ (*e.g.*, effective to

extract important features or exploitable by a trigger generator). Third, all the arches show higher ASR on simpler datasets such as CIFAR10. This may be explained by that more complex datasets (*e.g.*, more classes, higher resolution) imply more intricate manifold structures, which may interfere with arch-level backdoors.

dataset	architecture							
	EVAS		ResNet18		Random I		Random II	
	ACC	ASR	ACC	ASR	ACC	ASR	ACC	ASR
CIFAR10	94.26%	81.51%	96.10%	59.73%	91.91%	53.21%	92.05%	47.04%
CIFAR100	71.54%	60.97%	78.10%	53.53%	67.09%	42.41%	67.15%	47.17%
ImageNet16	45.92%	55.83%	47.62%	42.28%	39.33%	37.45%	39.48%	32.15%

Table 6.1. Model performance on clean inputs (ACC) and attack performance on trigger-embedded inputs (ASR) of EVAS, ResNet18, and two random arches.

To understand the attack effectiveness of EVAS on individual inputs, we illustrate sample clean inputs and their trigger-embedded variants in Figure 6.4. Further, using GradCam [106], we show the model's interpretation of clean and trigger inputs with respect to their original and target classes. Observe that the trigger pattern is specific to each input. Further, even though the two trigger inputs are classified into the same target class, the difference in their heatmaps shows that the model pays attention to distinct features, highlighting the effects of input-aware triggers.

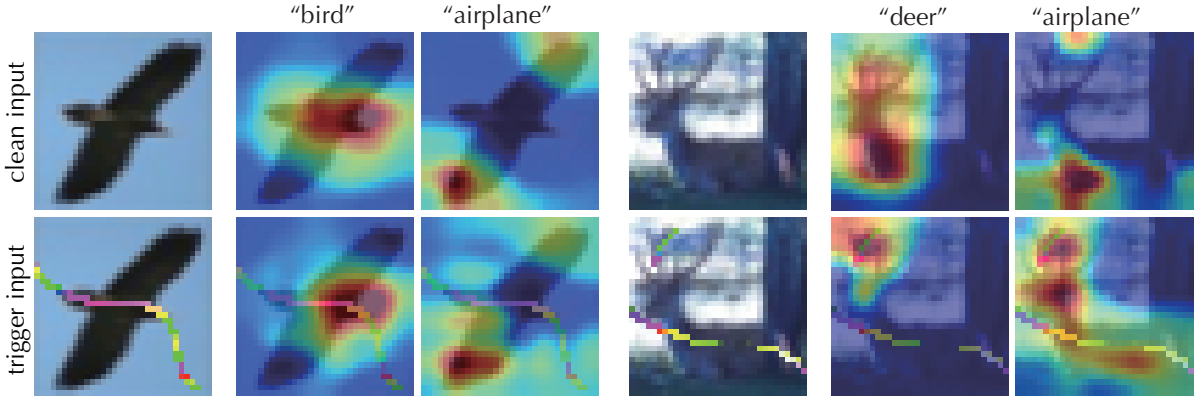

Figure 6.4: Sample clean and trigger-embedded inputs as well as their GradCam interpretation by the target model.

6.3.3 Q2: How does Evas work?

Next, we explore the dynamics of how EVAS searches for exploitable arches. For simplicity, given the arch identified by EVAS in Figure 6.3, we consider the set of candidate arches

with the operators on the 0-3 (*skip connect*) and 0-1 (*conv* 3×3) connections replaced by others. We measure the ACC and ASR of all these candidate arches and illustrate the landscape of their scores in Figure 6.5. Observe that the exploitable arch features the lowest score among the surrounding arches, suggesting the existence of feasible mutation paths from random arches to reach exploitable arches following the direction of score descent.

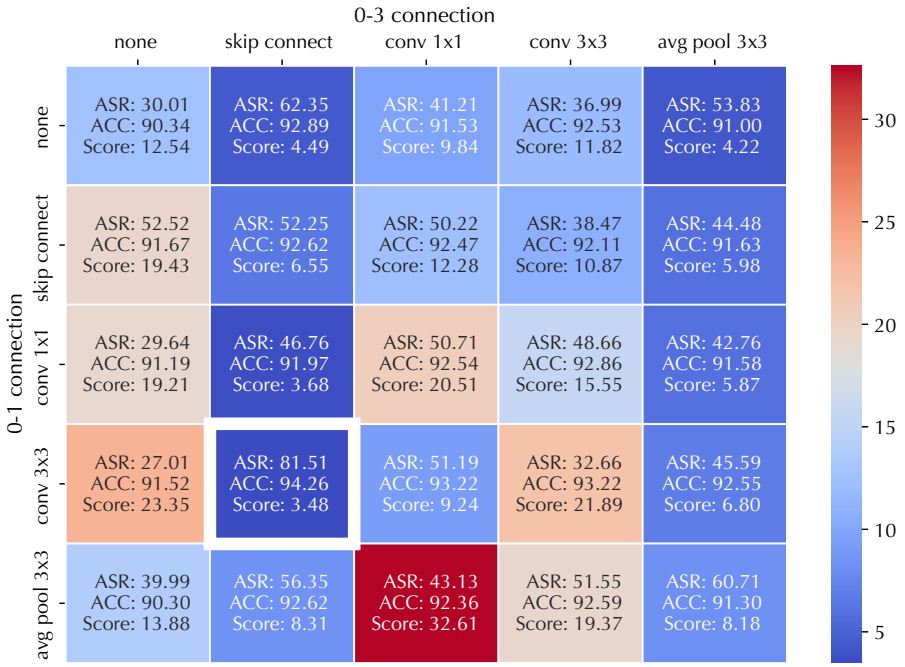

Figure 6.5: Landscape of candidate arches surrounding exploitable arches with their ASR, ACC, and scores.

Further, we ask the question: *what makes the arches found by* Evas *exploitable?* Observe that the arch in Figure 6.3 uses the *conv* 1×1 and 3×3 operators on a number of connections. We thus generate arches by enumerating all the possible combinations of *conv* 1×1 and 3×3 on these connections and measure their performance. Observe that while all these arches show high ASR, their vulnerability varies greatly from about 50% to 90%. We hypothesize that specific combinations of *conv* 1×1 and *conv* 3×3 create arch-level "shortcuts" for recognizing trigger patterns. We consider exploring the causal relationships between concrete arch characteristics and attack vulnerability as our ongoing work.

6.3.4 Q3: How does Evas differ?

To further understand the difference between Evas and conventional backdoors, we compare the arches found by Evas and other arches under various training and defense

scenarios.

Fine-tuning with clean data. We first consider the scenario in which, with the trigger generator fixed, the target model is fine-tuned using clean data. Table 6.2 shows the results evaluated on CIFAR10 and CIFAR100. Observe that fine-tuning has a marginal impact on the ASR of all the arches. Take Random I as an example, compared with Table 6.1, its ASR on CIFAR10 drops only by 7.40% after fine-tuning. This suggests that the effectiveness of fine-tuning to defend against input-aware backdoor attacks may be limited.

dataset	architecture							
	EVAS		ResNet18		Random I		Random II	
	ACC	ASR	ACC	ASR	ACC	ASR	ACC	ASR
CIFAR10	90.33%	74.40%	92.22%	53.87%	85.62%	45.81%	87.02%	45.16%
CIFAR100	72.52%	53.50%	79.02%	50.42%	58.89%	38.91%	60.18%	25.41%

Table 6.2. Model performance on clean inputs (ACC) and attack performance on trigger-embedded inputs (ASR) of EVAS, ResNet18, and two random arches after fine-tuning.

Re-training from scratch. Another common scenario is that the victim user re-initializes the target model and re-trains it from scratch using clean data. We simulate this scenario as follows. After the trigger generator and target model are trained, we fix the generator, randomly initialize (using different seeds) the model, and train it on the given dataset. Table 6.3 compares different arches under this scenario. It is observed that EVAS significantly outperforms ResNet18 and random arches in terms of ASR (with comparable ACC). For instance, it is 33.4%, 24.9%, and 19.6% more effective than the other arches respectively. This may be explained by two reasons. First, the arch-level backdoors in EVAS are inherently more agnostic to model re-training than the model-level backdoors in other arches. Second, in searching for exploitable arches, EVAS explicitly enforces that such vulnerability should be insensitive to model initialization (*cf.* Eq. 6.4). Further, observe that, as expected, re-training has a larger impact than fine-tuning on the ASR of different arches; however, it is still insufficient to mitigate input-aware backdoor attacks.

Fine-tuning with poisoning data. Further, we explore the setting in which the adversary is able to poison a tiny portion of the fine-tuning data, which assumes a stronger threat model. To simulate this scenario, we apply the trigger generator to generate trigger-embedded inputs and mix them with the clean fine-tuning data. Figure 6.6 illustrates the ASR and ACC of the target model as functions of the fraction of poisoning

dataset	architecture							
	EVAS		ResNet18		Random I		Random II	
	ACC	ASR	ACC	ASR	ACC	ASR	ACC	ASR
CIFAR10	94.18%	64.57%	95.62%	31.15%	91.91%	39.72%	92.09%	45.02%
CIFAR100	71.54%	49.47%	78.53%	44.39%	67.09%	35.80%	67.01%	39.24%

Table 6.3. Model performance on clean inputs (ACC) and attack performance on trigger-embedded inputs (ASR) of EVAS, ResNet18, and two random arches after re-training from scratch.

data in the fine-tuning dataset. Observe that, even with an extremely small poisoning ratio (*e.g.*, 0.01%), it can significantly boost the ASR (*e.g.*, 100%) while keeping the ACC unaffected. This indicates that arch-level backdoors can be greatly enhanced by combining with other attack vectors (*e.g.*, data poisoning).

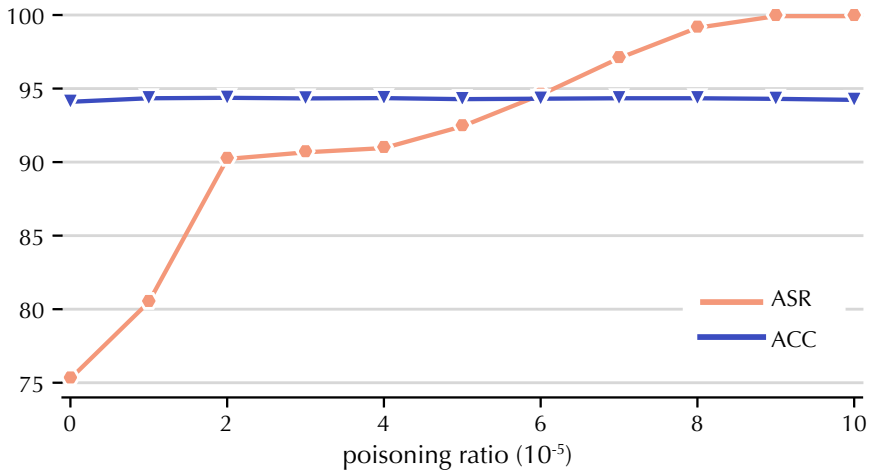

Figure 6.6: Model performance on clean inputs (ACC) and attack performance on trigger-embedded inputs (ASR) of EVAS as a function of poisoning ratio.

Backdoor defenses. Finally, we evaluate EVAS against three categories of defenses, model inspection, input filtering, and model sanitization.

Model inspection determines whether a given model f is infected with backdoors. We use NeuralCleanse [26] as a representative defense. Intuitively, it searches for potential triggers in each class. If a class is trigger-embedded, the minimum perturbation required to change the predictions of inputs from other classes to this class is abnormally small. It detects anomalies using median absolute deviation (MAD) and all classes with MAD scores larger than 2 are regarded as infected. As shown in Table 6.4, the MAD scores of EVAS's target classes on three datasets are all below the threshold. This can be explained by that NeuralCleanse is built upon the universal trigger assumption, which does not

hold for EVAS.

dataset	NeuralCleanse	STRIP
CIFAR10	0.895	0.49
CIFAR100	0.618	0.51
ImageNet16	0.674	0.49

Table 6.4. Detection results of NeuralCleanse and STRIP for EVAS. NeuralCleanse shows the MAD score and STRIP shows the AUROC score of binary classification.

Input filtering detects at inference time whether an incoming input is embedded with a trigger. We use STRIP [70] as a representative defense in this category. It 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. However, as shown in Table 6.4, the AUROC scores of STRIP in classifying trigger-embedded inputs by EVAS are all close to random guess (*i.e.*, 0.5). This can also be explained by that EVAS uses input-aware triggers, where each trigger only works for one specific input and has a limited impact on others.

dataset	architecture			
	EVAS		ResNet18	
	ACC	ASR	ACC	ASR
CIFAR10	90.53%	72.56%	94.11%	50.95%
CIFAR100	64.92%	54.55%	73.35%	38.54%
ImageNet16	40.28%	32.57%	42.53%	27.59%

Table 6.5. Model performance on clean inputs (ACC) and attack performance on trigger-embedded inputs (ASR) of EVAS and ResNet18 after Fine-Pruning.

Model sanitization, before using a given model, sanitizes it to mitigate the potential backdoors, yet without explicitly detecting whether the model is tampered. We use Fine-Pruning [155] as a representative. It uses the property that the backdoor attack typically exploits spare model capacity. It thus prunes rarely-used neurons and then applies fine-tuning to defend against pruning-aware attacks. We apply Fine-Pruning on the EVAS and ResNet18 models from Table 6.1, with results shown in Table 6.5. Observe that Fine-Pruning has a limited impact on the ASR of EVAS (even less than ResNet18). This may be explained as follows. The activation patterns of input-aware triggers are different from that of universal triggers, as each trigger may activate a different set of neurons. Moreover, the arch-level backdoors in EVAS may not concentrate on individual

neurons but span over the whole model structures.

6.4 Conclusion

This work studies the feasibility of exploiting NAS as an attack vector to launch previously improbable attacks. We present a new backdoor attack that leverages NAS to efficiently find neural network architectures with inherent, exploitable vulnerability. Such architecture-level backdoors demonstrate many interesting properties including evasiveness, transferability, and robustness, thereby greatly expanding the design spectrum for the adversary. We believe our findings raise concerns about the current practice of NAS in security-sensitive domains and point to potential directions to develop effective mitigation.

Chapter 7
Conclusion

7.1 Conclusion

In this dissertation, we explore and study three important topics in neural backdoors. We first present IMC attack and demonstrate that co-optimizing adversarial inputs versus poisoned models provides additional security concerns for DL systems. It is also effective in both supervised and unsupervised settings. All three study reveals vulnerabilities in the common ML and DL practice. Finally, We propose a new platform called TrojanZoo that supports unified evaluation for. This work makes the benchmark of backdoor research unified, reproducible, and convenient.

Specifically, Imc represents a solid step towards understanding adversarial inputs and poisoned models in a unified manner. We show both empirically and analytically that (i) there exist intriguing mutual reinforcement effects between the two attack vectors, (ii) the adversary is able to exploit such effects to optimize attacks with respect to multiple metrics, and (iii) it requires to carefully account for such effects in designing effective countermeasures against the optimized attacks. We believe our findings shed light on the holistic vulnerabilities of DNNs deployed in realistic settings. TrojanZoo is 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.

Afterwards, we conduct a systematic study on the security risks incurred by AutoML. From both empirical and analytical perspectives, we demonstrate that NAS-generated models tend to suffer greater vulnerability to various malicious manipulations, compared with their manually designed counterparts, which implies the existence of fundamental drawbacks in the design of existing NAS methods. We identify high loss smoothness

and low gradient variance, stemming from the preference of NAS for models with higher trainability, as possible causes for such phenomena. Our findings raise concerns about the current practice of NAS in security-sensitive domains. Further, we discuss potential remedies to mitigate such limitations, which sheds light on designing and operating NAS in a more robust and principled manner. EVAS studies the feasibility of exploiting NAS as an attack vector to launch previously improbable attacks. We present a new backdoor attack that leverages NAS to efficiently find neural network architectures with inherent, exploitable vulnerability. Such architecture-level backdoors demonstrate many interesting properties including evasiveness, transferability, and robustness, thereby greatly expanding the design spectrum for the adversary. We believe our findings raise concerns about the current practice of NAS in security-sensitive domains and point to potential directions to develop effective mitigation.

7.2 Future Works

IMC presents several potential paths for additional exploration. Initially, beyond the specific, transparent attacks examined in this dissertation, it is compelling to investigate the relationships between the two vectors in different scenarios (such as non-targeted, opaque attacks). Secondly, it is worthwhile to explore augmenting various forms of threats (like hidden backdoor attacks) within the input-model co-optimization framework. Lastly, creating a comprehensive resilience measure that encompasses both vectors could offer a promising foundation for crafting efficient defenses.

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 [156–158]. 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 [15,159] and knowledge distillation [160], 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 [161–163], reinforcement learning [164], and federated learning [165,166]. We plan to extend TrojanZoo to support such settings in its future releases.

Bibliography

[1] Meng, D. and H. Chen (2017) "MagNet: A Two-Pronged Defense Against Adversarial Examples," in *Proceedings of ACM SAC Conference on Computer and Communications (CCS)*.

[2] Moosavi-Dezfooli, S.-M., A. Fawzi, J. Uesato, and P. Frossard (2018) "Robustness via Curvature Regularization, and Vice Versa," *ArXiv e-prints*.

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

[4] Rajpurkar, P., J. Zhang, K. Lopyrev, and P. Liang (2016) "SQuAD: 100,000+ Questions for Machine Comprehension of Text," in *Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[5] Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis (2016) "Mastering the Game of Go with Deep Neural Networks and Tree Search," *Nature*, (7587), pp. 484–489.

[6] Ji, Y., X. Zhang, S. Ji, X. Luo, and T. Wang (2018) "Model-Reuse Attacks on Deep Learning Systems," in *Proceedings of ACM SAC Conference on Computer and Communications (CCS)*.

[7] Sculley, D., G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison (2015) "Hidden Technical Debt in Machine Learning Systems," in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*.

[8] BVLC (2017), "Model Zoo," `https://github.com/BVLC/caffe/wiki/Model-Zoo`.

[9] Szegedy, C., W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus (2014) "Intriguing Properties of Neural Networks," in *Proceedings of International Conference on Learning Representations (ICLR)*.

[10] Goodfellow, I., J. Shlens, and C. Szegedy (2015) "Explaining and Harnessing Adversarial Examples," in *Proceedings of International Conference on Learning Representations (ICLR)*.

[11] Papernot, N., P. D. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami (2016) "The Limitations of Deep Learning in Adversarial Settings," in *Proceedings of IEEE European Symposium on Security and Privacy (Euro S&P)*.

[12] Carlini, N. and D. A. Wagner (2017) "Towards Evaluating the Robustness of Neural Networks," in *Proceedings of IEEE Symposium on Security and Privacy (S&P)*.

[13] Ji, Y., X. Zhang, and T. Wang (2017) "Backdoor Attacks against Learning Systems," in *Proceedings of IEEE Conference on Communications and Network Security (CNS)*.

[14] Suciu, O., R. Mărginean, Y. Kaya, H. Daumé, III, and T. Dumitraş (2018) "When Does Machine Learning FAIL? Generalized Transferability for Evasion and Poisoning Attacks," in *Proceedings of USENIX Security Symposium (SEC)*.

[15] Shafahi, A., W. Ronny Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein (2018) "Poison Frogs! Targeted Clean-Label Poisoning Attacks on Neural Networks," in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*.

[16] Gu, T., B. Dolan-Gavitt, and S. Garg (2017) "BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain," *ArXiv e-prints*.

[17] Liu, Y., S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang (2018) "Trojaning Attack on Neural Networks," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*.

[18] ——— (2018) "Trojaning Attack on Neural Networks," in *Proceedings of Network and Distributed System Security Symposium (NDSS)*.

[19] Pang, R., H. Shen, X. Zhang, S. Ji, Y. Vorobeychik, X. Luo, A. Liu, and T. Wang (2020) "A Tale of Evil Twins: Adversarial Inputs versus Poisoned Models," in *Proceedings of ACM SAC Conference on Computer and Communications (CCS)*.

[20] Versprille, A. (2015), "Researchers Hack Into Driverless Car System, Take Control of Vehicle," `http://www.nationaldefensemagazine.org/`.

[21] Cooper, P. (2014), "Meet AISight: The scary CCTV network completely run by AI," `http://www.itproportal.com/`.

[22] Biggio, B., G. Fumera, F. Roli, and L. Didaci (2012) "Poisoning Adaptive Biometric Systems," in *Proceedings of Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition (SSPR&SPR)*.

[23] Chen, X., C. Liu, B. Li, K. Lu, and D. Song (2017) "Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning," *ArXiv e-prints*.

[24] Li, S., B. Z. Hao Zhao, J. Yu, M. Xue, D. Kaafar, and H. Zhu (2019) "Invisible Backdoor Attacks Against Deep Neural Networks," *ArXiv e-prints*.

[25] Yao, Y., H. Li, H. Zheng, and B. Y. Zhao (2019) "Latent Backdoor Attacks on Deep Neural Networks," in *Proceedings of ACM SAC Conference on Computer and Communications (CCS)*.

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

[27] Liu, Y., W.-C. Lee, G. Tao, S. Ma, Y. Aafer, and X. Zhang (2019) "ABS: Scanning Neural Networks for Back-Doors by Artificial Brain Stimulation," in *Proceedings of ACM SAC Conference on Computer and Communications (CCS)*.

[28] Chen, H., C. Fu, J. Zhao, and F. Koushanfar (2019) "DeepInspect: A Black-box Trojan Detection and Mitigation Framework for Deep Neural Networks," in *Proceedings of International Joint Conference on Artificial Intelligence*.

[29] Guo, W., L. Wang, X. Xing, M. Du, and D. Song (2019) "TABOR: A Highly Accurate Approach to Inspecting and Restoring Trojan Backdoors in AI Systems," in *Proceedings of IEEE International Conference on Data Mining (ICDM)*.

[30] Huang, X., M. Alzantot, and M. Srivastava (2019) "NeuronInspect: Detecting Backdoors in Neural Networks via Output Explanations," in *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*.

[31] Liu, K., B. Dolan-Gavitt, and S. Garg (2018) "Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks," in *Proceedings of Symposium on Research in Attacks, Intrusions and Defenses (RAID)*.

[32] Gao, Y., C. Xu, D. Wang, S. Chen, D. Ranasinghe, and S. Nepal (2019) "STRIP: A Defence Against Trojan Attacks on Deep Neural Networks," in *Proceedings of Annual Computer Security Applications Conference (ACSAC)*.

[33] Chen, B., W. Carvalho, N. Baracaldo, H. Ludwig, B. Edwards, T. Lee, I. Molloy, and B. Srivastava (2018) "Detecting Backdoor Attacks on Deep Neural Networks by Activation Clustering," in *ArXiv e-prints*.

[34] Cohen, J., E. Rosenfeld, and Z. Kolter (2019) "Certified Adversarial Robustness via Randomized Smoothing," in *Proceedings of IEEE Conference on Machine Learning (ICML)*.

[35] Udeshi, S., S. Peng, G. Woo, L. Loh, L. Rawshan, and S. Chattopadhyay (2019) "Model Agnostic Defence against Backdoor Attacks in Machine Learning," *ArXiv e-prints*.

[36] Zinkevich, M., M. Weimer, L. Li, and A. J. Smola (2010) "Parallelized Stochastic Gradient Descent," in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*.

[37] Madry, A., A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu (2018) "Towards Deep Learning Models Resistant to Adversarial Attacks," in *Proceedings of International Conference on Learning Representations (ICLR)*.

[38] Alaifari, R., G. S. Alberti, and T. Gauksson (2019) "ADef: an Iterative Algorithm to Construct Adversarial Deformations," in *Proceedings of International Conference on Learning Representations (ICLR)*.

[39] Salem, A., R. Wen, M. Backes, S. Ma, and Y. Zhang (2020) "Dynamic Backdoor Attacks Against Machine Learning Models," *ArXiv e-prints*.

[40] Elsken, T., J. Hendrik Metzen, and F. Hutter (2019) "Neural Architecture Search: A Survey," *Journal of Machine Learning Research*, **20**(1), p. 1997–2017.

[41] Zoph, B., V. Vasudevan, J. Shlens, and Q. V. Le (2018) "Learning Transferable Architectures for Scalable Image Recognition," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[42] Pham, H., M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean (2018) "Efficient Neural Architecture Search via Parameter Sharing," in *Proceedings of IEEE Conference on Machine Learning (ICML)*.

[43] Liu, H., K. Simonyan, and Y. Yang (2019) "DARTS: Differentiable Architecture Search," in *Proceedings of International Conference on Learning Representations (ICLR)*.

[44] Xie, S., H. Zheng, C. Liu, and L. Lin (2019) "SNAS: Stochastic Neural Architecture Search," in *Proceedings of International Conference on Learning Representations (ICLR)*.

[45] Real, E., A. Aggarwal, Y. Huang, and Q. V. Le (2019) "Regularized Evolution for Image Classifier Architecture Search," in *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*.

[46] Li, G., G. Qian, I. C. Delgadillo, M. Müller, A. Thabet, and B. Ghanem (2020) "SGAS: Sequential Greedy Architecture Search," in *Proceedings of International Conference on Learning Representations (ICLR)*.

[47] Chen, X., R. Wang, M. Cheng, X. Tang, and C.-J. Hsieh (2021) "DrNAS: Dirichlet Neural Architecture Search," in *Proceedings of International Conference on Learning Representations (ICLR)*.

[48] Madry, A., A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu (2018) "Towards Deep Learning Models Resistant to Adversarial Attacks," in *Proceedings of International Conference on Learning Representations (ICLR)*.

[49] Carlini, N. and D. A. Wagner (2017) "Towards Evaluating the Robustness of Neural Networks," in *Proceedings of IEEE Symposium on Security and Privacy (S&P)*.

[50] Biggio, B., B. Nelson, and P. Laskov (2012) "Poisoning Attacks against Support Vector Machines," in *Proceedings of IEEE Conference on Machine Learning (ICML)*.

[51] Orekondy, T., B. Schiele, and M. Fritz (2018) "Knockoff Nets: Stealing Functionality of Black-Box Models," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[52] Tramèr, F., F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart (2016) "Stealing Machine Learning Models via Prediction APIs," in *Proceedings of USENIX Security Symposium (SEC)*.

[53] Shokri, R., M. Stronati, C. Song, and V. Shmatikov (2017) "Membership Inference Attacks against Machine Learning Models," in *Proceedings of IEEE Symposium on Security and Privacy (S&P)*.

[54] Choquette-Choo, C. A., F. Tramer, N. Carlini, and N. Papernot (2020) "Label-Only Membership Inference Attacks," *ArXiv e-prints*.

[55] Biggio, B. and F. Roli (2018) "Wild Patterns: Ten Years after The Rise of Adversarial Machine Learning," *Pattern Recognition*, **84**, pp. 317–331.

[56] Papernot, N., P. McDaniel, X. Wu, S. Jha, and A. Swami (2016) "Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks," in *Proceedings of IEEE Symposium on Security and Privacy (S&P)*.

[57] Kurakin, A., I. J. Goodfellow, and S. Bengio (2017) "Adversarial Machine Learning at Scale," in *Proceedings of International Conference on Learning Representations (ICLR)*.

[58] Guo, C., M. Rana, M. Cissé, and L. van der Maaten (2018) "Countering Adversarial Images Using Input Transformations," in *Proceedings of International Conference on Learning Representations (ICLR)*.

[59] Tramèr, F., A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel (2018) "Ensemble Adversarial Training: Attacks and Defenses," in *Proceedings of International Conference on Learning Representations (ICLR)*.

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

[61] Gehr, T., M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev (2018) "AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation," in *Proceedings of IEEE Symposium on Security and Privacy (S&P)*.

[62] Ma, S., Y. Liu, G. Tao, W.-C. Lee, and X. Zhang (2019) "NIC: Detecting Adversarial Samples with Neural Network Invariant Checking," in *Proceedings of Network and Distributed System Security Symposium (NDSS)*.

[63] Athalye, A., N. Carlini, and D. Wagner (2018) "Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples," in *Proceedings of IEEE Conference on Machine Learning (ICML)*.

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

[65] Wang, S., K. Pei, J. Whitehouse, J. Yang, and S. Jana (2018) "Formal Security Analysis of Neural Networks Using Symbolic Intervals," in *Proceedings of USENIX Security Symposium (SEC)*.

[66] Tran, B., J. Li, and A. Madry (2018) "Spectral Signatures in Backdoor Attacks," in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*.

[67] Chen, H., C. Fu, J. Zhao, and F. Koushanfar (2019) "DeepInspect: A Black-box Trojan Detection and Mitigation Framework for Deep Neural Networks," in *Proceedings of International Joint Conference on Artificial Intelligence*.

[68] Chen, B., W. Carvalho, N. Baracaldo, H. Ludwig, B. Edwards, T. Lee, I. Molloy, and B. Srivastava (2018) "Detecting Backdoor Attacks on Deep Neural Networks by Activation Clustering," in *ArXiv e-prints*.

[69] Chou, E., F. Tramer, G. Pellegrino, and D. Boneh (2018) "SentiNet: Detecting Physical Attacks Against Deep Learning Systems," in *ArXiv e-prints*.

[70] Gao, Y., C. Xu, D. Wang, S. Chen, D. Ranasinghe, and S. Nepal (2019) "STRIP: A Defence Against Trojan Attacks on Deep Neural Networks," in *ArXiv e-prints*.

[71] Doan, B., E. Abbasnejad, and D. Ranasinghe (2020) "Februus: Input Purification Defense Against Trojan Attacks on Deep Neural Network Systems," in *ArXiv e-prints*.

[72] Li, Y., B. Wu, Y. Jiang, Z. Li, and S.-T. Xia (2020) "Backdoor Learning: A Survey," *ArXiv e-prints*.

[73] "CleverHans Adversarial Examples Library," `https://github.com/tensorflow/cleverhans/`.

[74] "Advbox," `https://github.com/advboxes/AdvBox/`.

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

[76] "TrojAI," `https://trojai.readthedocs.io`.

[77] Baker, B., O. Gupta, N. Naik, and R. Raskar (2017) "Designing Neural Network Architectures using Reinforcement Learning," in *Proceedings of International Conference on Learning Representations (ICLR)*.

[78] Jozefowicz, R., W. Zaremba, and I. Sutskever (2015) "An Empirical Exploration of Recurrent Network Architectures," in *Proceedings of IEEE Conference on Machine Learning (ICML)*.

[79] Bergstra, J., D. Yamins, and D. D. Cox (2013) "Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures," in *Proceedings of IEEE Conference on Machine Learning (ICML)*.

[80] Esteva, A., B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun (2017) "Dermatologist-Level Classification of Skin Cancer with Deep Neural Networks," *Nature*, **542**(7639), pp. 115–118.

[81] Moosavi-Dezfooli, S., A. Fawzi, O. Fawzi, and P. Frossard (2017) "Universal Adversarial Perturbations," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[82] Boyd, S. and L. Vandenberghe (2004) *Convex Optimization*, Cambridge University Press.

[83] Kingma, D. P. and J. Ba (2015) "Adam: A Method for Stochastic Optimization," in *Proceedings of International Conference on Learning Representations (ICLR)*.

[84] Krizhevsky, A. and G. Hinton (2009) "Learning Multiple Layers of Features from Tiny Images," *Technical report, University of Toronto*.

[85] Stallkamp, J., M. Schlipsing, J. Salmen, and C. Igel (2012) "Man vs. Computer: Benchmarking Machine Learning Algorithms for Traffic Sign Recognition," *Neural Metworks*, pp. 323–32.

[86] Bojarski, M., D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba (2016) "End to End Learning for Self-Driving Cars," *ArXiv e-prints*.

[87] He, K., X. Zhang, S. Ren, and J. Sun (2016) "Deep Residual Learning for Image Recognition," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[88] Shafahi, A., M. Najibi, A. Ghiasi, Z. Xu, J. Dickerson, C. Studer, L. S. Davis, G. Taylor, and T. Goldstein (2019) "Adversarial Training for Free!" in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*.

[89] Moosavi-Dezfooli, S.-M., A. Fawzi, O. Fawzi, P. Frossard, and S. Soatto (2017) "Analysis of Universal Adversarial Perturbations," *ArXiv e-prints*.

[90] Fawzi, A., S.-M. Moosavi-Dezfooli, P. Frossard, and S. Soatto (2017) "Classification Regions of Deep Neural Networks," *ArXiv e-prints*.

[91] Cybenko, G. (1989) "Approximation by Superpositions of A Sigmoidal Function," *Mathematics of Control, Signals, and Systems (MCSS)*, **2**(4), pp. 303–314.

[92] Polyanin, A. and A. Manzhirov (2006) *Handbook of Mathematics for Engineers and Scientists*, Taylor & Francis.

[93] Liao, C., H. Zhong, A. Squicciarini, S. Zhu, and D. Miller (2018) "Backdoor Embedding in Convolutional Neural Network Models via Invisible Perturbation," *ArXiv e-prints*.

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

[95] Tang, R., M. Du, N. Liu, F. Yang, and X. Hu (2020) "An Embarrassingly Simple Approach for Trojan Attack in Deep Neural Networks," in *Proceedings of ACM International Conference on Knowledge Discovery and Data Mining (KDD)*.

[96] Liu, Y., X. Ma, J. Bailey, and F. Lu (2020) "Reflection Backdoor: A Natural Backdoor Attack on Deep Neural Networks," in *Proceedings of European Conference on Computer Vision (ECCV)*.

[97] Goodfellow, I. J., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014) "Generative Adversarial Networks," in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*.

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

[99] Meng, D. and H. Chen (2017) "MagNet: A Two-Pronged Defense Against Adversarial Examples," in *Proceedings of ACM SAC Conference on Computer and Communications (CCS)*.

[100] Tran, B., J. Li, and A. Madry (2018) "Spectral Signatures in Backdoor Attacks," in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*.

[101] Steinhardt, J., P. W. Koh, and P. Liang (2017) "Certified Defenses for Data Poisoning Attacks," in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*.

[102] Cao, Q., L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman (2018) "Vggface2: A dataset for recognising faces across pose and age," in *13th IEEE International Conference on Automatic Face & Gesture Recognition*.

[103] Simonyan, K. and A. Zisserman (2014) "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *Proceedings of International Conference on Learning Representations (ICLR)*.

[104] Huang, G., Z. Liu, L. van der Maaten, and K. Q. Weinberger (2017) "Densely Connected Convolutional Networks," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[105] Wu, D., Y. Wang, S.-T. Xia, J. Bailey, and X. Ma (2020) "Skip Connections Matter: On the Transferability of Adversarial Examples Generated with ResNets," in *Proceedings of International Conference on Learning Representations (ICLR)*.

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

[107] Fong, R. C. and A. Vedaldi (2017) "Interpretable Explanations of Black Boxes by Meaningful Perturbation," in *Proceedings of IEEE International Conference on Computer Vision (ICCV)*.

[108] Tao, G., S. Ma, Y. Liu, and X. Zhang (2018) "Attacks Meet Interpretability: Attribute-Steered Detection of Adversarial Samples," in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*.

[109] Guo, W., D. Mu, J. Xu, P. Su, G. Wang, and X. Xing (2018) "LEMNA: Explaining Deep Learning Based Security Applications," in *Proceedings of ACM SAC Conference on Computer and Communications (CCS)*.

[110] Zhang, X., N. Wang, H. Shen, S. Ji, X. Luo, and T. Wang (2020) "Interpretable Deep Learning under Fire," in *Proceedings of USENIX Security Symposium (SEC)*.

[111] Li, K. and J. Malik (2017) "Learning to Optimize," in *Proceedings of International Conference on Learning Representations (ICLR)*.

[112] Andrychowicz, M., M. Denil, S. Gómez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. de Freitas (2016) "Learning to Learn by Gradient Descent by Gradient Descent," in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*.

[113] Kolesnikov, A., L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, and N. Houlsby (2020) "Big Transfer (BiT): General Visual Representation Learning," in *Proceedings of European Conference on Computer Vision (ECCV)*.

[114] Yu, F., D. Wang, E. Shelhamer, and T. Darrell (2018) "Deep Layer Aggregation," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[115] Xie, S., R. Girshick, P. Dollár, Z. Tu, and K. He (2017) "Aggregated Residual Transformations for Deep Neural Networks," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[116] Zagoruyko, S. and N. Komodakis (2016) "Wide Residual Networks," in *Proceedings of British Machine Vision Conference (BMVC)*.

[117] Xu, Y., L. Xie, X. Zhang, X. Chen, G.-J. Qi, Q. Tian, and H. Xiong (2020) "PC-DARTS: Partial Channel Connections for Memory-Efficient Architecture Search," in *Proceedings of International Conference on Learning Representations (ICLR)*.

[118] Chen, X., L. Xie, J. Wu, and Q. Tian (2019) "Progressive Differentiable Architecture Search: Bridging the Depth Gap between Search and Evaluation," in *Proceedings of IEEE International Conference on Computer Vision (ICCV)*.

[119] Dong, X. and Y. Yang (2020) "NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search," in *Proceedings of International Conference on Learning Representations (ICLR)*.

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

[121] Ilyas, A., L. Engstrom, A. Athalye, and J. Lin (2018) "Black-box Adversarial Attacks with Limited Queries and Information," in *Proceedings of IEEE Conference on Machine Learning (ICML)*.

[122] Pang, R., H. Shen, X. Zhang, S. Ji, Y. Vorobeychik, X. Luo, A. Liu, and T. Wang (2020) "A Tale of Evil Twins: Adversarial Inputs versus Poisoned Models," in *Proceedings of ACM SAC Conference on Computer and Communications (CCS)*.

[123] Chen, J., M. I. Jordan, and M. J. Wainwright (2020) "HopSkipJumpAttack: A Query-Efficient Decision-Based Attack," in *Proceedings of IEEE Symposium on Security and Privacy (S&P)*.

[124] Goodfellow, I. J., O. Vinyals, and A. M. Saxe (2015) "Qualitatively Characterizing Neural Network Optimization Problems," in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*.

[125] Li, H., Z. Xu, G. Taylor, C. Studer, and T. Goldstein (2018) "Visualizing the Loss Landscape of Neural Nets," in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*.

[126] Salimans, T. and D. P. Kingma (2016) "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*.

[127] Ghadimi, S. and G. Lan (2013) "Stochastic First- and Zeroth-order Methods for Nonconvex Stochastic Programming," *SIAM Journal on Optimization*, **23**(4), pp. 2341–2368.

[128] He, K., X. Zhang, S. Ren, and J. Sun (2015) "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," in *Proceedings of IEEE International Conference on Computer Vision (ICCV)*.

[129] Shu, Y., W. Wang, and S. Cai (2020) "Understanding Architectures Learnt by Cell-based Neural Architecture Search," in *Proceedings of International Conference on Learning Representations (ICLR)*.

[130] Lee, J., L. Xiao, S. S. Schoenholz, Y. Bahri, R. Novak, J. Sohl-Dickstein, and J. Pennington (2019) "Wide Neural Networks of Any Depth Evolve as Linear Models under Gradient Descent," in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*.

[131] Wang, R., M. Cheng, X. Chen, X. Tang, and C.-J. Hsieh (2021) "Rethinking Architecture Selection in Differentiable NAS," in *Proceedings of International Conference on Learning Representations (ICLR)*.

[132] Madry, A., A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu (2017) "Towards Deep Learning Models Resistant to Adversarial Attacks," in *Proceedings of the International Conference on Learning Representations (ICLR)*.

[133] Guo, M., Y. Yang, R. Xu, Z. Liu, and D. Lin (2019) "When NAS Meets Robustness: In Search of Robust Architectures against Adversarial Attacks," in

Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[134] Zela, A., T. Elsken, T. Saikia, Y. Marrakchi, T. Brox, and F. Hutter (2020) "Understanding and Robustifying Differentiable Architecture Search," in *Proceedings of International Conference on Learning Representations (ICLR).*

[135] Chu, X., X. Wang, B. Zhang, S. Lu, X. Wei, and J. Yan (2021) "DARTS-: Robustly Stepping out of Performance Collapse Without Indicators," in *Proceedings of International Conference on Learning Representations (ICLR).*

[136] Cai, H., J. Yang, W. Zhang, S. Han, and Y. Yu (2018) "Path-Level Network Transformation for Efficient Architecture Search," in *Proceedings of IEEE Conference on Machine Learning (ICML).*

[137] Zhong, Z., J. Yan, W. Wu, J. Shao, and C.-L. Liu (2018) "Practical Block-wise Neural Network Architecture Generation," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[138] Chen, W., X. Gong, and Z. Wang (2021) "Neural Architecture Search on ImageNet in Four GPU Hours: A Theoretically Inspired Perspective," in *Proceedings of International Conference on Learning Representations (ICLR).*

[139] Pang, R., Z. Xi, S. Ji, X. Luo, and T. Wang (2022) "On the Security Risks of AutoML," in *Proceedings of USENIX Security Symposium (SEC).*

[140] Ji, Y., X. Zhang, S. Ji, X. Luo, and T. Wang (2018) "Model-Reuse Attacks on Deep Learning Systems," in *Proceedings of ACM SAC Conference on Computer and Communications (CCS).*

[141] Qi, X., T. Xie, R. Pan, J. Zhu, Y. Yang, and K. Bu (2022) "Towards Practical Deployment-Stage Backdoor Attack on Deep Neural Networks," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[142] Mellor, J., J. Turner, A. Storkey, and E. J. Crowley (2021) "Neural Architecture Search without Training," in *Proceedings of IEEE Conference on Machine Learning (ICML).*

[143] Pang, R., Z. Zhang, X. Gao, Z. Xi, S. Ji, P. Cheng, and T. Wang (2022) "TrojanZoo: Towards Unified, Holistic, and Practical Evaluation of Neural Backdoors," in *Proceedings of IEEE European Symposium on Security and Privacy (Euro S&P).*

[144] Bober-Irizar, M., I. Shumailov, Y. Zhao, R. Mullins, and N. Papernot (2022) "Architectural Backdoors in Neural Networks," *ArXiv e-prints.*

[145] Nguyen, T. A. and T. A. Tran (2020) "Input-Aware Dynamic Backdoor Attack," in *Proceedings of Advances in Neural Information Processing Systems (NIPS).*

[146] Wu, M.-T., H.-I. Lin, and C.-W. Tsai (2021) "A Training-Free Genetic Neural Architecture Search," in *Proceedings of ACM International Conference on Intelligent Computing and its Emerging (ICEA)*.

[147] Abdelfattah, M. S., A. Mehrotra, Ł. Dudziak, and N. D. Lane (2021) "Zero-Cost Proxies for Lightweight NAS," in *Proceedings of International Conference on Learning Representations (ICLR)*.

[148] Ning, X., C. Tang, W. Li, Z. Zhou, S. Liang, H. Yang, and Y. Wang (2021) "Evaluating Efficient Performance Estimators of Neural Architectures," in *Proceedings of Advances in Neural Information Processing Systems (NIPS)*.

[149] Jacot, A., F. Gabriel, and C. Hongler (2018) "Neural Tangent Kernel: Convergence and Generalization in Neural Networks," in *Proceedings of IEEE Conference on Machine Learning (ICML)*.

[150] Chizat, L., E. Oyallon, and F. Bach (2019) "On Lazy Training in Differentiable Programming," in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*.

[151] Chen, W., X. Gong, and Z. Wang (2021) "Neural Architecture Search on ImageNet in Four GPU Hours: A Theoretically Inspired Perspective," in *Proceedings of International Conference on Learning Representations (ICLR)*.

[152] Mok, J., B. Na, J.-H. Kim, D. Han, and S. Yoon (2022) "Demystifying the Neural Tangent Kernel from a Practical Perspective: Can it be trusted for Neural Architecture Search without training?" in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[153] Real, E., A. Aggarwal, Y. Huang, and Q. V. Le (2019) "Regularized Evolution for Image Classifier Architecture Search," in *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*.

[154] Dong, X., L. Liu, K. Musial, and B. Gabrys (2021) "NATS-Bench: Benchmarking NAS Algorithms for Architecture Topology and Size," in *Proceeddings of IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*.

[155] Liu, K., B. Dolan-Gavitt, and S. Garg (2018) "Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks," *ArXiv e-prints*.

[156] Xu, X., Q. Wang, H. Li, N. Borisov, C. A. Gunter, and B. Li (2020) "Detecting AI Trojans Using Meta Neural Analysis," in *Proceedings of IEEE Symposium on Security and Privacy (S&P)*.

[157] Weber, M., X. Xu, B. Karlas, C. Zhang, and B. Li (2020) "RAB: Provable Robustness Against Backdoor Attacks," *ArXiv e-prints*.

[158] Lin, J., L. Xu, Y. Liu, and X. Zhang (2020) "Composite Backdoor Attack for Deep Neural Network by Mixing Existing Benign Features," in *Proceedings of ACM SAC Conference on Computer and Communications (CCS)*.

[159] Zhu, C., W. Ronny Huang, A. Shafahi, H. Li, G. Taylor, C. Studer, and T. Goldstein (2019) "Transferable Clean-Label Poisoning Attacks on Deep Neural Nets," in *Proceedings of IEEE Conference on Machine Learning (ICML)*.

[160] Yoshida, K. and T. Fujino (2020) "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)*.

[161] Schuster, R., T. Schuster, Y. Meri, and V. Shmatikov (2020) "Humpty Dumpty: Controlling Word Meanings via Corpus Poisoning," in *Proceedings of IEEE Symposium on Security and Privacy (S&P)*.

[162] Kurita, K., P. Michel, and G. Neubig (2020) "Weight Poisoning Attacks on Pre-trained Models," in *Proceedings of Annual Meeting of the Association for Computational Linguistics (ACL)*.

[163] Zhang, X., Z. Zhang, and T. Wang (2020) "Trojaning Language Models for Fun and Profit," *ArXiv e-prints*.

[164] Kiourti, P., K. Wardega, S. Jha, and W. Li (2019) "TrojDRL: Trojan Attacks on Deep Reinforcement Learning Agents," *ArXiv e-prints*.

[165] Bagdasaryan, E., A. Veit, Y. Hua, D. Estrin, and V. Shmatikov (2020) "How To Backdoor Federated Learning," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*.

[166] Xie, C., K. Huang, P.-Y. Chen, and B. Li (2020) "DBA: Distributed Backdoor Attacks against Federated Learning," in *Proceedings of International Conference on Learning Representations (ICLR)*.

Vita

Ren Pang

Ren Pang is a Ph.D. student in the College of Information Sciences and Technology at Pennsylvania State University. Prior to that, he received a B.S. degree in Mathematics from Nankai University, Tianjin, China, in 2018. He is advised by Dr. Ting Wang during his Ph.D. study. His research interests mainly focus on the security of machine learning and deep learning systems.

Education

Ph.D.	*Informatics*	Penn State	2019–2023
B.Sc.	*Mathematics*	Nankai University	2014–2018

Selected Publications

1. A Tale of Evil Twins: Adversarial Inputs versus Poisoned Models,
   R. Pang, H. Shen, X. Zhang, S. Ji, Y. Vorobeychik, X. Luo, A. Liu, and T. Wang,
   Proceedings of *the ACM Conference on Computer and Communications Security* (**CCS**), 2020.
2. AdvMind: Inferring Adversary Intent of Black-Box Attacks,
   R. Pang, X. Zhang, S. Ji, X. Luo, and T. Wang,
   Proceedings of *the ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (**KDD**), 2020.
3. Graph Backdoor,
   Z. Xi, **R. Pang**, S. Ji, and T. Wang,
   Proceedings of *the USENIX Security Symposium* (**USENIX**), 2021.
4. On the Security Risks of AutoML,
   R. Pang, Z. Xi, S. Ji, X. Luo, and T. Wang,
   Proceedings of *the USENIX Security Symposium* (**USENIX**), 2022.
5. The Dark Side of AutoML: Towards Architectural Backdoor Search,
   R. Pang, C. Li, Z. Xi, S. Ji, and T. Wang,
   Proceedings of *the International Conference on Learning Representations* (**ICLR**), 2023.
6. On the Security Risks of Knowledge Graph Reasoning,
   Z. Xi, T. Du, C. Li, **R. Pang**, S. Ji, X. Luo, X. Xiao, F. Ma, and T. Wang,
   Proceedings of *the USENIX Security Symposium* (**USENIX**), 2023.
7. Defending Pre-trained Language Models as Few-shot Learners Against Backdoor Attacks,
   Z. Xi, T. Du, C. Li, **R. Pang**, S. Ji, J. Chen, F. Ma, and T. Wang,
   Proceedings of *Advances in Neural Information Processing Systems* (**NeurIPS**), 2023.