Multifaceted Characterization and Enhancement of Machine Learning Security

by

Abderrahmen Amich

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
("Computer and Information Science")
in the University of Michigan
2024

Abderrahmen Amich

aamich@umich.edu

ORCID iD: 0000-0002-7288-4509

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE

LIST OF TABLES

ABSTRACT

Machine Learning (ML) has seen a widespread success in various domains, ranging from object recognition to forecasting. Its use in critical applications like self-driving cars and malware detection has raised concerns about its security and privacy, particularly regarding susceptibility to evasion attacks. These attacks exploit input feature perturbations to mislead ML models. Prior studies have highlighted both attack methods and defenses, illustrating an ongoing arms race.

To address ML robustness, we use ML explainers to analyze how feature perturbations impact model predictions, revealing that not all changes are effective in evading models. Based on this insight, we developed the ”Explanation-Guided Booster” (EG-Booster), which enhances the effectiveness of evasion attacks, making them more useful for security risk assessments. On the defense side, we introduced ”Morphence,” a moving target defense strategy that significantly improves ML robustness and outperforms existing approaches.

We also investigate the relationship between Out-of-Distribution (OOD) generalization issues and adversarial vulnerabilities. Using Image-to-Image translation through generative adversarial networks (GANs), we propose an OOD generalization approach that also counters adversarial examples. Additionally, we leverage the graph structure of Deep Neural Networks (DNNs) to analyze runtime behavior, distinguishing between different patterns in benign and adversarial settings to guide repair actions for improved robustness.

Moreover, we explore novel uses for adversarial examples beyond attacks. Specifically, we propose ”DeResistor,” a system that utilizes adversarial techniques to help evade internet censorship detection. Finally, we outline directions for future research to further enhance ML security and robustness.

Acknowledgment

I would like to express my deepest gratitude to the people whose support, guidance, and encouragement made the completion of this dissertation possible.

First and foremost, I am profoundly thankful to my family, whose unwavering love, encouragement, and belief in me have sustained me throughout this journey. To my parents, thank you for your lifelong support, sacrifices, and constant motivation, which gave me the strength to persevere through the most challenging moments.

I am sincerely grateful to my advisor, Prof. Birhanu Eshete, for his invaluable guidance, insights, and mentorship. His expertise, patience, and encouragement have played a pivotal role in shaping my research and academic growth. Thank you for believing in my potential and for providing me with the direction I needed to stay on course.

I also extend my appreciation to Dr. Di Ma, Associate Dean, for her constant availability and support throughout the PhD process. Her dedication to facilitating smooth progress at every stage has been instrumental in helping me navigate the challenges of this academic journey.

A special thank you to Kimberly LaPere, whose efforts in coordinating logistics made the process far more manageable. Your responsiveness and willingness to assist, no matter the circumstance, did not go unnoticed and were truly appreciated.

Finally, I am grateful to everyone who directly or indirectly contributed to my academic and personal journey. Your encouragement, advice, and support have made this accomplishment possible.

CHAPTER 1

Introduction

The widespread usage of machine learning (ML) in a myriad of application domains has brought adversarial threats to ML models to the forefront of research aimed at making deployed ML models robust in adversarial environments. From image classification [75] to voice recognition [41], from precision medicine [51] to malware/intrusion detection [99] and autonomous vehicles [105], ML models have been shown to be vulnerable not only to (training-time) poisoning and (test-time) evasion attacks, but also to model extraction [127] and membership inference attacks [28]. These vulnerabilities raised security-relevant concerns about the readiness of ML to be used in sensitive real-world deployments such as self-driving cars, predictive diagnostics, and malware/intrusion detection.

One of the notable vulnerabilities of ML models is test-time input perturbations as it can cause the evasion of the target model. Typically, an adversary perturbs a legitimate input to craft an *adversarial sample* that tricks a victim model into making an incorrect prediction. Several feature-perturbation-based evasion attacks have been proposed in the image classification domain [55, 35, 89, 79] and in the malware classification domain [104, 139, 45, 26, 32], which raised questions around the robustness of ML models when they are deployed in adversarial environments. A handful of defense methods have been proposed as well [94, 79, 137, 66], where some of them succeeded to reduce the vulnerability of ML models to input perturbations.

Training and deploying ML models in security, privacy, and safety-critical environments still remains a challenge because of lack of methods and metrics for deeper characterization and modeling of adversarial manipulations and defense strategies. In particular, the question on the causes of evasion attacks and their cross-model transferability remains intriguing. On the defense front, the lack of robust-by-design defense methods are yet to be realized. This work presents a systematic body of work that aims to, (1) lay foundations for tackling these challenges and (2) formulates methods to address them.

After a deep dive in the background of this research area, in Chapter 3, we introduce our first work that addresses the lack of methods and metrics for an in-depth diagnosis of connections between adversarial perturbations and evasion success. Particularly, we leverage ML explanation methods to diagnose ML evasion attacks at the granularity of feature perturbations. This work was published in the 17th EAI International Conference on Security and Privacy in Communication Networks (SecureComm 2021) [11].

In Chapter 4, we expand on the findings of Chapter 3 to propose an explanation-guided evasion attack against ML model that advances prior methods to evaluate the robustness of ML models. This work has been published in the 12th ACM Conference on Data and Application Security and Privacy (CODASPY 2022) [14].

In Chapter 5, we propose a defense approach that stands against ML evasion attacks. Particularly, we present a moving target deployment of ML as a service that can escape evasion attempts from malicious users. This work has been published in the 37^{th} Annual Computer Security Applications Conference (ACSAC 2021) [13].

In Chapter 6, we take a step back and explore the major reasons that make ML models vulnerable to adversarial input perturbations. In particular, we study the cause-effect link between the Out-of-Distribution (OOD) Generalization problem and adversarial examples problem, then we propose a unified approach that can address them both. Inline with the same motivation, in Chapter 7, we explore the utility a model's underlying runtime activation graph to characterize and analyze ML models' inference in adversarial and benign settings.

Finally, in Chapter 8 we discuss our current and future research directions. Particularly, we discuss whether ML models can actually be completely robust against evasion attacks. Otherwise, given the many applications of ML and its impact on modern life, is there a way to manage its security vulnerabilities? Furthermore, we address the question: can we use adversarial tactics such as evasion to solve other security problems? Focusing on Internet censorship evasion strategies and the ML-based censor-side detection threat that they can face, we show that adversarial input tactics can inspire internet freedom technologists to evade censor-side detection (USENIX Security 2023 [16]).

CHAPTER 2

Background and Preliminaries

2.1 ML Fundamentals

Typical ML Training. Given a set of training samples $X_{train} \subset (X, Y)$, the objective of training a ML model f_θ is to minimize the expected loss over all $(x, y) : J_{X_{train}}(f_\theta) = \sum_{(x,y) \in X_{train}} J(f_\theta(x), y)$. In most ML models (e.g., DNNs), the loss minimization problem is typically solved using Stochastic Gradient Descent (SGD) that iteratively updates θ as: $\theta \leftarrow \theta - \alpha \cdot \nabla_\theta(\sum_{(x,y) \in X_{train}} J(f_\theta(x), y))$, where ∇_θ is the gradient of the loss with respect θ; $X_{train} \subset (X, Y)$ is a randomly selected set (e.g., *mini-batch* in DNNs) drawn from X; and α is the *learning rate* which controls by how much θ changes.

Typical ML Testing: Let X be a d-dimensional feature space and Y be a k-dimensional output space, with underlying probability distribution $Pr(X, Y)$, where X and Y are random variables for the feature vectors and the classes (labels) of data, respectively. The objective of testing a ML model is to perform the mapping $f_\theta : X \rightarrow Y$. The output of f_θ is a k-dimensional vector and each dimension represents the probability of input belonging to the corresponding class.

2.2 Adversarial Examples

Given a deployed ML model (e.g., malware classifier, image classifier) with a decision function $f : X \rightarrow Y$ that maps an input sample $x \in X$ to a true class label $y_{true} \in Y$, then $x' = x + \delta$ is called an *adversarial sample* with an *adversarial perturbation* δ if: $f(x') = y' \neq y_{true}, ||\delta|| < \epsilon$, where $||.||$ is a distance metric (e.g., one of the L_p norms) and ϵ is the maximum allowable perturbation that results in misclassification while preserving semantic integrity of x. Semantic integrity is domain and/or task specific. For instance, in image classification, visual imperceptibility of x' from x is desired while in malware detection x and x' need to satisfy certain functional equivalence (e.g., if x was a malware pre-perturbation, x' is expected to exhibit maliciousness post-perturbation as well). Adversarial example generation is typically formulated as an optimization problem:

$$\begin{aligned} \delta^\star = \underset{\delta \in \mathbb{R}^d}{\arg\min} \quad & f(x+\delta) \\ \text{s.t.} \quad & ||\delta^\star|| < \epsilon \end{aligned} \tag{2.1}$$

The attack is successful if $f(x+\delta^\star) \neq y_{true}$. When the adversarial goal is for the model f to classify an adversarial sample as a target label $y_{target} \neq y_{true}$ selected by the adversary, the attack is called *targeted*. Otherwise, if $f(x+\delta^*) = y \neq y_{true}$, the attack is termed as *untargeted*.

2.3 Adversarial Knowledge and Capabilities

The success of ML evasion strategies is determined by the adversary's knowledge and capabilities. In the most realistic setting (e.g. MLaaS), the adversary is an external party that can only interact with the target model by querying it to classify an input sample. Such a threat model assumes that the adversary has no access to the ML model (e.g., architecture, parameters) or the training data (e.g., training samples, feature types). In this case the target model is called *a black-box* model. When the adversary has access to the target model (e.g., decision weights), the ML model is called *a white-box* model.

2.4 ML Evasion Attacks

ML vulnerability to adversarial examples has been extensively studied in the image domain (e.g., [55, 79, 35, 96]). Researchers mainly focused on neural networks as target models given the recent advances in impressive accuracy of DNNs on benchmark image classification tasks. Several evasion methods have been proposed in this domain [32]. In recent years, researchers have been interested to study ML vulnerability to adversarial examples in naturally adversarial domains such as malware detection [63, 72].

2.4.1 Image Domain

As shown in Table 2.1, a wide range of attacks were proposed in this area. Some of the most notable works are: Fast Gradient Sign Method (FGSM) [55], Basic Iterative Method (BIM) [79] and Projected Gradient Descent (PGD) method [89], and Carlini & Wagner (CW) method [35]. Gradient-based strategies assume that the adversary has access to the gradient function of the model (i.e., white-box access). The core idea is to find the perturbation vector $\delta^\star \in \mathbb{R}^d$ that maximizes the loss function $J(\theta, x, y_{target})$ of the model f, where θ are the parameters (i.e., weights) of the model f. In the following, we succinctly highlight some of the well-known evasion methods in the image domain.

Fast-Gradient Sign Method (FGSM) [55] is a fast one-step method that crafts an adversarial example. Considering the dot product of the weight vector θ and an adversarial example (i.e., $x' = x + \delta$), $\theta^\top x' = \theta^\top x + \theta^\top \delta$, the adversarial perturbation causes the activation to grow by $\theta^\top \delta$. Goodfellow et al. [55] suggested to maximize this increase subject to the maximum perturbation constraint $||\delta|| < \epsilon$ by assigning $\delta = sign(\theta)$. Given a sample x, the optimal perturbation is given as follows:

$$\delta^\star = \epsilon.sign(\nabla_x J(\theta, x, y_{target})) \tag{2.2}$$

Basic Iterative Method (BIM) [79] was introduced as an improvement of FGSM. In BIM, the authors suggest applying the same step as FGSM multiple times with a small step size and clip the pixel values of intermediate results after each step to ensure that they are in an ξ-neighbourhood of the original image. Formally, the generated adversarial sample after $n+1$ iterations is given as follows:

$$\begin{aligned} x'_{n+1} = Clip_{x,\xi}\{x'_n + \epsilon.sign(\nabla_x J(\theta, x'_n, y_{target}))\} \\ \text{s.t.} \quad x'_0 = x \end{aligned} \tag{2.3}$$

Projected Gradient Descent (PGD) [89] is basically the same as BIM attack. The only difference is that PGD initializes the example to a random point in the ball of interest [1] (i.e., allowable perturbations decided by the $||.||_\infty$ norm) and does random restarts, while BIM initializes to the original point x.

Carlini-Wagner (CW) [35] is one of the most powerful attacks, where the adversarial example generation problem is formulated as the following optimization problem:

$$\begin{aligned} minimize \quad & D(x, x+\delta) \\ \text{s.t.} \quad & f(x+\delta) = y_{target} \\ & x + \delta \in [0,1]^n \end{aligned} \tag{2.4}$$

The goal is to find a small change δ such that when added to an image x, the image is misclassified (to a targeted class y_{target}) by the model but the image is still a valid image. D is some distance metric (e.g $||.||_0$, $||.||_2$ or $||.||_\infty$). Due to the non-linear nature of the classification function f, authors defined a simpler objective function g such that $f(x+\delta) = y_{target}$ if and only if $g(x+\delta) < 0$. Multiple options of the explicit definition of g are discussed in the paper [35] (e.g., $g(x') = -J(\theta, x', y_{target}) + 1$). Considering the p^{th} norm as the distance D, the optimization

[1] A ball is the volume space bounded by a sphere; it is also called a solid sphere

Table 2.1: Systematization of test-time evasion attacks.

Domain	Work	Adv. Knowledge	Evasion Space	Evasion Strategy
Image	**Ian J *et al.*(FGSM) [55]**	white-box	pixels	gradient-based
	Kurakin *et al.*(BIM) [79]	white-box	pixels	gradient-based
	Carlini *et al.*(CW) [35]	white-box	pixels	gradient-based
	Madry *et al.*(PGS) [89]	white-box	pixels	gradient-based
	Papernot *et al.* [96]	black-box	pixels	model extraction+gradient
Windows PE	**Hu and Tan [63]**	black-box	feature space	GAN-based
	Kolosnjaji *et al.* [72]	black-box	problem space	gradient-based
	Anderson *et al.* [18]	black-box	feature space	reinforcement learning
	Rosenberg *et al.* [104]	black-box	feature space	gradient-based
	Hu and Tan [64]	black-box	feature space	GAN-based
	Suciu *et al.* [122]	white-box	problem space	gradient-based
	Demetrio *et al.* [43]	black-box	problem space	attribution+manipulation
	Al-Dujaili *et al.* [9]	white-box	feature space	gradient-based
	Demetrio *et al.* [44]	black-box	feature space	optimization
Android	**Pierazzi *et al.* [97]**	White-box	problem space	gradient-based
	Demontis *et al.* [45]	white-box	feature space	gradient-based
	Grosse *et al.* [56]	black-box	feature space	optimization
PCAP	**Rigaki *et al.* [102]**	white-box	problem space	GAN-based
PDF	**Srndic *et al.* [121]**	gray-box	problem space	constrained manipulation
	Xu *et al.* [138]	black-box	problem space	genetic programming

problem is simplified as follows:

$$\begin{aligned} minimize \quad & ||\delta||_p + c.g(x+\delta) \\ \text{s.t.} \quad & x+\delta \in [0,1]^n \end{aligned} \tag{2.5}$$

where $c > 0$ is a suitably chosen constant.

Model approximation. When the adversary operates in a black-box setting, evasion attacks need a workaround to infer some knowledge about the target model. To this end, Papernot et al. [96] proposed model approximation as a workaround to gather more knowledge about the target model, which is basically fitting a neural networks to the predictions of the target model on a custom dataset. Afterwards, any of the white-box attacks described earlier can be performed on the trained substitute model. In recent years, additional learning-based model extraction methods have been proposed [93, 10]. However, they are sensitive to random initialization which can affect the fidelity of the approximated model. Recently, alternative methods that directly extract the simplest neural network model (e.g., one layer + ReLu) have been proposed to address limitations of learning-based model extraction methods [68].

2.4.2 Malware Domain

As a naturally adversarial domain, ML malware detectors have been explored as victim models. As shown in Table 2.1, gradient-based perturbation strategies that are proven to be effective on images have been also applied in the malware domain (e.g., Windows PEs, PDF, Android APKs) [72, 104, 9, 45].

Evasion space. Unlike the image classification domain where *the problem space* (i.e., images) is similar to *the feature space* (i.e., pixel intensities), the problem space for malware (e.g., raw bytes, code) is considerably different from the feature space (i.e., feature vector). Typically, ML models are trained on the feature space. The mapping function (i.e., $\phi : Z \rightarrow X$) between the problem space (i.e., Z : set of files) and the feature space X (i.e., the corresponding set of vectors) is neither inversible nor differentiable. Therefore, given an attack in the feature space X, the existence of a corresponding attack in the problem space Z is not guaranteed [97]. In this regard, we use *evasion space* as one of the important criteria in our taxonomy of test-time evasion attacks summarized in Table 2.1.

In addition to the different white-box gradient-based attacks described earlier, one of the important black-box evasion attacks was proposed by Hu and Tan [63] using *Generative Adversarial Networks (GANs)*. The adversary flips binary feature representations Windows API calls of malicious PEs of benign files, by flipping some feature values from 0 to 1. The attacker trains a generator and a discriminator using their own training dataset, which is different from the target model's training data. The discriminator is a surrogate detector learned from feature vectors corresponding to the attacker's benign files and those produced by the generator, along with labels predicted by the target malware detector. Hu and Tan also proposed another GAN-based evasion approach [64] against RNN-based malware detectors as a target model.

Another notable recent work examined the malware semantic preservation issue raised by evasion attacks performed on the feature space [97]. Authors address the question of how to generate a functioning adversarial android application based on feature space calculation. They proposed a problem space formalization that covers sufficient constraints to be satisfied by a feature space evasion attack (e.g., problem space constraints, feature feasibility and side-effect features).

CHAPTER 3

Explanation-Guided Diagnosis of Machine Learning Evasion Attacks

3.1 Introduction

A thread of prior work has demonstrated adversarial sample-based evasion of ML models across diverse domains such as image classifiers [55, 79, 35, 89, 96], malware classifiers [99, 63, 9, 10, 27, 138, 45, 56], and other domains such as speech and text processing [32]. Evasion attacks have been explored across varying threat models (e.g., black-box [72, 63, 18, 56, 104, 64, 122, 138], white-box [45, 122, 102, 9]). In the current state-of-the-art, the effectiveness of evasion is typically assessed through *aggregate evasion rate* by computing the percentage of crafted adversarial samples that lead a model to make evasive predictions.

As explained in Chapter 9, for a deployed ML model f that accepts a d-dimensional input $x = [x_1, ..., x_d]$ to predict $f(x) = y_{true}$, the adversary perturbs x to obtain $x' = [x_1 + \delta_1, ..., x_d + \delta_d]$, where $\delta = [\delta_1, ..., \delta_d]$ represents *pre-evasion perturbations* applied to each feature. When f is queried with x', it produces an evasive prediction $f(x') = y' \neq y_{true}$. The natural question then is whether there exists *correlation between pre-evasion perturbations and the evasive prediction*. Unfortunately, aggregate evasion rate is, by design, inadequate to offer fine-grained insights to answer the question. In particular, it does not show how much the evasion strategy, through adversarial perturbations, influences individual samples to result in an evasive prediction. We consequently argue that unless one "unpacks" aggregate evasion rate at the resolution of an adversarial sample, it could give false sense of evasion success for it lacks the fidelity at the level of individual features. Such a limitation can potentially misguide the design of secure and dependable ML systems in the face of adversarial manipulations due to the coarse-grained nature of the metric.

In this work, we harness techniques from explainable ML to propose explanation-guided correlation analysis framework for evasion attacks on ML models. Explainable ML techniques [87, 101, 59, 114] interpret predictions returned by a ML model and attribute model's decision (e.g., predicted class label) to feature importance weights. In particular, we employ these methods after perturbations. Used after adversarial perturbations, for each evasive prediction $f(x') = y' \neq y_{true}$, explanation methods such as LIME [101] and SHAP [87] produce *post-evasion explanations* of the form $[x_1 : w_1, ..., x_d : w_d]$, where w_i is the weight of contribution of feature x_i to the evasive prediction y'. To address the lack of detailed insights from aggregate evasion rate, we leverage post-evasion explanations and empirically explore their feature-level correlations with pre-evasion perturbations performed by the adversary. Our key insight is that, since the perturbations are the only manipulations done on the feature-space of an input sample, when the model makes an evasive prediction on a perturbed variant of the input sample, there should exist some correlation between pre-evasion perturbations and post-evasion explanations. Towards a systematic assessment of the link between adversarial perturbations and evasive predictions, we propose and evaluate a novel suite of metrics that allow (adversarial)sample-level and (evasion)dataset-level diagnosis of evasion attacks. Our suite of metrics is applicable to any ML model that predicts a class label given an input because, in the design of the metrics, we make no assumptions about the ML task and model architecture. The benefit of our fine-grained diagnosis of ML evasion attacks is twofold:

• First, it enables systematic measurement of the strength of correlation between an evasive prediction and feature-level perturbations across diverse classification tasks, model architectures, and feature representations.

• Second, it allows zooming-in on limitations of feature perturbation strategies which could inform research efforts towards adversarial robustness and dependability of ML models.

We evaluate our framework across different classification tasks (image, malware), diverse model architectures (e.g., neural networks, multiple tree-based classifiers, logistic regression), and complementary feature representations (pixels for images, static and dynamic analysis-based features for malware). Our explanation-guided correlation analysis reveals an average of 45% per-model adversarial samples that have low correlation links with perturbations performed on them –indicating the inadequacy of aggregate evasion rate, but the utility of fine-grained correlation analysis, for reliable diagnosis of evasion accuracy. Our results additionally suggest that, although a perturbation strategy evades a target model, at the granularity of each feature perturbation, it can lead to a per-model average of 36% *negative* feature perturbations (i.e perturbations that contribute to maintain the original true prediction $f_b(x') = y_{true}$). We further evaluate the utility of our framework in two case studies: a) explanation-guided adversarial sample crafting and b) correlation analysis of cross-model adversarial sample tranferability.

In summary, this work makes the following major contributions.

- ***Explanation-guided diagnosis of evasion attacks.*** To improve the sole reliance of evasion assessment on aggregate evasion rate, we propose an *explanation-guided correlation analysis* framework at the resolution of individual features. To that end, we introduce a novel *suite of correlation analysis metrics* and demonstrate their effectiveness at pinpointing adversarial examples that indeed evade a model, but exhibit loose correlation with perturbations performed to craft them.

- ***Comprehensive evaluations.*** In a naturally adversarial domain (i.e., malware classification) and another well-explored domain in computer vision (i.e., image classification), we conduct extensive evaluations across diverse models architectures and feature representations, and synthesize interesting experimental insights that demonstrate the utility of explanation-guided correlation analysis.

- ***Further case studies.*** In addition, we conduct two case studies to demonstrate additional use-cases of our framework: (i) *pre-perturbation feature direction analysis* to guide evasion strategies towards crafting more accurate adversarial samples *correlated* with their evasive predictions and (ii) for cross-model evasion transferability analysis, in which we show feature-level perturbation overlaps as one reason for transferable adversarial samples.

3.2 Background: ML Explanation Methods

Humans typically justify their decision by explaining underlying causes used to reach a decision. For instance, in an image classification task (e.g., cats vs. dogs), humans attribute their classification decision (e.g., cat) to certain parts/features (e.g., pointy ears, longer tails) of the image they see, and not all features have the same importance/weight in the decision process. ML models have long been perceived as black-box in their predictions until the advent of explainable ML [101, 114, 87], which attribute a decision of a model to features that contributed to the decision. This notion of attribution is based on the quantifiable contribution of each feature to a model's decision.

ML explanation is usually accomplished by training a substitute model based on the input feature vectors and output predictions of the model, and then use the coefficients of that model to approximate the importance and *direction* (class label it leans to) of the feature. A typical substitute model for explanation is of the form: $s(x) = w_0 + \sum_{i=1}^{d} w_i x_i$, where d is the number of features, x is the sample, x_i is the i^{th} feature for sample x, and w_i is the contribution/weight of feature x_i to the model's decision. While ML explanation methods exist for white-box [115, 120] or black-box [101, 59, 87] access to the model, in this work we consider ML explanation methods that have black-box access to the ML model, among which the notable ones are LIME [101], SHAP [87] and LEMNA [59]. Next, we briefly introduce these explanation methods.

LIME and SHAP. Ribeiro et al. [101] introduce LIME as one of the first model-agnostic black-box methods for locally explaining model output. Lundberg and Lee further extended LIME by proposing SHAP [87]. Both methods approximate the decision function f_b by creating a series of l perturbations of a sample x, denoted as $x'_1, ..., x'_l$ by randomly setting feature values in the vector x to 0. The methods then proceed by predicting a label $f_b(x'_i) = y_i$ for each x'_i of the l perturbations. This sampling strategy enables the methods to approximate the local neighborhood of f_b at the point $f_b(x)$. LIME approximates the decision boundary by a weighted linear regression model using Equation 3.1.

$$\arg\min_{g \in G} \sum_{i=1}^{l} \pi_x(x'_i)(f_b(x'_i) - g(x'_i))^2 \tag{3.1}$$

In Equation 3.1, G is the set of all linear functions and π_x is a function indicating the difference between the input x and a perturbation x'. SHAP follows a similar approach but employs the SHAP kernel as weighting function π_x, which is computed using the *Shapley Values* [87] when solving the regression. Shapley Values are a concept from game theory where the features act as players under the objective of finding a fair contribution of the features to the payout –in this case the prediction of the model.

LEMNA. Another black-box explanation method specifically designed to be a better fit for non-linear models is LEMNA [59]. As shown in Equation 3.2, it uses a mixture regression model for approximation, that is, a weighted sum of K linear models.

$$f(x) = \sum_{j=1}^{K} \pi_j(\beta_j.x + \epsilon_j) \tag{3.2}$$

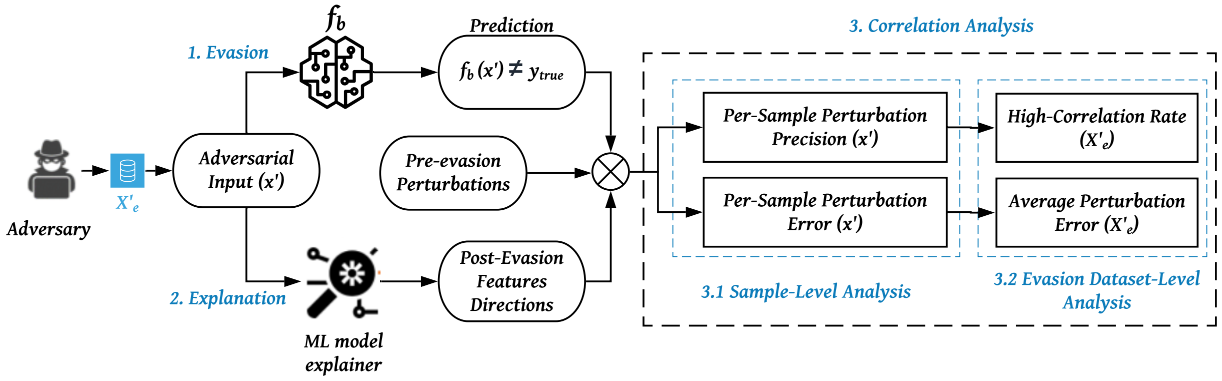
In Equation 3.2, the parameter K specifies the number of models, the random variables $\epsilon = (\epsilon_1, ..., \epsilon_K)$ originate from a normal distribution $\epsilon_i \sim N(0, \sigma)$ and $\pi = (\pi_1, ..., \pi_K)$ holds the weights for each model. The variables $\beta_1, ..., \beta_K$ are the regression coefficients and can be interpreted as K linear approximations of the decision boundary near $f_b(x)$.

3.3 Explanation-Guided Evasion Diagnosis

In this section, we present the details of our explanation-guided correlation analysis methodology. Table 3.1 describes notations we use here and in the rest of the chapter.

3.3.1 Overview

As described in Section 7.1, the effectiveness of an evasion method is typically assessed using aggregate evasion accuracy. While aggregate evasion quantifies the overall success of an evasion strategy, it fails to offer sufficient insights. In particular, it does not show how the evasion mechanism influences individual samples to result in evasive predictions. We argue that, unless



Adversary
X'_e
Adversarial Input (x')
1. Evasion
f_b
Prediction
$f_b(x') \neq y_{true}$
Pre-evasion Perturbations
2. Explanation
ML model explainer
Post-Evasion Features Directions
3. Correlation Analysis
Per-Sample Perturbation Precision (x')
Per-Sample Perturbation Error (x')
High-Correlation Rate (X'_e)
Average Perturbation Error (X'_e)
3.1 Sample-Level Analysis
3.2 Evasion Dataset-Level Analysis

Figure 3.1: Explanation-guided correlation analysis framework.

Table 3.1: Notations.

Notation	Brief Description
X_b	training set of black-box model f_b.
X_e	evasion set disjoint with X_b.
X'_e	adversarial counterpart of X_e.
X_s	training set of explanation model f_s.
$x \in X_e$	sample in evasion set.
$x' \in X'_e$	adversarial variant of x.
$Y = \{y_1, ..., y_k\}$	set of classes (labels)
$x = [x_1, ..., x_d]$	d-dimensional feature vector of sample x.
$W_{x,y_i} = \{w_1, ..., w_d\}$	feature weights (explanations) of a sample x toward the class y_i.
$pos(x, y_i)$	number of features in x *positive* to the prediction $f_b(x) = y_i$
$neg(x, y_i)$	number of features in x *negative* to the prediction $f_b(x) = y_i$
$neut(x, y_i)$	number of features in x *neutral* to the prediction $f_b(x) = y_i$
$P(x')$	number of perturbed features in x'
τ	threshold to decide high-correlated adversarial samples.

one examines evasion success at the resolution of each adversarial sample, aggregate evasion rate could give false sense of adversarial success for it lacks feature-level fidelity of perturbations that result in an adversarial sample. To address the stated lack of fidelity in aggregate evasion accuracy, we systematically explore how ML explanation methods are harnessed to assess feature-level correlations between pre-evasion adversarial perturbations and post-evasion explanations.

Figure 3.1 shows an overview of our explanation-guided correlation analysis framework. Given an evasion set X'_e of adversarial samples, our framework enables correlation analysis both at the sample-level (for each $x' \in X'_e$ at the granularity of each perturbed feature) and at the evasion dataset-level ($\forall x' \in X'_e$). Intuitively, given a decisive feature (obtained via ML explanations) of an evasive sample ($f_b(x') \neq y_{true}$), for such a feature to be considered the cause of (correlated to) the evasion, there needs to be a corresponding feature that was perturbed in the original sample x. By repeating this process of correlating each decisive feature with its perturbed counterpart, our sample-level correlation analysis seeks to establish empirical evidence that links an evasive prediction with the cause.

More precisely, our correlation analysis is performed by harnessing the *Post-Evasion Features Directions* ("2. Explanation" in Figure 3.1) of adversarial samples ("1. Evasion" in Figure 3.1). First, we explore the feature directions of the *Pre-evasion Perturbations* in order to obtain an assessment of the contribution of each feature perturbation to the attack (i.e., *feature-level assessment*). Second, we use those results in order to zoom-out to a *Sample-Level Assessment* (3.3.3). Finally, we move to the higher level of the whole evasion dataset in order to obtain an overall assessment of the evasion attack (3.3.4).

Conducting such fine-grained correlation analysis has two key benefits. Firstly, it verifies whether evasion can be attributed to the adversarial perturbations employed on the sample, and, in effect, performs diagnosis on aggregate evasion accuracy. Secondly, it provides visibility into how sensitive certain samples and/or features are to adversarial perturbations, which could inform efforts that aim to build ML models robust and dependable in the face of evasion attacks.

3.3.2 Post-Evasion Feature Direction

In a typical classification task, for an input sample x, $f_b(x) = y_i \in Y = \{y_1, ..., y_k\}$, where Y is the set of k possible labels. For example, in the multi-class handwritten digit recognition model of the MNIST [80] dataset, the input is an image of a handwritten digit and the label is one of the 10 digits (i.e., $Y = \{0, .., 9\}$ where $k = 10$). In the malware detection domain, the typical model is a binary classifier (i.e., $Y = \{\texttt{Benign}, \texttt{Malware}\}$ where $k = 2$). In the following, we use MNIST as an illustrative example to describe post-evasion feature direction.

Explanations returned from ML explanation methods reveal the *direction* of each feature. For each class $y_i \in Y$ and an adversarial sample x', a ML explanation method returns a set of *feature weights* $W_{x',y_i} = \{w_1, .., w_d\}$ where w_j reflects the importance (as the magnitude of w_j) and the *direction* (as the sign of w_j) of the feature x'_j towards the prediction $f_b(x') = y_i$. Depending on the sign of w_j, feature x'_j can be *positive*, *negative*, or *neutral* with respect to the prediction $f_b(x') = y_i$. When $w_j > 0$, we say x'_j is positive to (directed towards) y_i. Conversely, when $w_j < 0$, x'_j is negative to (directed away from) y'_i. When $w_j = 0$, we say x'_j is neutral to y_i (does not have any impact on the prediction decision). In case of binary classification ($k = 2$), if x'_j is not directed to the label y_1 (`Benign` for malware detection) and is not neutral, then x'_j can only be directed to the other label y_2 (`Malware`) and vice versa. To illustrate how we leverage feature

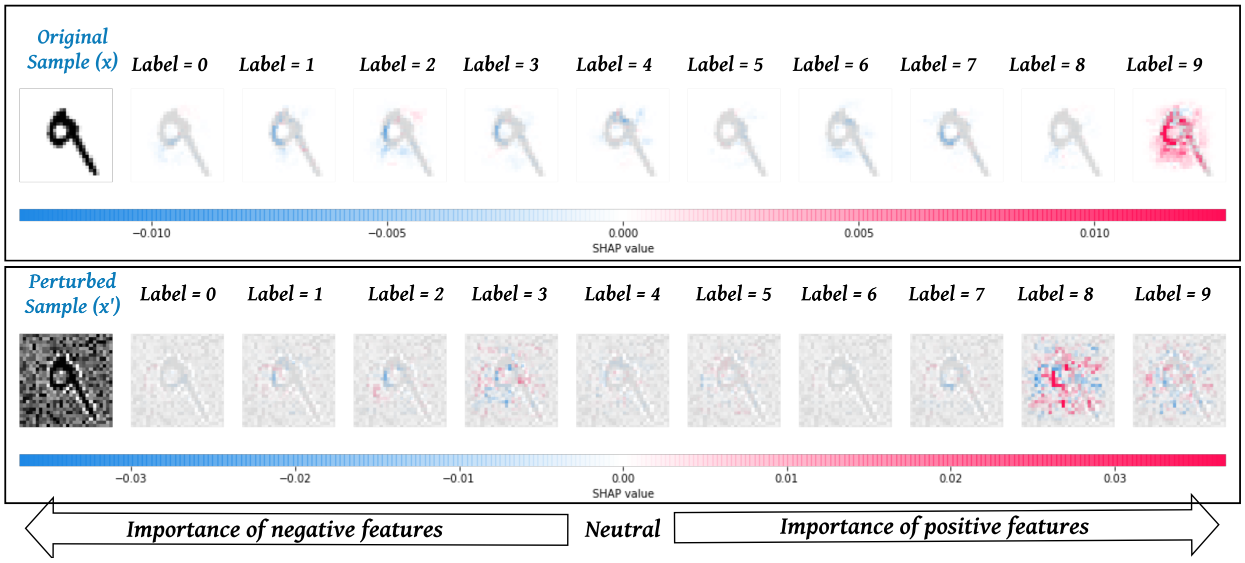


Figure 3.2: A comparative illustration of pre-perturbation and post-perturbation explanations using the SHAP [87] ML explanation framework on a test sample from the MNIST [80] dataset.

direction in our analysis, next we describe a concrete example from the MNIST [80] handwritten digit recognition model.

In Figure 3.2, the upper box shows SHAP [87] pre-evasion feature explanations of a correct prediction on an image of number "9" such that $f_b(x) = 9$. The lower box shows post-evasion feature explanations of the misclassification $f_b(x') \neq 9$ using an adversarial variant $x' \leftarrow x + \delta$ (perturbation = δ). Each column (i.e., "$Label = y_i$"; $y_i \in \{0, ..., 9\}$) represents the feature directions for the possibility of a prediction $f_b(x) = y_i$ (upper box) and $f_b(x') = y_i$ (lower box). The color codes are interpreted as follows: given an explanation, pink corresponds to positive features while blue corresponds to negative features. The intensity of either color (pink or blue) is directly proportional to the importance of the feature weight towards the prediction. Neutral features are represented with white. For instance, focusing on the correct prediction label $y_{true} = 9$ in the upper-box, we notice a large concentration of pink features which positively contribute to the predicted label (9).

Our approach primarily relies on post-evasion explanations (lower box in Figure 3.2) and we observe that feature importance weights vary for each studied label as a potential prediction $f_b(x') = y_i \in \{0, ..., 9\}$. When the prediction $f_b(x') = 8$ (image below 'Label = 8' in lower box), the explanations show that most features are *positive* (directed to label 8), which explains the change of the prediction label from 9 to 8. Examining the colors, we realize that most features that were directed to label 9 in the pre-perturbation explanations have become either neutral to the prediction $f_b(x') = 9$ or are positive towards $f_b(x') = 8$. It is noteworthy that some perturbed features are oriented to the original label 9 (notice pink pixels in the image below 'Label = 9' in lower box). Such observations suggest that even though the attack is successful (i.e., $f_b(x') = 8 \neq 9$), the effectiveness of each single feature perturbation is not guaranteed to result in an evasive prediction. Thus, the evasion success may not always be correlated with each feature perturbation the adversary performs on the original sample. We, therefore, argue that a perturbation strategy that produces many features that are uncorrelated with the misclassification might perform poorly on other feature representations (e.g., colored or not centered images in image classification) or other feature types (e.g., static vs. dynamic features in malware detection) which reflects a potential limitation of the stability of a perturbation method. Next, we introduce novel sample-level metrics that capture the fine-grained assessment that leverages post-evasion explanations. We refer to Table 3.1 for the feature direction-related notations. Our focus will be on the *post-perturbation feature directions* of an evasive sample x' and we suppose that its original prediction (pre-perturbation) is $f_b(x) = y_{true}$.

3.3.3 Sample-Level Analysis

Post-evasion explanations reveal the direction (*positive*, *negative*, or *neutral*) of each perturbed feature in an adversarial sample x'. Sample-level analysis is performed in order to empirically assess feature perturbations that positively contribute towards misclassification (*positive perturbations*) against the ones that contribute to maintain the true label as a prediction (*negative perturbations*). Next, we introduce two sample-level metrics which will later serve as foundations to conduct overall correlation analysis over the evasion dataset.

Definition 1: Per-Sample Perturbation Precision (PSPP). Out of all feature perturbations ($P(x')$) performed to produce an adversarial sample x', ***PSPP*** enables us compute the rate of perturbations that contribute to change the original prediction y_{true} to another label $y_i \in Y - \{y_{true}\}$. In other words, it measures the rate of perturbed features that are ***"negative"*** to the original prediction (y_{true}) and ***"positive"*** to other predictions $y_i \neq y_{true}$. We call such perturbations *positive perturbations* because they positively advance the evasion goal. More formally, the Per-Sample Perturbation Precision for an adversarial sample x' is computed as follows:

$$PSPP(x') = \frac{1}{2}\left(\frac{1}{k-1}\left(\sum_{\substack{y_i \in Y \\ y_i \neq y_{true}}} \frac{pos(x', y_i)}{P(x')}\right) + \frac{neg(x', y_{true})}{P(x')}\right) \tag{3.3}$$

Equation 3.3 is the average of two ratios:

• $\left(\frac{1}{k-1}\left(\sum_{\substack{y_i \in Y \\ y_i \neq y_{true}}} \frac{pos(x', y_i)}{P(x')}\right)\right.$: The average rate of perturbed features that are directed to a class $y_i \neq y_{true}$, over all $k-1$ possible false classes $y_i \in Y - \{y_{true}\}$.

• $\left(\frac{neg(x', y_{true})}{P(x')}\right)$: The rate of perturbed features that are not directed to the original label y_{true} and not neutral.

Both ratios that are considered in Equation 3.3 measure *Positive Perturbations* that contribute to a misclasssification. We note that $PSPP(x')$ falls in the range $[0, 1]$. The closer $PSPP(x')$ is to 1, the more the overall perturbations performed on the features of x' are precise (effective at feature level). More importantly, when x' evades the model, i.e., $f_b(x') \neq f_b(x)$, then the closer $PSPP(x')$ is to 1 the stronger the correlation between the evasion success and each performed feature perturbation that produced adversarial sample x'.

Definition 2: Per-Sample Perturbation Error (PSPE). Another per-sample measurement for our correlation analysis is the Per-Sample Perturbation Error, $PSPE(x')$, that computes the rate of perturbed features that are directed to the original class y_{true} (*positive* to the original prediction y_{true}). These features stand against the adversary's goal of misclassifying x'. Such features are considered *negative perturbations* with respect to the original class. More formally, $PSPE(x')$ is defined as follows.

$$PSPE(x') = \frac{pos(x', y_{true})}{P(x')} \tag{3.4}$$

Given an adversarial sample x', $PSPE(x')$ returns the rate of perturbation errors over all perturbed features. We note that a perturbed feature that is *neutral* ($w_j = 0$) to the original prediction ($f_b(x') = y_{true}$) is considered neither as perturbation error nor an effective manipulation to advance the evasion goal. Thus, $PSPE(x')$ may not be directly computed from $PSPP(x')$ and vice versa. Moreover, in the case of a slightly different threat model in-which the evasion is *targeted* to change the original prediction y_{true} to a new target label $y_{target} \in Y - \{y_{true}\}$, then only the term $\frac{pos(x', y_{target})}{P(x')}$ would be considered to compute the perturbation precision $PSPP(x')$, and only the term $\frac{neg(x', y_{target})}{P(x')}$ suffices to compute the rate of committed perturbation errors, $PSPE(x')$.

3.3.4 Evasion Dataset-Level Analysis

Using $PSPP(x')$ and $PSPE(x')$ defined in Equations 3.3 and 3.4 as foundations, we now introduce novel correlation analysis metrics that operate at the level of the evasion dataset X'_e to empirically analyze correlation between perturbations and post-evasion explanations.

Definition 3: High-Correlation Rate (HCR). As explained in Section 3.3.3, $PSPP(x')$ quantifies the correlation of each single feature perturbation with the evasion $f_b(x') \neq y_{true}$. The closer $PSPP(x')$ is to 1, the higher is the correlation and vice-versa. We consider a threshold τ that indicates the "strength" of the correlation between positive perturbations on x that resulted in x' and the important features that "explain" $f_b(x') \neq y_{true}$. Based on an empirically estimated τ, we call an adversarial sample x' a *High-Correlated Sample* if $PSPP(x')$ falls in $[\tau, 1]$. In our evaluation, based on empirical observations, we use $\tau = 0.5$.

Based on the above definition, we compute *High-Correlation Rate (HCR)* as the percentage of

High-Correlated Samples in the evasion set X'_e as follows:

$$HCR = \frac{|X'_e(PSPP > \tau)|}{|X'_e|} \tag{3.5}$$

where $X'_e(PSPP > \tau) = \{x' \in X'_e : PSPP(x') > \tau\} \cap \{f_b(x') \neq y_{true}\}$

We note that *HCR* quantifies the degree to which adversarial samples are both *evasive* and *correlated* to most feature perturbations performed on original samples.

Definition 4: Average Perturbation Error (APE). As shown in Equation 3.4, $PSPE(x')$ computes the number of errors committed during the perturbation of each feature in x to produce the manipulated sample x' (which is the same as computing the number of *negative perturbations*). We leverage $PSPE(x')$ to compute the average of negative perturbations (APE) over all samples in X'_e. Formally, APE is given as follows:

$$APE = \sum_{x' \in X'_e} \frac{PSPE(x')}{|X'_e|} \tag{3.6}$$

As opposed to aggregate evasion rate that computes the percentage of *evasive samples* versus *non-evasive samples* without deeper insights about the effectiveness of each single feature perturbation, APE computes the rate of *"evasive features"* versus *"non-evasive features"* of each evasive sample, over all perturbed samples. Such in-depth investigations into evasion attacks provide a fine-grained assessment of any evasion strategy on ML models.

3.4 Evaluation

We now evaluate the utility of our suite of metrics for high-fidelity correlation analysis of ML evasion attacks. We first validate our methodology in Section 3.4.3, and extend our evaluation with two case studies in Section 3.5.1 and 3.5.2.

3.4.1 Datasets

We use three datasets from two domains. From the malware classification domain, we use two complementary datasets, one based on static analysis, and the other on execution behavioral analysis. From image classification, we use a benchmark handwritten digits recognition dataset. We selected these two as representative domains because (a) malware detection is a naturally adversarial domain where adversarial robustness to evasion attacks is expected and (b) image recognition has been heavily explored for evasion attacks in recent adversarial ML literature [32]. We describe these datasets next.

Dataset-1 (PE Malware). To validate our framework on dynamic analysis based malware classifiers, we collected 40K Windows PE files with 50% malware (collected from VirusShare [2]) and the other 50% benign PEs (collected from a public goodware site [1]). We use 60% of the dataset as a training set for the target black-box model, 25% as a training for explanation substitute model, and the remaining 15% as evasion test. Each sample is represented as a binary feature vector. Each feature indicates the presence/absence of behavioral features captured up on execution of each PE in the Cuckoo Sandbox [5]. Behavioral analysis of 40K PEs resulted in 1549 features, of which 80 are API calls, 559 are I/O system files, and 910 are loaded DLLs.

EMBER (PE Malware). To assess our framework on complementary (static analysis-based) malware dataset, we use EMBER [17], a benchmark dataset of malware and benign PEs released with a trained LightGBM with 97.3% test accuracy. EMBER consists of 2351 features extracted from 1M PEs using a static binary analysis tool LIEF [6]. The training set contains 800K samples composed of 600K labeled samples with 50% split between benign and malicious PEs and 200K unlabeled samples, while the test set consists of 200K samples, again with the same ratio of label split. VirusTotal [3] was used to label all the samples. The feature groups include: PE metadata, header information, byte histogram, byte-entropy histogram, string information, section information, and imported/exported functions. We use 100K of the test set for substitute model training, and the remaining 100K as our evasion set against the LightGBM pre-trained model and a DNN which we trained. We use version 2 of EMBER.

MNIST (Image). To further evaluate our framework on image classifiers, we use the MNIST [80] dataset, which comprises 60K training and 10K test images of handwritten digits. The classification task is to identify the digit corresponding to each image. Each 28x28 gray-scale sample is encoded as a vector of pixel intensities in the interval [0 : 1].

3.4.2 Models and Setup

Next, we describe models trained, evasion attacks we employed, and explanation methods used for our experiments.

Studied ML Models. Overall, across Dataset-1, EMBER and MNIST, we train 8 models: Multi-Layer Perceptron (MLP), Logistic Regression (LR), Random Forest (RF), Extra Trees (ET), Decision Trees (DT), Light Gradient Boosting decision tree Model (LGBM), a feed-forward Deep Neural Network (DNN), and a 2D Convolutional Neural Network (CNN). Following prior work [95, 96], we choose these models because they are representative of applications of ML across domains including image classification, malware/intrusion detection, and they also complement each other in terms of their architecture and susceptibility to evasion.

Employed Evasion Attacks. Using the evasion set of each dataset, we craft adversarial samples. For the evasion attack, we consider a threat model where the adversary has no knowledge about the target model, but knows features used to train the model (e.g., API calls for malware classifiers, pixels for image classifiers). More precisely, for Dataset-1 and EMBER we incrementally perturb features of a Malware sample until the model flips its label to Benign. Following previous adversarial sample crafting methods [63, 122], we adopt only additive manipulations. For instance, for binary features of Dataset-1 (where 1 indicates presence and 0 indicates absence of an API call), we flip only a 0 to 1. Similar to prior work [44], we also respect the allowable range of perturbations for each static feature in EMBER (e.g., file size is always positive). For MNIST, we add a random noise to the background of the image to change the original gray-scale of each pixel without perturbing white pixels that characterize the handwritten digit. The outcome is an adversarial image that is still recognizable by humans, but misclassified by the model. Table II shows the comparison between pre-evasion accuracy and post-evasion accuracy. All models exhibit significant drop in the test accuracy after the feature perturbations. We recall that the main purpose of our analysis is to explore the correlation between a perturbed feature and the misclassification result, regardless of the complexity of the evasion strategy. Thus, our choice of perturbation methods is governed by convenience (e.g., execution time) and effectiveness (i.e., results in evasion).

Employed ML Explanation Methods. Informed by recent studies [132, 50] that compare the utility of ML explanation methods, we use LIME [101] on Dataset-1 and EMBER, and SHAP [87] on MNIST. More specifically, these studies perform comparative evaluations of black-box ML explanation methods (e.g., LIME [101], SHAP [87], and LEMNA [59]) in terms of effectiveness (e.g., accuracy), stability (i.e., similarity among results of different runs), efficiency (e.g., execution time), and robustness against small feature perturbations. On the one hand, these studies show that LIME performs best on security systems (e.g., Drebin+ [21], Mimicus+ [59]). Thus, we employ LIME on the two malware detection systems (i.e., Dataset-1 and EMBER). On the other hand, SHAP authors proposed a ML explainer called "Deep Explainer", designed for deep learning models, specifically for image classification. Thus, we use SHAP to explain predictions of a CNN on MNIST. We note that two independent recent studies [132, 50] have also noted that both LIME and SHAP outperform LEMNA [59] (hence, not included in our evaluations).

3.4.3 Correlation Analysis Results

We use the suite of correlation analysis metrics introduced in Section 7.2 (Equations 3.3-3.6). In Table 3.3, column 3 is *HCR* (Equation 3.5) and column 4 is *APE* (Equation 3.6).

Results Overview. Across all models and the three datasets, the evasion attack scores an average $HCR = 55\%$ and $APE = 36\%$. Linking back to what these metrics mean, an average

Table 3.2: Pre-evasion accuracy and post-evasion accuracy across studied models.

Dataset	**Model**	**Pre-Evasion Accuracy**	**Post-Evasion Accuracy**	**Aggregate Evasion Accuracy.**
Dataset-1	MLP	96%	6.05%	89.95%
Dataset-1	LR	95%	21.75%	73.25%
Dataset-1	RF	96%	18.61%	77.39%
Dataset-1	DT	96%	7.8%	88.20%
Dataset-1	ET	96%	16.27%	79.73%
EMBER	LGBM	97.3%	56.06%	40.94%
EMBER	DNN	93%	12.08%	80.92%
MNIST	CNN	99.4%	33.1%	66.30%

Table 3.3: High-Correlation Rate (HCR) and Average Perturbation Error (APE) values across studied models.

Dataset	**Model**	**High-Correlation Rate**	**Average Perturbation Error**
Dataset-1	MLP	34.56%	32.96%
Dataset-1	LR	34.96%	38.92%
Dataset-1	RF	36.09%	60.73%
Dataset-1	DT	66.85%	37.86%
Dataset-1	ET	33.41%	54.62%
EMBER	LGBM	95.03%	7.47%
EMBER	DNN	96.72%	4.89%
MNIST	CNN	44.31%	49.40%

$HCR = 55\%$ shows that for each model an average of only 55% of the adversarial samples have strong feature-level correlation with their respective perturbations. That entails an average of 45% adversarial samples per-model are loosely correlated with their perturbations). *APE* assesses the per-model average number of negative perturbations per sample. Results in Table 3.3 suggest a significant rate of negative perturbations are produced by the evasion attack. More precisely, on average across all models around 36% of the perturbations are negative (i.e., they lead the evasion strategy in the wrong direction, by increasing the likelihood of predicting the original label). Next, we expand on these highlights of our findings.

Does evasion imply consequential perturbations for all features? Although an evasion attack can achieve a seemingly high aggregate evasion rate (e.g., as high as 94% accuracy drop on MLP on Dataset-1), we notice that, the correlation between each single feature perturbation and a misclassification is not guaranteed. In fact, averaged across models, 45% of the crafted adversarial samples have low-correlated perturbations more than high-correlated ones ($PSPP(x') < 0.5$), suggesting that almost 1 in 2 adversarial samples suffers from weak correlation between post-evasion explanations and pre-evasion perturbations. As a result, counting in such samples in the aggregate evasion rate would essentially give false sense of the effectiveness of an attack strategy at the granularity of each feature perturbation. We underscore that such insights wouldn't have been possible to infer without the high-fidelity correlation analysis. In summary, these results confirm that *not all evasive predictions of an adversarial sample are correlated with the performed feature manipulations*.

Visualizing the Per-Sample Perturbation Precision. Figure 3.3 shows visual interpretation of the distribution of the Per-Sample Perturbation Precision ($PSPP$) values of all crafted malware samples of Dataset-1 across the 5 models. In the figure, we use the shorthand PP instead of $PSPP$ in the y-axis and we refer to the index of each sample in x-axis. The true prediction of each malware sample is $f_b(x') = 1$, while the evasive prediction is $f_b(x') = 0$ (i.e., adversarial malware sample is misclassified as benign). The red line in the middle represents the threshold τ that decides whether the adversarial sample has more positive perturbations or more negative ones. In almost all the plots, we notice the occurrence of a significant number of low-correlation adversarial malware samples ($PP(x') < \tau$) that evaded the classifier (purple circles below the red line). Once again, these findings suggest that the evasion attack results in a high number of negative perturbations. However, despite the low number of positive perturbations for these samples, the evasion is still successful. This result goes along with our previous finding that high evasion aggregate accuracy can be totally uncorrelated with the performed perturbations. Thus, even for a successful evasion the perturbations at the feature-level can apparently be ineffective. These insights can only be confirmed by examining the high-fidelity correlation analysis results.

It is noteworthy that Figure 3.3 exhibits an unusual behavior. In particular, some crafted samples with a true prediction $f_b(x') = 1$ (yellow circles) appear to have high Perturbation Precision, $PP(x') > \tau$, despite the failure to flip the model's prediction from 1 to 0. This is especially true for models: LR, DT, ET and RF. On one hand, this observation suggests that even a small number of negative perturbations ($PP(x') < \tau$) may affect the final outcome of the evasion. On the other hand, it suggests potential limitations of Black-Box ML explanation methods in terms of accuracy and stability between different runs. More discussion is provided about this in Section 3.6.

What do correlation analysis results suggest across different classification tasks and model architectures? While our results so far strongly suggest the importance of post-evasion correlation analysis for an in-depth assessment of an evasion attack strategy, we also observe that HCR and APE values vary across studied domains (malware, image), model architectures, and feature representations (static, dynamic). This variation speaks to the sensitivity of different domains, models, and feature values to adversarial feature perturbations with implications on robustness and dependability in the face of individual feature perturbation. In fact, ML models trained on EMBER (LGBM and DNN) showed acceptably low rate of negative perturbations (i.e., $APE = 6\%$ on average) and a high rate of samples with highly-correlated perturbations (i.e., $HCR = 96\%$

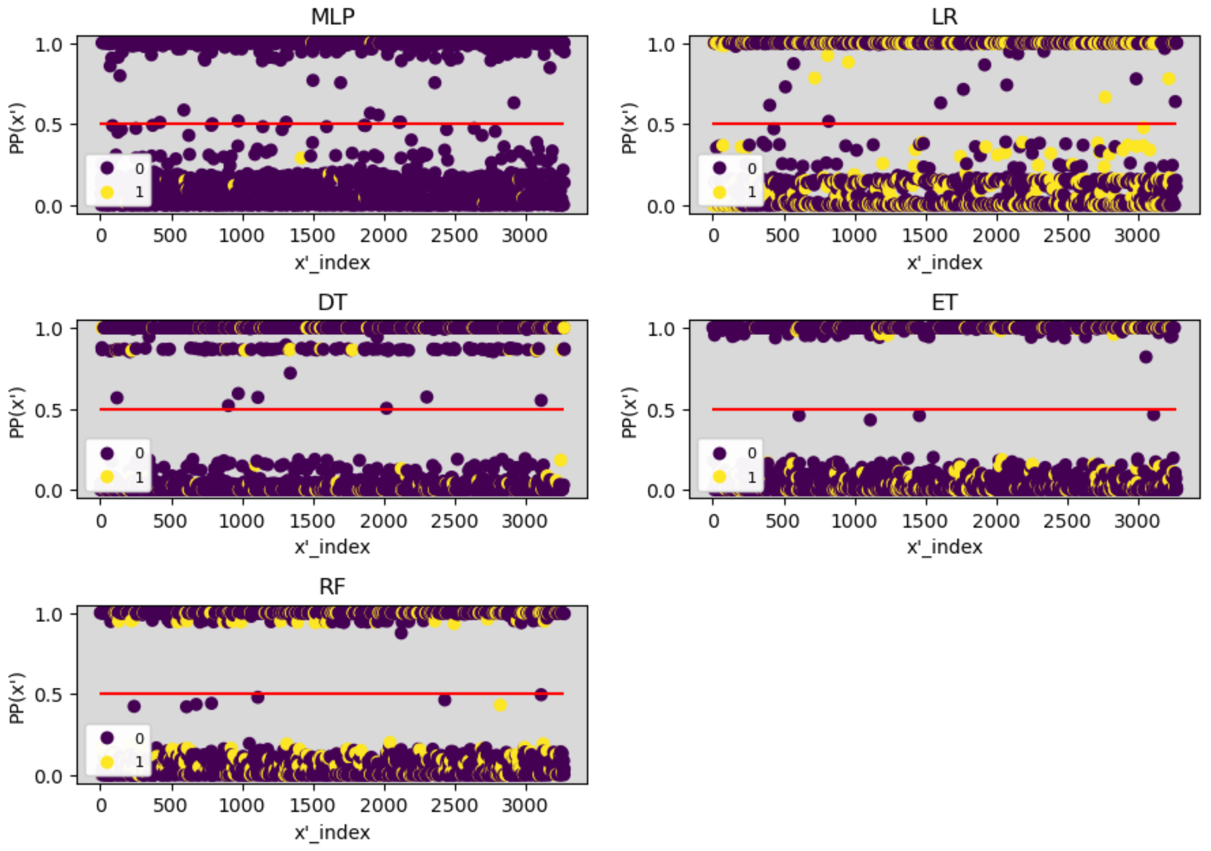


Figure 3.3: Distribution of Perturbation Precision (PSPP) values of each adversarial sample across models on Dataset-1.

on average). This suggests that static features of Windows PE malware are more sensitive to a feature perturbation considering the higher rate of negative perturbations on dynamic features in Dataset-1. In terms of comparison between different domains and different ML models, despite the high evasion rate at sample-level, almost all ML models showed some robustness at the level of a single feature perturbation. Most importantly, RF on Dataset-1 showed the highest robustness since more than 60% of the overall feature perturbations are negative which suggest that they did not contribute in the misclassification decision. ET on Dataset-1 ($APE = 54\%$) and CNN on MNIST ($APE = 49\%$) showed lower robustness than RF, but higher than the other models.

★ **Summary.** Our results suggest that the aggregate evasion accuracy measured over all the 8 models is not enough to assess the efficacy of perturbation attack strategy. Our findings also validate that explanation-guided correlation analysis plays a crucial role in diagnosing aggregate evasion rates to winnow high-correlation adversarial samples from low-correlation ones for precise feature-level assessment of evasion accuracy. Furthermore, using our correlation analysis we succeeded at pinpointing the sensitivity of different malware feature types to perturbations.

3.5 Case Studies

We now present two case studies that demonstrate the application of our correlation analysis framework on (a) explanation-guided evasion strategy and (b) cross-model adversarial sample transferability analysis.

3.5.1 Case Study 1: Explanation-Guided Evasion Strategy

The correlation analysis results showed that, while an evasion strategy may result in an evasive adversarial sample, at the granularity of a single feature perturbation it may produce a considerable number of *negative perturbations*. In other words, from the adversary's standpoint, the correlation analysis can be leveraged towards more accurate evasion strategy that significantly minimizes negative perturbations. In the following, we explore the potential of explanation methods to guide a more effective evasion strategy.

Explanation-guided pre-perturbation feature selection. In this case study, we demonstrate how an adversary leverages ML explanation methods to examine pre-perturbation predictions before making feature manipulations. In particular, the *pre-perturbation feature directions* reveal *positive features* that significantly contribute to the true prediction (pink pixels in Figure 3.4). Intuitively, positive features are strong candidates for perturbations, while *negative features* (blue pixels in Figure 3.4) should be intact and need not be perturbed since they are already directed away from the true label, which is in favor of the adversary's goal. *Neutral features* (white pixels in Figure 3.4) are also not candidates for perturbations since they have no effect on the original label decision. We note that in this case study we consider all positive features (i.e., pink pixels) as candidates for perturbation regardless of the color intensity that represents its explanation weight. In Figure 3.4, some positive pixels ($w_i > 0$) with a low explanation weight ($w_i \sim 0$) are almost neutral (i.e., closer to the white color) but still perturbed since they are directed to the true label. Using the same experimental setup, we enhance the evasion strategies used on the three datasets with *explanation-guided pre-perturbation feature selection*. Then, we measure changes to *post-evasion accuracy*, *HCR*, and *APE* for all studied models.

Impact of explanation-guided pre-perturbation feature selection. Results in Table 3.4 suggest overall improvement not only in aggregate evasion accuracy, but also in the correlation strength between evasion explanations and individual feature perturbations. Comparing the "Post-Evasion Accuracy" columns of Tables 3.2 and 3.4, using explanation-guided evasion strategy post-evasion accuracy drops for all studied models, with an average per-model drop of 13.4% (which translates to the same percentage of improvement in aggregate evasion accuracy). Interestingly, in 4 out of the 5 models in Dataset-1, post-evasion accuracy drops to zero, with up to 21% drop in post-evasion accuracy for models such as LR. We note that the eventual complete evasion in al-

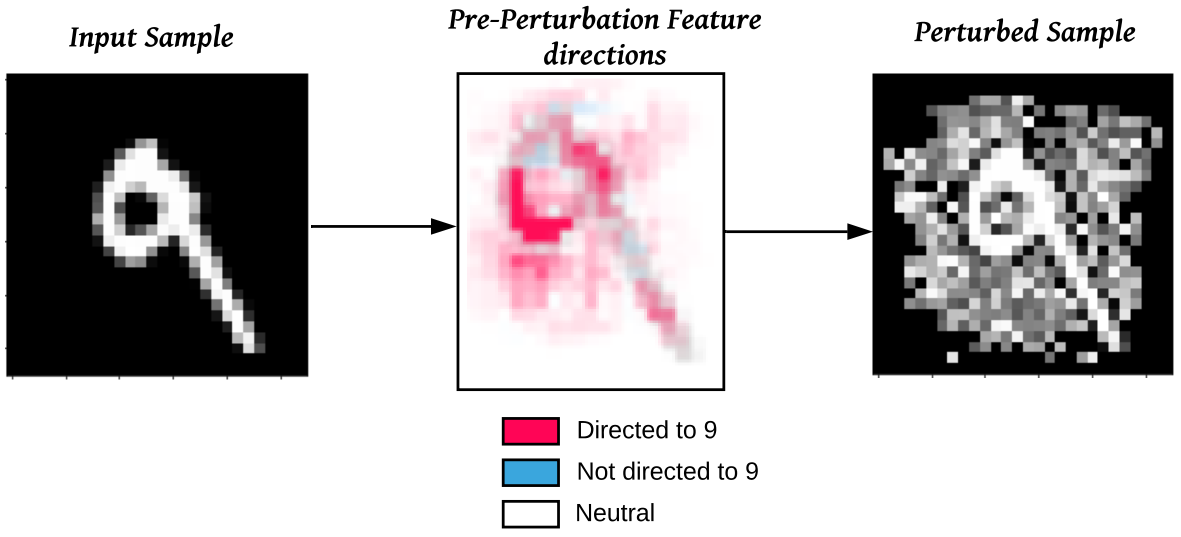


Figure 3.4: An example of explanation-guided feature perturbations on an input sample from MNIST.

most all models in Dataset-1 is most likely attributed to the binary nature of the features, where the explanation-guided feature selection filters out negative features and leaves only positive features that are flipped with just one perturbation. Comparing the HCR columns of Tables 3.4 and 3.3, we notice an increase in HCR for all studied models. On average, HCR increased by 27$\%$ per-model, which shows the positive utility of the pre-perturbation explanations that guided the evasion strategy to perturb positive features instead of negative ones. Again, comparing the APE columns of Tables 3.4 and 3.3, we notice a significant drop in APE, with an average per-model decrease of 20%, which indicates a decrease in the number of *negative perturbations*. Better performance in terms of post-evasion accuracy is also observed for all studied target models.

It is noteworthy that, although we perturb only *positive features*, in the APE column of Table 3.4 all values are still above zero. Ideally, the explanation method would guide the perturbation strategy to perform only positive perturbations and make no mistaken perturbations. Nevertheless, we still observe a minimal percentage of *negative perturbations* due to the inherent limitations of the accuracy and stability of explanations by LIME and SHAP, which is also substantiated by recent studies [132, 50] that evaluated LIME and SHAP among other ML explanation methods. We will expand on limitations of ML explanation methods in Section 3.6.

★ **Summary.** This case study suggests that when used on top of existing feature perturbation methods, an explanation-guided feature selection strategy leads to more effective evasion results both in terms of aggregate evasion accuracy and effectiveness at the level of each feature manipulation.

Table 3.4: Post-Evasion Accuracy, HCR, and APE values across studied models using explanation-guided evasion.

Dataset	Model	Post-Evasion Accuracy	HCR	APE
Dataset-1	MLP	0%	92.03%	14.22%
Dataset-1	LR	0%	96.25%	6%
Dataset-1	RF	0%	48.53%	51.31%
Dataset-1	DT	1.41%	98.84%	3.41%
Dataset-1	ET	0%	48.11%	1.21%
EMBER	LGBM	27.16%	99.58%	2.69%
EMBER	DNN	11.7%	97.8%	2.71%
MNIST	CNN	24.67%	64.5%	43.4%

3.5.2 Case Study 2: Explanation-Guided Transferability Analysis

One of the intriguing observations in evasion attacks against ML models is the transferability of samples crafted for one model to another, even architecturally dissimilar model [55, 95]. Similar to evasion accuracy, cross-model adversarial sample transferability is usually reported as the aggregate number of overlapping adversarial samples between a pair of models. One of the open questions around transferability is "what causes adversarial samples to transfer?". Previous work has pointed to causes such as limitations of the training process [95], scarcity of training data [95], inherent model vulnerability [46], and similarity (e.g., piecewise linearity) among different model architectures [55, 95]. However, the question is still under active investigation. Towards contributing a piece of clarity to address this question, we explore the utility of our metrics on cross-model adversarial sample transferability using Dataset-1 as a case study.

Aggregate transferability. Table 3.5 shows the percentage of adversarial samples that evade model$_i$ (row) that also evade model$_j$ (column). Overall, our transferability results are consistent with prior work that demonstrate the transferability phenomenon on malware (e.g., [46, 123]) and images (e.g., [55, 95, 46]. However, the aggregate transferability rates in Table 3.5 do not offer detailed insights as to the correlation between perturbations of transferable adversarial samples. Next, we use our metrics to dig deeper into the aggregate transferability rate at the level of perturbed features.

High-fidelity transferability analysis. Table 3.6 shows high-fidelity view of the transferability rates in Table 3.5, where each cell (i, j) is the average number of overlapping positive perturbations for a sample pair (x'_i, x'_j) from model$_i$ (row) and model$_j$ (column). We recall that a high-correlation adversarial sample x' has a high rate of positive perturbations ($PSPP(x') > \tau$). Overall, the percentage of overlapping positive perturbations in Table 3.6 show that *one of the*

Table 3.5: Aggregate adversarial sample transferability on Dataset-1. Cell (i, j) is the percentage of adversarial samples that evaded model$_i$ (row) that also evade model$_j$ (column).

	MLP	**LR**	**RF**	**ET**	**DT**
MLP	–	7%	4%	3%	18%
LR	86%	–	12%	5%	57%
RF	95%	68%	–	59%	83%
ET	97%	71%	53%	–	84%
DT	48%	30%	15%	16%	–

Table 3.6: Average overlap of positive perturbations on features detected among high-correlation adversarial samples transferable from model$_i$ (row) to model$_j$ (column).

	MLP	**LR**	**RF**	**ET**	**DT**
MLP	-	63.78%	53.68%	53.42%	60.66%
LR	55.54%	-	49.01%	50.50%	60.10%
RF	60.02%	62.14%	-	74.41%	61.28%
ET	60.14%	64.33%	75.34%	-	60.96%
DT	47.10%	53.83%	43.59%	43.53%	-

reasons for transferability is the empirically evident correlation between perturbations performed on a transferable adversarial sample pair (x'_i, x'_j) in model pair (model$_i$, model$_j$). For instance, looking at cell (ET, MLP) in Table 3.6 with 60.14% overlap in underlying positive perturbations, it can be seen from Table 3.5 that its counterpart cell has the highest aggregate transferability rate (97% to be exact). Among aggregate transferability rates reported in Table 3.5, cell (MLP, RF) and cell (MLP, ET) are the lowest transferability rates (MLP→RF: 4%, MLP→ET: 3%). Further examination of the corresponding cells in Table 3.6 suggests that within the 4% transferable samples for MLP→RF, we observe 53.68% average overlap of positive perturbations. Similarly for MLP→ET, within the 3% transferable samples, there exists 53.42% average overlap of positive perturbations among transferable sample pairs. Such deeper insights into feature-level correlation among transferable sample pairs once again demonstrate a practical use case for our methodology.

★ **Summary.** Our results suggest that shared positive perturbations among transferable sample pairs as one of the causes for adversarial sample transferability.

3.6 Discussion

Recent studies [132, 50] have systematically compared the performance of ML explanation methods especially on security systems. In addition to general evaluation criteria (e.g., explanation accuracy and sparsity), Warnecke et al. [132] focused on other security-relevant evaluation metrics (e.g., stability, efficiency, and robustness). Fan et al. [50] also proposed a similar framework that led to the same evaluation results. To discuss the limitations of the ML methods employed in our work (LIME [101] and SHAP [87]), in what follows we focus on *accuracy* (degree to which relevant features are captured in an explanation), *stability* (how much explanations vary between runs), and *robustness* (the extent to which explanations and prediction are coupled).

Limitations of Explanation Methods. While LIME and SHAP produce more accurate results compared with other black-box explanation methods such as DeepLIFT [114] and LEMNA [59]) the *accuracy* of the explanation may vary across different ML model architectures (e.g., MLP, RF, DT, etc), and across different ML tasks/datasets (e.g., Dataset-1, EMBER, and MNIST). For instance, the inherent linearity LIME's approximator could negatively influence its accuracy and stability in explaining predictions of complex models such as RF and ET. More importantly, like all learning-based methods, LIME and SHAP are sensitive to non-determinism (e.g., random initialization, stochastic optimization) which affect their *stability* between different runs. In other words, it is likely to observe a slight variation in the output of multiple runs performed by the same explanation method using the same input data. In fact, we observed that the average difference between Shapley values (i.e., feature importance weights) returned by SHAP is around 1% over 100 runs on the same MNIST sample. Such variation in ML explanation methods outputs might partly explain some of the unexpected results of our explanation-guided analysis that we noted in Sections 3.4.3, 3.5.1, and 3.5.2.

Vulnerability of Explanation Methods. Another issue worth considering is *robustness* of ML explanation methods against adversarial attacks. Recent studies [52, 62] have demonstrated that the explanation results are sensitive to small systematic feature perturbations that preserve the predicted label. Such attacks can potentially alter the explanation results, which might in effect influence our explanation-guided analysis. Consequently, our analysis may produce potentially misleading results for correlation metrics such as HCR and APE.

Future Outlook. In light of the utility of ML explanations we demonstrated in this chapter, we believe that the currently active research in explanation methods will lead to more reliable and robust ML explanation methods. We also note that vulnerability to adversarial attacks is a broader problem for any ML model, and progress in defense against attacks such as adversarial examples will potentially inspire and inform stronger robustness properties for ML explanation methods.

3.7 Conclusion

In this chapter, we introduced the first explanation-guided methodology for the diagnosis of ML evasion attacks. The core insight of the methodology is the use of feature importance-based ML explanation methods to perform high-fidelity correlation analysis between feature perturbations and post-evasion prediction explanations. To systematize the analysis in a model-agnostic manner, we proposed and evaluated a novel suite of metrics. Using image classification and malware detection as representative ML tasks, we demonstrated the utility of the methodology across diverse ML model architectures and feature representations. Through two case studies we additionally confirm that our methodology enables evasion attack improvement via pre-evasion feature direction analysis and feature-level correlation analysis of transferable adversarial samples. We believe this work serves as the first step towards fine-grained sanity check of evasion attacks to build dependable and secure ML systems.

CHAPTER 4

EG-Booster: Explanation-Guided Booster of ML Evasion Attacks

4.1 Introduction

Machine Learning (ML) models are vulnerable to test-time evasion attacks called *adversarial examples* –adversarially-perturbed inputs aimed to mislead a deployed ML model [55]. Evasion attacks have been the subject of recent research [55, 79, 89, 47, 35, 36, 128] that led to understanding the potential threats posed by these attacks when ML models are deployed in security-critical settings such as autonomous vehicles [105], malware classification [72], speech recognition [41], and natural language processing [19].

Given a ML model f and an input x with a true label y_{true}, the goal of a typical evasion attack is to perform minimal perturbations to x and obtain x' similar to x such that f is fooled to misclassify x' as $y' \neq y_{true}$. For a defender whose goal is to conduct pre-deployment robustness assessment of a model to adversarial examples, one needs to adopt a reliable input crafting strategy that can reveal the potential security limitations of the ML model. In this regard, a systematic examination of the suitability of each feature as a potential candidate for adversarial perturbations can be guided by the contribution of each feature in the classification result [11]. In this context, recent progress in feature-based ML explanation techniques [101, 87, 59] is interestingly positioned inline with the robustness evaluation goal. In fact, ML explanation methods have been recently utilized to guide robustness evaluation of models against backdoor poisoning of ML models [111], model extraction attacks [92, 23], and membership inference attacks [113], which highlights the utility of ML explanation methods beyond ensuring the transparency of model predictions.

In this chapter, we present EG-Booster , an explanation-guided evasion booster that leverages techniques from explainable ML [101, 87, 59] to guide adversarial example crafting for improved robustness evaluation of ML models. Inspired by a case study in [11], our work is the first to leverage black-box model explanations as a guide for systematic robustness evaluation of ML models against adversarial examples. The key insight in EG-Booster is the use of feature-based explanations of model predictions to guide adversarial example crafting. Given a model f, a d-dimensional input sample x, and a true prediction label y_{true} such that $f(x) = y_{true}$, a ML explanation method returns a weight vector $W_{x,y_{true}} = [w_1, ..., w_d]$ where each w_i quantifies the contribution of feature x_i to the prediction y_{true}. The sign of w_i represents the *direction* of feature x_i with respect to y_{true}. If $w_i > 0$, x_i is directed towards y_{true} (in this case we call x_i a *positive feature*). In the opposite case, i.e., $w_i < 0$, x_i is directed away from y_{true} (in this case we call x_i a *negative feature*). EG-Booster leverages sign of individual feature weights in two complementary ways:

- First, it uses positive weights to identify positive features that are worth perturbing and *introduces consequential perturbations* likely to result in the misclassification of x'.
- Second, it uses negative weights to identify negative features that need not be perturbed and *eliminates non-consequential perturbations* unlikely to result in the misclassification of x'.

is agnostic to model architecture, adversarial knowledge and capabilities (e.g., black-box, white-box), and supports diverse distance metrics (e.g., common $||.||_p$ norms) used previously in the adversarial examples literature.

In an orthogonal line of study on explanation *stability* and adversarial *robustness* of model explanations, some limitations of explanation methods have been documented [50, 132]. Recognizing these potential limitations which entail systematic vetting of the reliability of ML explanation methods before using them for robustness assessment of ML models, we introduce an explanation assessment metric called *k-Stability*, which measures the average stability of evasion results based on the similarities of the target model's predictions returned by multiple runs of EG-Booster .

We evaluate EG-Booster through comprehensive experiments on two benchmark datasets (MNIST and CIFAR10), across white-box and black-box attacks, on state-of-the-art undefended and defended models, covering commonly used $||.||_p$ norms. From white-box attacks we use the Fast Gradient Sign Method (FGS) [55], the Basic Iterative Method (BIM) [79], the Projected Gradient Descent method (PGD) [89], and the Carlini and Wagner attack (C&W) [35]. From black-box attacks, we use the Momentum Iterative Method (MIM) [47], the HopSkipJump Attack (HSJA) [36], and the Simultaneous Perturbation Stochastic Approximation (SPSA) attack [128].

Across all studied models, we observe a significant increase in the evasion rate of baseline attacks when combined with EG-Booster . Specifically, our results suggest an average increase of 28.78% in evasion rate across all attacks on undefended MNIST-CNN model. Similar findings are observed on undefended CIFAR10-CNN models and a defended CIFAR10-ResNet model with average evasion evasion rate increase, respectively, of 6.23% and 5.41%. In addition to the reduction of the total number of perturbations compared to the baseline attacks, these findings prove that ML explanation methods can be harnessed to guide ML evasion attacks towards more consequential perturbations. Furthermore, the stability analysis results show that, EG-Booster 's outputs are stable across multiple runs. Such findings highlight reliability of explanation methods to boost evasion attacks, despite their stability concerns highlighted in prior work. More detailed results are discussed in section 8.2.1.3.

In summary, this work makes the following contributions:

- **Explanation-Guided Evasion Booster:** We introduce the first approach that leverages ML explanation methods towards evaluating robustness of ML models to adversarial examples.
- **Stability Analysis:** We introduce a novel stability metric that enables vetting the reliability of ML explanation methods before they are used to guide ML robustness evaluation.
- **Comprehensive Evaluation:** We conduct comprehensive evaluations on two benchmark datasets, four white-box attacks, three black-box attacks, different $||.||_p$ distance metrics, on undefended and defended state-of-the-art target models.
- **Artifact Availability:** To foster reproducability, we have made available EG-Booster source code with directions to repeat our experiments at: `https://github.com/um-dsp/EG-Booster`.

4.2 EG-Booster Approach

In Section 4.2.1, we first describe an overview of EG-Booster . Next, in Sections 4.2.2, 4.2.3, and 4.2.4, we present details of EG-Booster .

4.2.1 Overview

In the reference attacks we introduced in Section 2.4, without loss of generality, the problem of crafting an adversarial example is stated as follows: given an input $x \in \mathbb{R}^d$ such that $f(x) = y_{true}$, the goal of the attack is to find the optimal perturbation $\delta^\star$ such that the adversarial example $x' = x + \delta^\star$ is misclassified as $f(x') = y \neq y_{true}$. This problem is formulated as:

$$\begin{aligned} \delta^\star = \arg\min \delta \in \mathbb{R}^d \quad & f(x+\delta) \\ \text{s.t.} \quad & ||\delta^\star|| < \epsilon \end{aligned} \tag{4.1}$$

The only constraint of Equation 4.1 is that the perturbation size of the vector δ is bounded by a maximum allowable perturbation size ϵ (i.e., $||\delta||_p < \epsilon$), which ensures that adversarial manipulations preserve the semantics of the original input. However, it does not guarantee that all single

Algorithm 1: Explanation-Guided Booster Attack.

```
Result: $x'_{boost}$, $N_{nc}$, $N_c$
Input:
$f$ : black-box classification function;
$x$: legitimate input sample;
$x'_{baseline}$: $x$'s adversarial variant returned by baseline attack;
$W_{x,y_{true}}$: feature weights (explanations) of a $x$ toward $y_{true}$;
$||.||_p$: norm used in the baseline attack;
$\epsilon$: perturbation bound used in the baseline attack;
$max\_iter$: maximum iterations to make bounded
  perturbation $\delta_i$;
Initialization:
$W_{x,y_{true}} \leftarrow ML\_Explainer(f, x, y_{true})$;
$d \leftarrow dim(x)$;
$x'_{boost} \leftarrow x'_{baseline}$;
$y_{baseline} \leftarrow f(x'_{baseline})$;
$N_{nc}$,$N_c \leftarrow 0$;
Output:
// Eliminate non-consequential
   perturbations
foreach $w_i \in (W_{x,y_{true}},), i \leq d$ do
    if $w_i < 0$ then
        $x'_{boost}[i] \leftarrow x[i]$;
        if $y_{baseline} \neq y_{true}$   $f(x'_{boost}) = y_{true}$ then
            $x'_{boost}[i] \leftarrow x'_{baseline}[i]$
        else
            $N_{nc} = N_{nc} + 1$
        end
    end
end
// Add consequential perturbations if
   baseline attack fails
foreach $w_i \in (W_{x,y_{true}},), i \leq d$ do
    if $w_i > 0$   $f(x'_{boost}) = y_{true}$ then
        $\delta_i = get\_delta(x)$;
         $x'_{boost}[i] \leftarrow x'_{boost}[i] + \delta_i$;
        $iter \leftarrow 0$;
         while $||x'_{boost} - x||_p > \epsilon$ do
            $iter \leftarrow iter + 1$;
             $reduce\_and\_repeat(\delta_i)$;
             if $iter = max\_iter$ then
                 break;
            end
        end
        if $||x'_{boost} - x||_p > \epsilon$ then
            $x'_{boost}[i] \leftarrow x'_{baseline}[i]$;
        else
            $N_c = N_c + 1$
        end
    end
end
```

Figure 4.1: Explanation-Guided Booster Attack.

feature perturbations (i.e., $x'_i = x_i + \delta_i$) are *consequential* to result in evasion. Our Explanation-Guided Booster (EG-Booster) approach improves Equation 4.1 to guide any L_p norm-based attack to perform only necessary perturbations that result in evasion. In addition to the upper bound constraint ϵ, EG-Booster satisfies an additional constraint that guarantees the perturbation of only the features that initially contribute to a correct prediction (*positive features*). In other words, EG-Booster guides the state-of-the-art attacks to selectively perturb features which have positive explanation weights towards the true label y_{true} (i.e., $w_i > 0$ such that $w_i \in W_{x,y_{true}} = \{w_1, ..., w_d\}$). Formally, the EG-Booster adversarial example crafting problem is stated as:

$$\begin{aligned} \delta^\star = \arg\min \delta \in \mathbb{R}^d \qquad & f(x+\delta) \\ \text{s.t.} \qquad & ||\delta^\star|| < \epsilon \\ & W_{x,y_{true}}(\delta^\star) > 0 \end{aligned} \tag{4.2}$$

The second constraint $W_{x,y_{true}}(\delta^\star) > 0$ ensures that only features with positive explanation weights are selected for perturbation.

As shown in Algorithm 4.1 (line 10), first, EG-Booster performs ML explanation on the input x to capture the original *direction* of each feature x_i with respect to the true label y_{true}. Next, using explanation results ($W_{x,y_{true}}$), EG-Booster reviews initial perturbations performed by a baseline attack by eliminating features that have negative explanation weight (lines 16–25) and adding more perturbations on features that have positive explanation weight (lines 26–40). We ensure that EG-Booster 's intervention is pre-conditioned on maintaining at least the same evasion success of an evasion attack, if not improving it. Specifically, in case the initial adversarial sample $x'_{baseline}$ generated by a baseline attack succeeds to evade the model, EG-Booster eliminates only perturbations that do not affect the initial evasion result. In this case, even if there exist unperturbed positive features when $x'_{baseline}$ was crafted, EG-Booster does not perform any additional perturbations since the evasion goal has been already achieved. Next, we use Algorithm 4.1 to further explain the details of EG-Booster .

4.2.2 Eliminating Non-Consequential Perturbations

As shown in Algorithm 4.1 (lines 16–25), EG-Booster starts by eliminating unnecessary perturbations that are *non-consequential* to the evasion result (if any). If the adversarial input $x'_{baseline}$ produced by the baseline attack is already leading to a successful evasion, EG-Booster ensures that the perturbation elimination step does not affect the initial evasion success (lines 19–20). Accordingly, EG-Booster intervenes to eliminate the perturbations that have no effect on the final evasive prediction. More precisely, it searches for perturbed features that are originally not directed to the true label y_{true} (line 17), and then it restores the original values of those features (i.e., eliminates

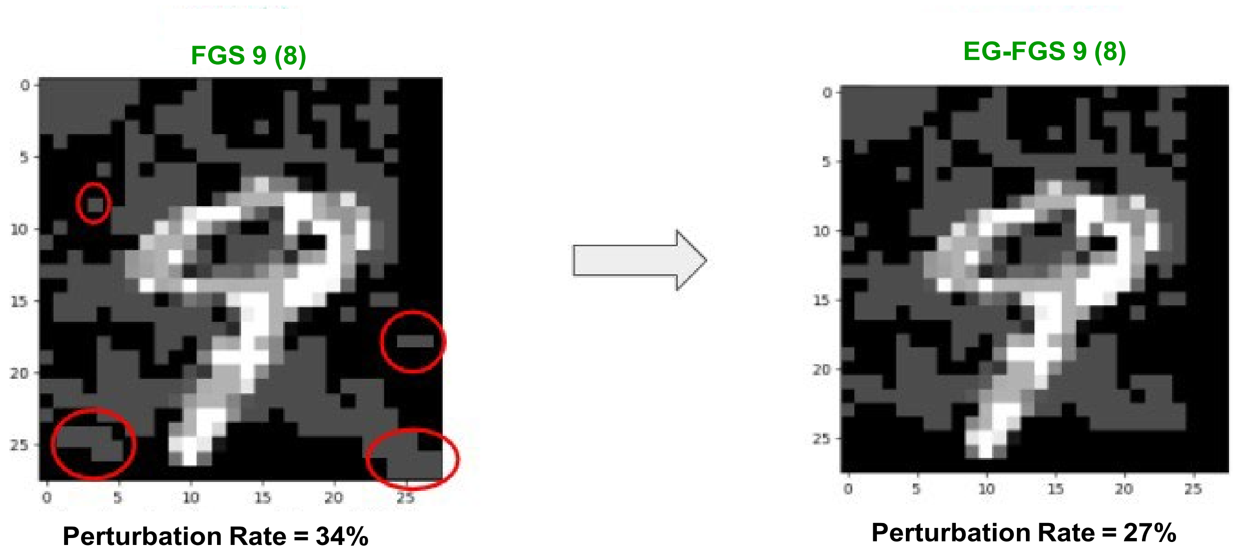


Figure 4.2: An example of the impact of eliminating non-consequential perturbations from a perturbed MNIST input that evades a CNN model using FGS. The new prediction of each version of the input image is in parentheses (.).

the perturbation) (line 18) while ensuring the validity of the initial evasion if it exists (it does so by skipping any elimination of feature perturbations that do not preserve the model evasion). As we will show in our experiments (Section 4.3.3), this step leads to a significant drop in the total number of perturbed features which results in a smaller perturbation size $||\delta^{\star}||_p$.

Figure 4.2 shows an MNIST-CNN example of the impact of enhancing the FGS [55] attack with EG-Booster when the baseline perturbations are already evasive. In this example, FGS perturbations fool the CNN model to mis-predict '9' as '8'. The red circles on the left-hand side image show the detection of non-consequential perturbations signaled by EG-Booster using the pre-perturbation model explanation results. The new version of the input image shown on the right illustrates the impact of eliminating those non-consequential feature perturbations. The number of feature perturbations is reduced by 7% while preserving the evasion success.

Alternatively, when the baseline attack initially fails to evade the model, it is crucial to perform this perturbation elimination step on all non-consequential features before making any additional perturbations. Doing so reduces the perturbation size $||\delta||$ which provides a larger gap ($\epsilon - ||\delta||$) for performing additional consequential perturbations more likely to result in misclassification.

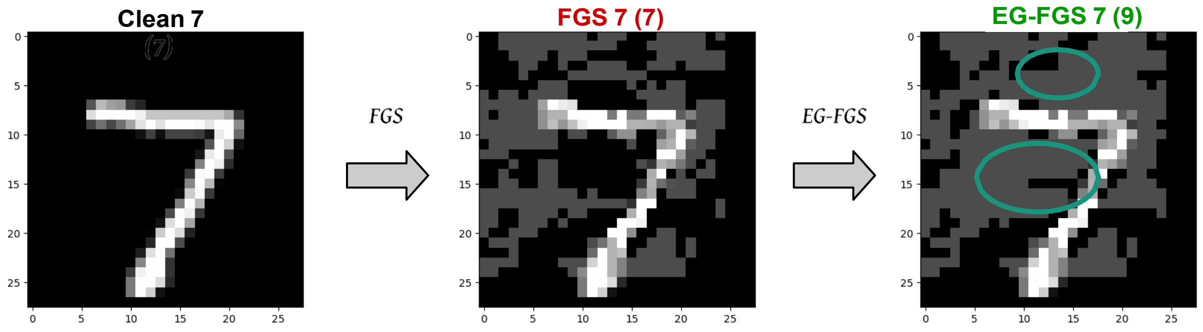


Figure 4.3: An example of the impact of adding consequential perturbations to a perturbed MNIST input that failed to evade a CNN model using FGS. We put the new prediction of each version of the input image in parentheses (.).

4.2.3 Adding Consequential Perturbations

This second step is only needed in case the adversarial sample has still failed to evade the model after eliminating non-consequential perturbations. In this case, EG-Booster starts searching for unperturbed features that are directed towards the true label y_{true}. When it finds such features, it then incrementally adds small perturbations δ_i to these features since they are signaled by the ML explainer as crucial features directed to a correct classification (lines 27–29). The function *get_delta(x)* chooses a random feature perturbation δ_i that satisfies the feature constraint(s) of the dataset at hand (line 28). For instance, in image classification, if the feature representation of the samples is in gray-scale, pixel perturbation should be in the allowable range of feature values (e.g., $[0, 1]$ for normalized MNIST pixel values in [0,255]). In case a feature perturbation $x_i + \delta_i$ crosses the upper bound constraint $||\delta||_p > \epsilon$, EG-Booster iteratively keeps reducing the perturbation δ_i until the bound constraint is reached or the number of iterations hits its upper bound (lines 30–37).

In case all features directed to the correct label are already perturbed, EG-Booster proceeds by attempting to increase the perturbations initially performed by the baseline attack on positive features. It does so while respecting the upper bound constraint ($||\delta|| < \epsilon$). This process terminates when evasion succeeds.

Figure 4.3 shows an MNIST-CNN example of the impact of enhancing the FGS attack [55] with EG-Booster . In this example, FGS perturbations originally failed to fool the model to predict an incorrect label (i.e., $y \neq$ ‘7’). The image on the right shows the perturbations (circled in green) added by EG-Booster that are consequential enough to fool the model into predicting the input image as ‘9’ instead of ‘7’.

4.2.4 Stability Analysis

When evaluated on security-sensitive learning tasks (e.g., malware classifiers), ML explanation methods have been found to exhibit output *instability* whereby explanation weights of the same input sample x vary from one run to another [132, 50]. To ensure that EG-Booster does not inherit the instability limitation of the ML explainer, we perform stability analysis where we compare the *similarity* of the prediction results returned by the target model f after performing an explanation-guided attack over multiple runs. To do so, we define the *k-stability* metric that quantifies the stability of EG-Booster across *k* distinct runs. It measures the average run-wise (e.g., each pair of runs) *similarity* between the returned predictions of the same adversarial sample after EG-Booster attack. The *similarity* between two runs i and j, $similarity(i, j)$, is the intersection size of the predictions returned by the two runs over all samples in the test set (i.e., the number of matching predictions), and is computed as:

$$\textit{k-stability} = \frac{1}{k(k-1)} \sum_{i=1}^{k} \sum_{\substack{j=1 \\ j \neq i}}^{k} similarity(i, j) \tag{4.3}$$

The instability of explanation methods is mainly observed in the minor differences between the magnitude of the returned explanation weights of each feature $||w_i||$ which might sometimes lead to different feature ranking. However, the direction of a feature x_i is less likely to change from one run to another as it is uniquely decided by the sign of its explanation weight $sign(w_i)$. This is mainly visible on images as it is possible to plot feature directions with different colors on top of the original image. EG-Booster relies only on the sign of the explanation weights $w_i \in W_{x,y_{true}}$ to detect *non-consequential perturbations* and candidates for *consequential perturbations*. As a result, the output of EG-Booster is expected to be more stable than output of the underlying explanation method.

To validate our intuition, we compare the stability of EG-Booster with the stability of the employed explanation method. Thus, we compute the *(k,l)-Stability* metric proposed by a prior work [50] that evaluates the stability of ML explanation methods for malware classification models. The *(k,l)-Stability* measure computes the average explanation similarity based on intersection of features that are ranked in the top-l, returned by separate k runs of an explanation method. More precisely, given two explanation results, $W_{x,y_{true}}(l, i)$ and $W_{x,y_{true}}(l, j)$ returned by two different runs i and j, and a parameter l, their similarity is obtained based on the Dice coefficient as:

$$\textit{sim}(W_{x,y_{true}}(l,i), W_{x,y_{true}}(l,j)) = 2 * \frac{W_{x,y_{true}}(l,i) \cap W_{x,y_{true}}(l,j)}{|W_{x,y_{true}}(l,i)| + |W_{x,y_{true}}(l,j)|}, \tag{4.4}$$

where $W_{x,y_{true}}(l, i)$ denotes the top-l features of sample x with respect to the prediction y_{true}

returned by run i. Over a total of k-runs, the average *(k,l)-Stability* on a sample x is defined as:

$$\textit{(k,l)-stability} = \frac{1}{k(k-1)} \sum_{i=1}^{k} \sum_{\substack{j=1 \\ j \neq i}}^{k} sim(W_{x,y_{true}}(l,i), W_{x,y_{true}}(l,j)) \tag{4.5}$$

More empirical discussions about the stability results over all samples in a test set X_t are presented in Section 4.3.4.

4.3 Evaluation

We now present the evaluation of EG-Booster . We first describe our experimental setup in Section 5.3.1. In Section 4.3.2, we present details of EG-Booster evasion accuracy effectiveness. In Sections 4.3.3, 4.3.4, and 4.3.5, we present our findings on perturbation change, stability analysis, and execution time of EG-Booster , respectively.

4.3.1 Experimental Setup

Datasets. We evaluate EG-Booster on two benchmark datasets used for adversarial robustness evaluation of deep neural networks: MNIST [80] —a handwritten digit recognition task with ten class labels $(0 - 9)$ and CIFAR10[73] —an image classification task with ten classes. We use the 10K test samples of both datasets to evaluate the performance of EG-Booster on undefended models (described next). To evaluate EG-Booster against a defended model, we use 5K samples of the CIFAR10 test set.

Models. For MNIST, we train a state of the art 7-layer CNN model from "Model Zoo"[1], provided by the SecML library [91]. It is composed of 3-conv layers+ReLU, 1-Flatten layer, 1-fully-connected layer+ReLU, a dropout-layer (p=0.5), and finally a Flatten-layer. We call this model **MNIST-CNN**. It reaches a test accuracy of 98.32% over all 10K test samples.

For CIFAR10, we consider two different models: a CNN model [4] with 4-conv2D and a benchmark adversarially-trained ResNet50 model. In our experimental results, we call the undefended model **CIFAR10-CNN** and the defended model **CIFAR10-ResNet**. More precisely, the structure of CIFAR10-CNN is: 2-conv2D+ ReLU, 1-MaxPool2D, 1-dropout(p=0.25), 2-conv2D+ReLU, 1-MaxPool2D, 1-dropout(p=0.25), 2-fully-connected+ReLU, 1-dropout(p=0.25), and 1-fully-connected. To avoid the reduction of height and width of the images, padding is used in every convolutional layer. This model reaches a test accuracy of 86.41% after 50 epochs. It outperforms the benchmark model adopted by Carlini & Wagner (2017)[35] and Papernot et al. (2016) [94].

[1]https://gitlab.com/secml/secml/-/tree/master/src/secml/model_zoo

CIFAR10-ResNet is a state-of-the-art adversarially-trained model from the "robustnes" library [48] proposed by MadryLab[2]. It is trained on images generated with PGD attack using L_2 norm and $\epsilon = 0.5$ as perturbation bound. It reaches a test accuracy of 92.36% over all 10K test set of CIFAR10.

Baseline Attacks. We evaluate EG-Booster on 4 *white-box* baseline attacks (FGS [55], BIM [79], PGD [89], and C&W [35]) and 3 *black-box* attacks (MIM [47], HSJA [36], and SPSA [128]). Detailed explanations of all the 7 baseline attacks are provided in Chapter 2.

ML Explainer. The performance of EG-Booster is influenced by the effectiveness of the employed ML explanation method in detecting the *direction* of each feature x_i in a sample x. It is, therefore, crucial to suitably choose the ML explainer according to the studied systems and the deployed models. In our experiments, we focus on image datasets and neural network models, therefore, we pick SHAP [87], as it is proved to be effective in explaining deep neural networks, especially in the image classification domain. Furthermore, SHAP authors proposed a ML explainer called "Deep Explainer", designed for deep learning models, specifically for image classification. SHAP has no access to the target model, which makes EG-Booster suitable for either black-box or white-box threat model. We note that independent recent studies [132, 50] evaluated ML explanation methods for malware classifiers. They revealed that LIME [101] outperforms other approaches in security systems in terms of accuracy and stability. Thus, we recommend using LIME for future deployment of EG-Booster for robustness evaluation of ML malware detectors.

Evaluation metrics. We use the following four metrics to evaluate EG-Booster :

Evasion Rate: First, we quantify the effectiveness of EG-Booster by monitoring changes to the *percentage of successful evasions* with respect to the total test set, from the baseline attacks to EG-Booster attacks.

Average Perturbation Change: Second, we keep track of the average changes to the number of perturbed features for baseline attack versus EG-Booster attacks to show the impact of the addition or elimination of perturbations performed by EG-Booster . Referring to Algorithm 4.1 (lines 23 and 42), EG-Booster keeps track of the number of added perturbations (N_c) and the number of eliminated perturbations (N_{nc}) per-image. Thus, the total of perturbation changes per-image is $N_c - N_{nc}$, and the average perturbation change across all images of the test set (X_t) is:

$$\textit{Average Perturbation Change} = \frac{1}{|X_t|} \sum_{x \in X_t} \frac{N_c(x) - N_{nc}(x)}{N_{baseline}(x)}, \tag{4.6}$$

where $N_{baseline}(x)$ is the total number of perturbations initially performed by the baseline attack on sample x. When *Average Perturbation Change*> 0, on average, EG-Booster is performing more perturbation additions than eliminations ($N_c > N_{nc}$). Otherwise, it is performing more elimination

[2]https://github.com/MadryLab/robustness/blob/master/robustness/cifar_models/ResNet.py

Baseline Attacks	Undefended MNIST-CNN					
	White-Box					Black-Box
	FGS ($X_t = 10K$)	*BIM* ($X_t = 10K$)	*PGD* ($X_t = 10K$)		*C&W* ($X_t = 5K$)	*MIM* ($X_t = 10K$)
$\|\|.\|\|_p$ **Norms**	L_∞	L_∞	L_2	L_∞	L_2	L_∞
Initial Evasion Rate	28.20%	49.92%	99.80 %	55.09%	99.14%	41.34%
EG-Booster Evasion Rate	**89.71%**	**89.08%**	**99.89%**	**88.69%**	**99.72%**	**79.08%**
Average Perturbation Change	+14.44%	+14.47%	**-5.46%**	**-4.39%**	**-15.36%**	**-5.49%**

Table 4.1: Summary of EG-Booster results on undefended MNIST-CNN Model ($\epsilon = 0.3$).

Baseline Attacks	Undefended CIFAR10-CNN					
	White-Box					Black-Box
	FGS ($X_t = 10K$)		*PGD* ($X_t = 10K$)		*C&W*($X_t = 5K$)	*HSJA* ($X_t = 10K$)
$\|\|.\|\|_p$ **Norms**	L_2	L_∞	L_2	L_∞	L_2	L_∞
Initial Evasion Rate	22.36%	83.04%	22.46%	91.07%	99.22%	14.04%
EG-Booster Evasion Rate	**30.14%**	**87.45%**	**31.12%**	**92.34%**	99.22%	**28.87%**
Average Perturbation Change	**-48.52%**	**-49.62%**	**-48.25%**	**-51.15%**	**-52.16%**	**-36.18%**

Table 4.2: Summary of EG-Booster results on undefended CIFAR10-CNN Model ($\epsilon = 0.3$).

of non-consequential perturbations ($N_c < N_{nc}$). The average becomes zero when there are equal number of added and eliminated perturbations.

k-Stability and ***(k,l)-Stability***: To evaluate the reliability of EG-Booster , we use the $k-Stability$ (Equation 4.3) and $(k,l)-Stability$ (Equation 4.5) measures we introduced in Section 4.2.4. We compute the *k-Stability* of EG-Booster for different k values and we compare it to the value of the *(k,l)-Stability* of the employed explanation method (i.e., SHAP), using the top-10 features ($l = 10$) and different values of k. Both metrics are calculated in average over 1000 samples.

Execution Time: Across both datasets and all models, we measure the per-image execution time (in seconds) taken by EG-Booster for different baseline attacks.

4.3.2 EG-Booster Evasion Effectiveness

Tables 4.1, 4.2, and 4.3 report results of EG-Booster on different models: undefended MNIST-CNN, undefended CIFAR10-CNN, and defended (adversarially-trained) CIFAR10-ResNet, respectively.

Evasion rate in a nutshell: Across the three tables, we observe a significant increase in the evasion rate of studied baseline attacks after combining them with EG-Booster . For instance, Table 4.1 shows an average increase of 28.78% of the evasion rate across all attacks performed on the undefended MNIST-CNN model. Similarly, the evasion rate results in Table 4.2 convey that, across

	Defended CIFAR10-ResNet					
Baseline Attacks	White-Box					Black-Box
	FGS ($X_t = 5K$)		*PGD* ($X_t = 5K$)		*C&W* ($X_t = 1K$)	*SPSA* ($X_t = 1K$)
$\|\|.\|\|_p$ **Norms**	L_2	L_∞	L_2	L_∞	L_2	L_∞
Initial Evasion Rate	9.75%	73.27%	9.66%	86.25%	99.00%	10.80%
EG-Booster Evasion Rate	**18.05%**	**74.73%**	**18.37%**	**86.76%**	**99.75%**	**23.46%**
Average Perturbation Change	**-41.23%**	**-45.04%**	**-41.16%**	**-46.51%**	**-51.20%**	**-42.11%**

Table 4.3: Summary of EG-Booster results on defended (adversarially-trained) CIFAR10-ResNet50 Model ($\epsilon = 0.3$).

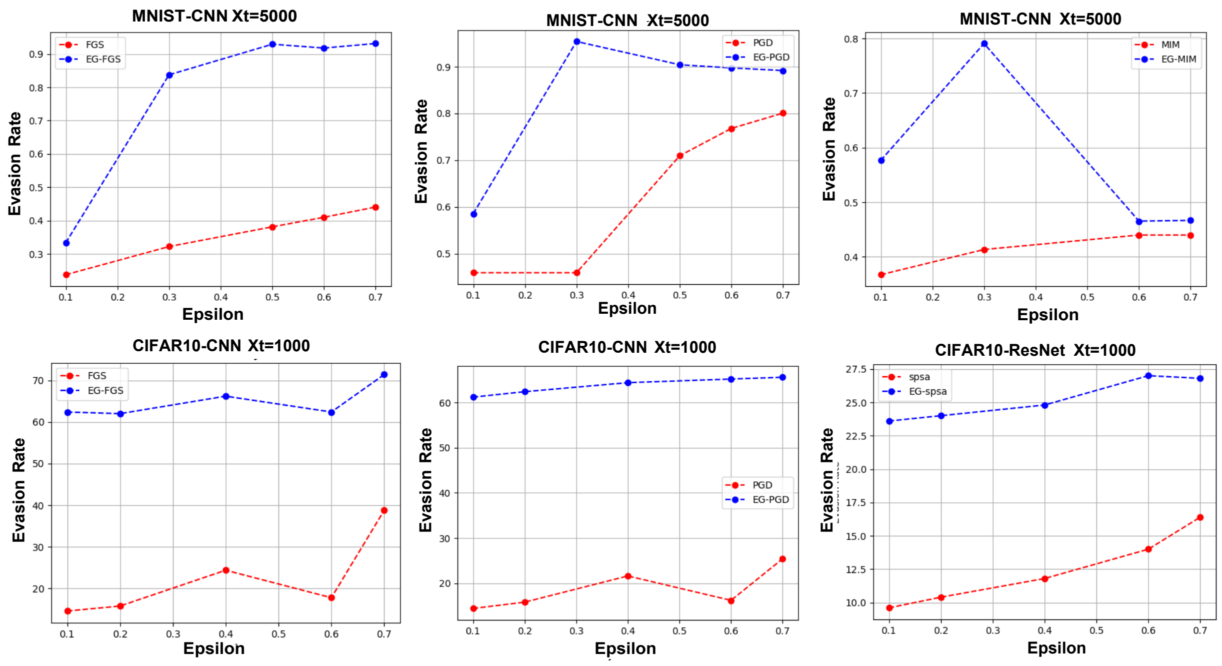


Figure 4.4: The evasion rate curve of baseline attacks against EG-Booster attacks across different perturbation bounds ϵ, using $||.||_\infty$ for MNIST and $||.||_2$ for CIFAR10 as distance metrics.

all studied baseline attacks, an average of 6.23% more adversarial images are produced by EG-Booster to evade the undefended CIFAR10-CNN model. Same observations can be drawn from Table 4.3. More precisely, overall baseline attacks performed on the adversarially-trained CIFAR-ResNet model, we observe an average of 5.41% increase in the evasion rate, when combined with EG-Booster . In a nutshell, our findings consistently suggest that explanation methods can be employed to effectively guide evasion attacks towards higher evasion accuracy.

EG-Booster is consistently effective across model architectures, threat models, and distance metrics: Our results consistently suggest that EG-Booster is agnostic to model architecture, threat model, and supports diverse distance metrics. For instance, the increase in evasion rate is observed on white-box baseline attacks (i.e., FGS, BIM, PGD, and C&W) as well as for black-box attacks (i.e., MIM, HSJA, and SPSA). Additionally, the improvement in baseline attacks performance is observed for different $||.||_p$ norms. For the same perturbation bound value $\epsilon = 0.3$ and the same attack strategy (e.g., FGS, PGD), we notice an increase in the evasion rate regardless of the employed $||.||_p$ distance metric.

EG-Booster is consistently effective over a range of perturbation bounds: We additionally assess EG-Booster for different ϵ values. Figure 4.4 reports the evasion rate curve of the state-of-the-art attacks before and after being guided with EG-Booster (see EG-PGD, EG-MIM, etc in Figure 4.4), using different perturbation bounds $\epsilon = 0.1 \rightarrow 0.7$. For all target models trained on MNIST and CIFAR10, we observe a significant improvement in the evasion rate of all baseline attacks regardless of ϵ. However, for most attacks, we observe that, a higher perturbation bound ϵ which allows a greater perturbation size can lead to a higher evasion rate.

It is noteworthy that the increase in rate of evasion results is considerably higher for baseline attacks that initially have a low evasion rate. For instance, $\textit{FGS}_{L_\infty}$ and $\textit{BIM}_{L_\infty}$ that initially produced, respectively, 28.20% and 49.92% evasive adversarial samples from the total of 10K MNIST samples, have improved at least twofold after employing EG-Booster . Similar observations are drawn from CIFAR10-CNN and CIFAR10-ResNet. Particularly, for CIFAR10-CNN, the increase rate of evasion results on $\textit{FGS}_{L_2}(22.36\% \rightarrow 30.14\%)$ is higher than the increase rate of $\textit{FGS}_{L_\infty}$ $(83.04\% \rightarrow 87.45\%)$. However, even though EG-Booster results in higher increase rate in evasive samples for less performing attacks, we note that, the total evasion rate of EG-Booster combined with stronger attacks is more important. This is expected, as EG-Booster builds on the initial results of baseline attacks which result in a correlation between the initial evasion rate of a baseline attack and the post-EG-Booster evasion rate. This is mainly observed for the C&W attack, as across all models, combined with C&W, EG-Booster exhibits the highest evasion rates (i.e., $> 99\%$). Such correlation is also evident in the evasion rate curves from Figure 4.4.

EG-Booster maintains its effectiveness against defended models: Table 4.3 shows effectiveness of EG-Booster against a defended model, compared to adversarial training defense. Since PGD_{L_2} is used for adversarial training, CIFAR10-ResNet is expected to be specifically robust against gradient-based attacks performed using L_2 norm. Incontrovertibly, FGS_{L_2} and PGD_{L_2} achieve considerably lower initial evasion rates on CIFAR10-ResNet ($\sim 9\%$), compared to undefended models ($\sim 20\%$). Nevertheless, combined with EG-Booster , FGS_{L_2} and PGD_{L_2} result in more evasive samples ($\sim 18\%$), even against a defended model. Same findings are observed for other baseline attacks (e.g., FGS_{L_∞}, $SPSA_{L_\infty}$). From these observations, we conclude that EG-Booster is still effective in the face of defended models.

4.3.3 Perturbation Change

In addition to *Evasion Rate*, for each experiment, we keep track of *Average Perturbation Change* introduced by EG-Booster (using Equation 4.6). Results from the last rows of Tables 4.1, 4.2, and 4.3 show that for most baseline attacks, EG-Booster performs less perturbations while improving the evasion rate. This is explained by the negative sign of the average perturbation change for most of the studied baseline attacks across the three tables. These findings prove that, without considering the pre-perturbation feature direction explanations, baseline attacks are initially performing a considerable number of non-consequential (unnecessary) perturbations. Consequently, in addition to the improvement of the evasion rate, these results demonstrate the importance of taking into account feature explanation weights in the formulation of evasion attacks (as shown in Equation 4.2).

It is noteworthy that the magnitude of the negative average perturbation change values is specifically more important for baseline attacks that initially have a high evasion rate. This is mainly true for the C&W attack across the 3 tables and PGD_{L_∞} in Table 4.2. We explain this observation by the fact that, EG-Booster performs additional perturbations only when the baseline adversarial sample originally fails to evade the model (Algorithm 4.1: line 27). Otherwise, it only eliminates non-consequential perturbations while maintaining the original evasion result.

In some of the experiments, we notice a positive value of the average perturbation change. This is particularly the case for FGS_{L_∞} and BIM_{L_∞} in Table 4.1. In this cases, EG-Booster performs more additional perturbations than eliminations which reflects the drastic improvement of the evasion rate for these two attacks. These findings particularly demonstrate the direct impact of perturbing *positive* features.

Focusing on Tables 4.1 and 4.2, we notice that, on average, the increase rate in evasion results for MNIST (28.78%) is higher than CIFAR-10 (6.23%) using a CNN model for both datasets. Additionally, the average perturbation change for all experiments in Table 4.2 (i.e., CIFAR10-CNN) is negative and have a higher magnitude than their counterparts in Table 4.1 (MNIST-CNN). Further investigations have shown that these two observations are related. We found that baseline attacks performed on CIFAR10-CNN are already perturbing a considerable number of features that are directed to the true label which makes the rate of added perturbations by EG-Booster in CIFAR10-CNN lower than the ones added in MNIST-CNN. This observation might explain the difference in evasion increase between both datasets. However, as discussed in Section 7.2, EG-Booster specifically examines this case i.e., the case where an important number of detected consequential features turn out to be already perturbed. More precisely, in case of initial evasion failure, EG-Booster proceeds by additionally perturbing features that are already perturbed by the baseline attack while ensuring that the perturbation size δ is still within the bound ($||\delta||_p < \epsilon$).

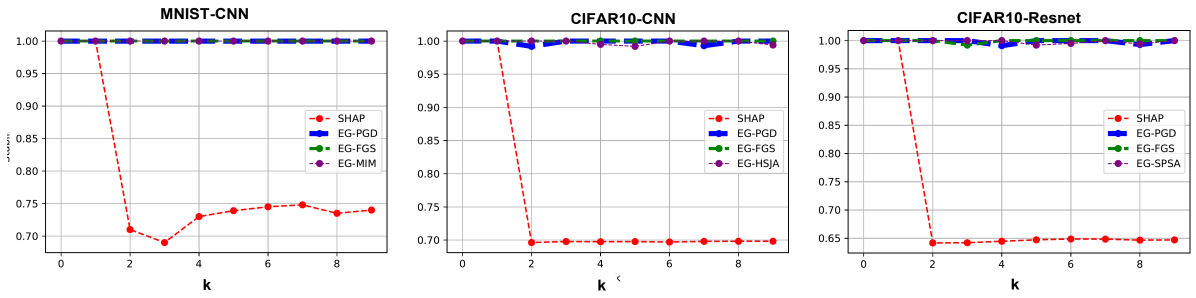


Figure 4.5: Comparison between the stability of EG-Booster and the employed explanation method, SHAP, across models and datasets, for different k values. $(k, l) - Stability$ of SHAP is computed on the top-10 features ($l = 10$). EG-Booster is performed using different baseline attacks subject to $||\delta||_\infty < 0.3$. $k-$runs are performed on 1000 test samples for each attack.

4.3.4 Stability Analysis

As highlighted by prior works [132, 50] and discussed in this chapter, due to their stability concern, ML explanation methods should be carefully adopted in security-critical settings. Thus, we evaluate the reliability of explanation methods to be employed for the robustness assessment of ML systems via a comparative analysis between the stability of the employed ML explainer (i.e., SHAP) and the stability of our explanation-based method. We use the *(k-l)-Stability* and *k-Stability* metrics defined in Section 4.2.4, to respectively compute the output's stability of SHAP, and EG-Booster combined with baseline attacks, across different studied models.

EG-Booster doesn't inherit instabilities of SHAP: In Figure 4.5, we plot stability analysis results. Focusing on the stability curves of EG-Booster , we deduce that, it is almost 100% stable for all studied models and across different baseline attacks. Consequently, the classification results returned by the target ML model are the same across different runs of EG-Booster . Such findings prove the reliability of EG-Booster to adopt it for improved robustness assessment of ML models against baseline attacks. Compared to the stability curves of SHAP, EG-Booster does not inherit the instability concern indicated by ML explainer's output. As discussed in Section 4.2.4, these findings are explained by the reliance only on the sign of explanation weights to decide the feature directions (i.e., *positive* and *negative* features). Since the distortions across different runs of the feature directions results returned by an accurate ML explainer are minor, the feature selection for perturbation performed by EG-Booster returns the same feature set across different runs which overall leads to the same evasion result.

Although EG-Booster is not deeply influenced by the *stability* of the employed ML explainer, its success is, however, relying on their *accuracy* to produce precise explanations, specifically precise feature directions in our case. Given the promising results, that our approach showed when relying on SHAP, we hope that future improvements in ML explanation methods would lead to an even better performance of EG-Booster .

4.3.5 Execution Time

Our measurements are based on a hardware environment with 6 vCPUs, 18.5GB memory, and 1 GPU-NVIDIA Tesla K80.

We recall that EG-Booster is performed on top of the baseline attack. Therefore, the execution time of EG-Booster is the sum of the time taken to craft the baseline attack and EG-Booster to improve the baseline attack. Consequently, the execution time of the baseline attack (e.g., FGSM, C&W) has a direct influence on the execution time of EG-Booster . In Figure 4.6, we plot the range of execution times of EG-Booster recorded by different experiments (i.e., on top of different baseline attacks). Over all studied attacks, *baseline + EG-Booster* take, on average, $0.55s$ for MNIST-CNN, $27.0s$ for CIFAR10-CNN, and $82.0s$ for CIFAR10-ResNet per-sample (orange line). Overall, these observations reflect the efficiency of EG-Booster . However, they also reveal the impact of the input dimension and the model architecture on the running time of EG-Booster . For instance, it takes more time on CIFAR10, compared to MNIST, which is due to the difference in input dimension between the datasets i.e., (28x28x1) for MNIST and (32x32x3) for CIFAR10. Additionally, for CIFAR10, we observe a considerable difference in execution time of EG-Booster between CNN and ResNet50 (i.e., $27.0s \rightarrow 82.0s$) which is due to the higher dimension of ResNet50 compared to CNN. From the performance overhead, EG-Booster is specifically useful when used on top of *fast* and *weak* baseline attacks (e.g., FGSM). However, using EG-Booster on strong attacks that usually take long time to converge and reach high evasion rate (e.g., C&W) may result in higher overhead although our experiments (Tables 4.1, 4.2 and 4.3) showed that EG-Booster improves even strong attacks such as C&W.

4.4 Conclusion

In this chapter, we introduce EG-Booster , the first explanation-guided booster for ML evasion attacks. Guided by feature-based explanations of model predictions, EG-Booster significantly improves evasion accuracy of state-of-the-art adversarial example crafting via introduction of consequential perturbations and elimination of non-consequential ones. Extensive evaluation of EG-Booster on MNIST and CIFAR10, across white-box and black-box attacks against undefended and defended models shows that EG-Booster significantly improves evasion accuracy of reference evasion attacks on undefended and adversarially-trained models. We also empirically show the reliability of explanation methods to be adopted for robustness assessment of ML models. We

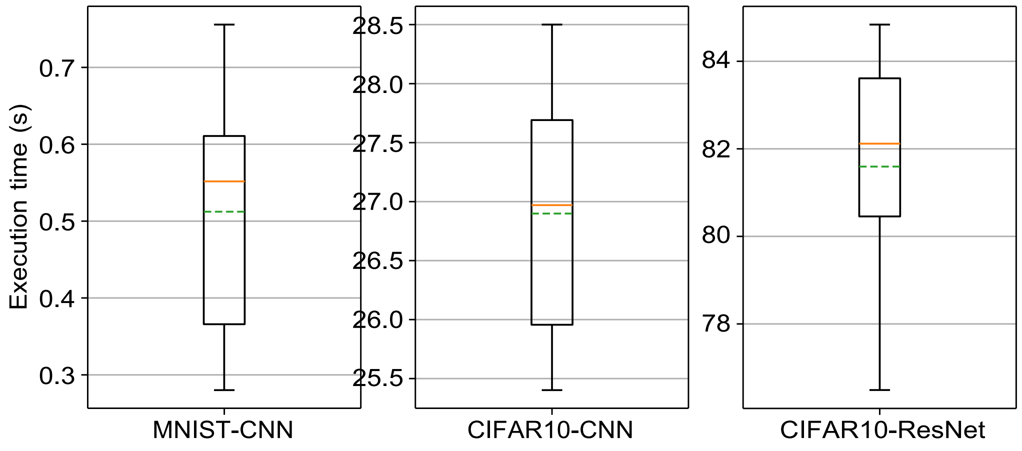


Figure 4.6: EG-Booster performance overhead. Execution time(s) =*Time of baseline attack + time of Morphence-2.0 perturbation check*), per-sample, across the studied models.

hope that EG-Booster will be used by future work as a benchmark for robustness evaluation of ML models against adversarial examples.

CHAPTER 5

Morphence: Moving Target Defense against Adversarial Examples

5.1 Introduction

Machine learning (ML) continues to propel a broad range of applications in image classification [75], voice recognition [41], precision medicine [51], malware/intrusion detection [99], autonomous vehicles [105], and so much more. ML models are, however, vulnerable to *adversarial examples* —minimally perturbed legitimate inputs that fool models to make incorrect predictions [55, 27]. Given an input x (e.g., an image) correctly classified by a model f, an adversary performs a small perturbation δ and obtains $x' = x + \delta$ that is indistinguishable from x to a human analyst, yet the model misclassifies x'. Adversarial examples pose realistic threats on domains such as self-driving cars, healthcare, and malware detection for the consequences of incorrect predictions are highly likely to cause real harm [49, 72, 63].

To defend against adversarial examples, previous work took multiple directions each with its pros and cons. Early attempts [57, 65] to harden ML models provided only marginal robustness improvements. Heuristic defenses based on *defensive distillation* [94], *data transformation* [42, 25, 141, 135, 88, 58], and *gradient masking* [31, 119] were subsequently broken [34, 61, 22, 35].

While *adversarial training* [55, 126] defends against known attacks, robustness comes at the expense of accuracy loss on clean data. Similarly, data transformation-based defenses also degrade accuracy on benign inputs. *Certified defenses* [81, 37, 82] provide formal robustness guarantee, but are limited to a class of attacks constrained to LP-norms [81, 134].

As pointed out by [53], a shared limitation of prior defenses is the *static and fixed target* nature of the deployed ML model. We argue that, although defended by methods such as adversarial training, the fact that a ML model is a *fixed target* that continuously responds to prediction queries makes it *a prime target for repeated/correlated adversarial attacks*. As a result, given enough time, an adversary can repeatedly query the prediction API and build enough knowledge about the ML model and eventually fool it. Once the adversary launches a successful attack, it will be always effective since the model is not moving from its compromised "location".

In this article, we introduce two releases of a moving target defense called Morphence. Morphence-2.0 builds on the MTD of Morphence-1.0 [13] in three ways: enhanced approach, more comprehensive experimental evaluations, and new insights. To differentiate advances in Morphence-2.0 , next we first briefly summarize Morphence-1.0 and then describe values added by Morphence-2.0
.

Morphence-1.0 [13]. By regularly moving the decision function of a model, Morphence-1.0 makes it challenging for an adversary to fool the model through adversarial examples. Morphence-1.0 thwarts once successful and repeated attacks and attacks that succeed after iterative probing of a fixed target model through correlated sequence of attack queries. To do so, Morphence-1.0 deploys a pool of n models generated from a base model in a manner that introduces sufficient randomness when it selects the most suitable model to respond to prediction queries. The selection of the *most suitable model* is governed by a scheduling strategy that relies on the prediction confidence of each model on a given query input. To ensure repeated or correlated attacks fail, the deployed pool of n models automatically expires after a query budget is reached. The model pool is then seamlessly replaced by a new pool of n models generated and queued in advance. To be practical, Morphence-1.0 aims to improve robustness to adversarial examples across white-box and black-box attacks (*Challenge-1*); maintain accuracy on benign samples as close to that of the base model as possible (*Challenge-2*); and increase diversity among models in the pool to reduce adversarial example transferability among them (*Challenge-3*). It addresses *Challenge-1* by enhancing the MTD aspect through larger model pool size, a model selection scheduler, and dynamic pool renewal (Sections 5.2.2 and 5.2.4). *Challenge-2* is addressed by re-training each generated model to regain accuracy loss caused by perturbations (Section 5.2.2: step-2). It addresses *Challenge-3* by making the individual models distant enough via distinct transformed training data used to re-train each model (Section 5.2.2: step-2). Training a subset of the generated models on distinct adversarial data is an additional robustness boost to address *Challenges 1 and 3* (Section 5.2.2: step-3). While Morphence-1.0 made significant advances on MTD-based prior work [118, 110, 98] (see Chapter 9), in this work we significantly enhance its scheduling strategy, extend experimental evaluations, and draw novel insights in the context of MTD against adversarial examples.

Morphence-2.0 . We significantly overhaul the scheduling strategy in Morphence-1.0 by introducing a layer of OOD detection that further guides the model selection strategy with respect to the nature of the received query (i.e., adversarial vs. benign). To this end, we draw insights from [15] that makes empirical observations that suggest most adversarial examples are OOD samples. In particular, adversarial perturbations usually shift the distribution of the perturbed sample $x' = x + \delta$ away from its initial distribution $\mathbb{P}_{train}$ used in training the target model.

Consequently, we leverage and adapt OOD detection [109] for the sake of adversarial examples detection (details in Section 5.2.3).

We evaluate Morphence-2.0 on two benchmark image classification datasets (MNIST and CIFAR10) against four attacks: three white-box attacks (FGSM [55], PGD [89], and C&W [35]) and one iterative black-box attack (SPSA [128]). We compare Morphence-2.0 's robustness with Morphence-1.0 and adversarial training defense of a fixed model. We then conduct detailed evaluations on the impact of the MTD strategy in defending previously successful repeated attacks and the effectiveness of OOD detection to boost the scheduling strategy. Additionally, through extensive experiments, we shed light on each component of Morphence-2.0 and its impact on improving the robustness results and reducing the transferability rate across models. Overall, our evaluations suggest that Morphence-2.0 advances the state-of-the-art in robustness against adversarial examples, even in the face of strong white-box attacks such as C&W [35], while maintaining accuracy on clean data and reducing attack transferability. In summary, this work builds on Morphence-1.0 and makes the following contributions:

- Powered by OOD detection, Morphence-2.0 is able to precisely select the most accurate decision function for each query.
- Morphence-2.0 thwarts repeated attacks that leverage previously successful attacks and correlated attacks performed through dependent consecutive queries.
- Morphence-2.0 outperforms adversarial training and Morphence-1.0 while preserving accuracy on clean data and reducing attack transferability within a model pool.
- Morphence-2.0 improves the state-of-the-art defenses on both white-box and black-box attacks.
- Morphence-2.0 code is available as free and open-source software at: `https://github.com/um-dsp/Morphence`.

5.1.1 Background -SSD: Out-of-Distribution Detector

SSD [109] is proposed to detect OOD data that lies far away from the training distribution $\mathbb{P}_{train}$ of a ML model f. Its appealing side is that it is a self-supervised method that can reach good performance using only unlabeled data instead of fine-grained labeled data that can be hard to produce. Given unlabeled training data, SSD leverages *Contrastive self-supervised representation learning*, which aims to train a feature extractor of the in-distribution data, by discriminating between individual samples, to learn a good set of representation without the need to data labels [109]. Next, the OOD detection is performed through a cluster-conditioned detection. Using k-means clustering,

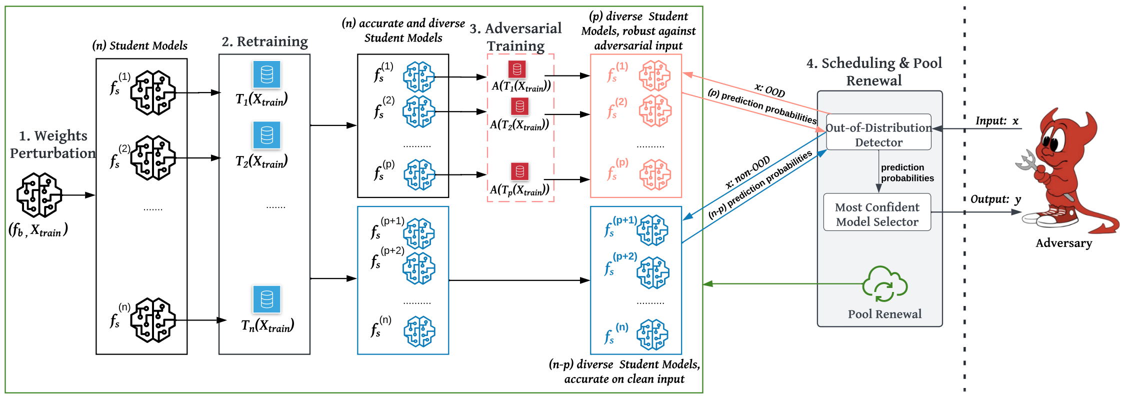

1. Weights Perturbation
(f_b, X_{train})
(n) Student Models
2. Retraining
$T_1(X_{train})$
$T_2(X_{train})$
$T_n(X_{train})$
(n) accurate and diverse Student Models
3. Adversarial Training
$A(T_1(X_{train}))$
$A(T_2(X_{train}))$
$A(T_p(X_{train}))$
(p) diverse Student Models, robust against adversarial input
(n-p) diverse Student Models, accurate on clean input
4. Scheduling & Pool Renewal
x: OOD
(p) prediction probabilities
x: non-OOD
(n-p) prediction probabilities
Out-of-Distribution Detector
prediction probabilities
Most Confident Model Selector
Pool Renewal
Input: x
Output: y
Adversary

Figure 5.1: Morphence-2.0 system overview illustrated with model pool generation (1-3) and scheduling and model pool renewal (4).

extracted features of in-distribution data are partitioned into m clusters. Features of each cluster are modeled independently to calculate an outlier score of an input x, using the Mahalanobis distance $M(x, \mathbb{P}_{train})$. It is equivalent to the euclidean distance, but scaled with eigenvalues in the eigenspace. SSD discriminates between in-distribution (e.g CIFAR-10) and OOD (e.g CIFAR-100) data along each principal eigenvector. With euclidean distance (i.e. in absence of scaling), components with higher eigenvalues have more weights but provide least discrimination. Scaling with eigenvalues removes the bias. In other words, $M(x, \mathbb{P}_{train})$ is more effective for outlier detection in the feature space. More details about the choice of the distance metric and the background of the Contrastive self-supervised learning can be retrieved in the cited paper [109]. More importantly, we choose SSD for our approach, given that it outperforms all other OOD detection tools, at the time of the submission of this article.

5.2 Approach

We first describe Morphence-2.0 at a high level in Section 5.2.1 and then dive deeper into details in Sections 5.2.2 - 5.2.4.

5.2.1 Overview

Figure 5.1 illustrates Morphence-2.0 . The key intuition in Morphence-2.0 is making a model a moving target in the face of adversarial example attacks. To do so, Morphence-2.0 deploys a pool of n models instead of a single target model. We call each model in pool a *student model*. The model pool is generated from a base model f_b in a manner that introduces sufficient randomness when Morphence-2.0 responds to prediction queries, without sacrificing accuracy of f_b on clean inputs. In response to each prediction query, Morphence-2.0 selects the *most suitable model* to predict the label for the input. The selection of the *most suitable model* is governed by a scheduling strategy that relies on the prediction confidence of each model on a given input. The deployed pool of n models automatically expires after a query budget Q_{max} is exhausted. An expired model pool is seamlessly replaced by a new pool of n models generated and queued in advance. As a result, repeated/correlated attacks fail due to the moving target nature of the model. In the following, we use Figure 5.1 to highlight the core components of Morphence-2.0 focusing on *model pool generation*, *scheduling*, and *model pool renewal*.

Model Pool Generation. The model pool generation method is composed of three main steps (1-3 in Figure 5.1). Each step aims to tackle one or more of challenges 1–3. As shown in Figure 5.1, a highly accurate f_b initially trained on a training set X_{train} is the foundation from which n models that make up the Morphence-2.0 student model pool are generated. Each student model is initially obtained by slightly perturbing the parameters of f_b (Step-1, *weights perturbation*, in Figure 5.1). By introducing different and random perturbations to model parameters, we *move* f_b's decision function into n different *locations* in the prediction space.

Given the randomness of the weights perturbation, Step-1 is likely to result in inaccurate student models compared to f_b. Simply using such inaccurate models as part of the pool penalizes prediction accuracy on clean inputs. As a result, we introduce Step-2 to address *Challenge-2* by *re-training* the student models so as to boost their accuracy and bring it close enough to f_b's accuracy. However, since the n models are generated from the same f_b, their transferability rate of adversarial examples is usually high. To reduce evasion transferability among student models, we use distinct training sets for each student model (Step-2: *retraining* in Figure 5.1). For this purpose, we harness data transformation techniques (i.e., image transformations) to produce n distinct training sets. In Section 5.3.5, we empirically explore to what extent this measure reduces the transferability rate and addresses *Challenge-3*. Finally, a subset of p student models ($p < n$) is adversarially trained (Step-3: *Adversarial training* in Figure 5.1) as a reinforcement to the MTD core of Morphence-2.0 , which addresses *Challenge-1*. In Section 5.2.2, we explain the motivations and details of each step.

Scheduling and Pool Renewal. Instead of randomly selecting a model from the model pool or taking the majority vote of the n models (as in [118]), for a given input query Morphence-2.0 returns the prediction of the *most confident model*. The motivation to pick the most confident model is twofold. First, the sufficient diversity among the n models, where a subset of the models is pre-hardened with adversarial training (hence perform much better on adversarial inputs) and the remaining models are trained to perform more confidently on legitimate inputs. Second, the routing of an input to either adversarially trained p models or the remaining $n - p$ models is powered by an OOD detection component (Step 4 in Figure 5.1).
The model pool is automatically renewed after a defender-set query upper-bound Q_{max} is reached. The choice of Q_{max} requires careful consideration of the model pool size (n) and the time it takes

to generate a new model pool while Morphence-2.0 is serving prediction queries on an active model pool. In Section 5.2.4, we describe the details of the model pool renewal with respect to Q_{max}.

5.2.2 Student Model Pool Generation

Using Algorithms 1 and 2, we now describe the details of steps 1–3 with respect to challenges 1–3.

Step-1: Model Weights Perturbation. Shown in Step-1 of Figure 5.1, the first step to transform the fixed model f_b into a moving target is the generation of multiple instances of f_b. To effectively serve the MTD purpose, the generated instances of f_b need to fulfill two conditions. First, they need to be *sufficiently diverse* to reduce attack transferability among themselves. Second, they need to preserve the accuracy of f_b.

By applying n different small perturbations on the model weights θ_b of f_b, we generate n variations of f_b as $f_s = \{f_s^{(1)}, f_s^{(2)}, ..., f_s^{(n)}\}$, and we call each $f_s^{(i)}$ a *student model*. The n perturbations should be sufficient to produce n diverse models that are different from f_b. More precisely, higher perturbations lead to a larger distance between the initial f_b and a student model $f_s^{(i)}$, which additionally contribute to greater movement of the decision function of f_b. However, the n perturbations are constrained by the need to preserve the prediction accuracy of f_b for each student model $f_s^{(i)}$. As shown in Algorithm 1 (line 2), we perturb the parameters θ_b by adding noise sampled from the *Laplace* distribution.

Laplace distribution is defined as $\frac{1}{2\lambda}\exp(-\frac{|\theta_b-\mu|}{\lambda})$ [107]. The center of the post-perturbation weights distribution is the original weights θ_b. We fix the mean value of the added Laplace noise as $\mu = 0$ (line 2). The perturbation bound defined by the Laplace distribution is $exp(\lambda)$, which is a function of the noise scale λ, also called the exponential decay. Our choice of the Laplace mechanism is motivated by the way the exponential function scales multiplicatively, which simplifies the computation of the multiplicative bound $exp(\lambda)$.

However, there is no exact method to find the maximum noise scale $\lambda_{max} > 0$ that guarantees acceptable accuracy of a generated student model. Thus, we approximate λ_{max} empirically with respect to the candidate student model by incrementally using higher values of $\lambda > 0$ until we obtain a maximum value λ_{max} that results in a student model with unacceptable accuracy. Additionally, we explore the impact of increasing λ on the overall performance of the prediction framework and the transferability rate of evasion attacks across the n models (Section 5.3.5). We note that the randomness of Laplace noise allows the generation of n different student models using the same noise scale λ. Furthermore, even in case of a complete disclosure of the fabric of our approach, it is still difficult for an adversary to reproduce the exact pool of n models to use for adversarial example generation, due to the random aspect of the model pool generation approach.

Step-2: Retraining on Transformed Data. Minor distortions of the parameters of f_b have the

Algorithm 1: Student model generation.

```
    Result: $f_s$ : student model
1   Input:
        $f_b$: base model;
        $X_{train}$ : training set;
        $X_{test}$ : testing set;
        $acc_b \leftarrow Accuracy(f_b, X_{test})$;
        $T_i$ : data transformation function;
        $\lambda > 0$ : noise scale;
        $\epsilon > 0$ : used to detect the convergence of model training;
        $max\_acc\_loss$ : allowed margin of accuracy loss between $f_b$ and $f_s$;
        $adv\_train$ : boolean variable that indicates whether to train student model on adversarial data;
        $\Lambda$ : mixture of evasion attacks to use for adversarial training when $adv\_train = TRUE$;
        Step-1:// model weights perturbation.
2   $f_s \leftarrow f_b + Lap(0, \lambda)$;
        // $Lap(\mu, \lambda)$ returns an array of noise samples drawn from Laplace
        distribution $\frac{1}{2\lambda}\exp(-\frac{|\theta_b - \mu|}{\lambda})$
3   Step-2:// retraining on transformed data.
4   $f_s \leftarrow$ retrain($f_s$, $T_i(X_{train})$, $X_{test}$, $\epsilon$, $Adv = FALSE$);
    $acc_s \leftarrow$Accuracy($f_s$, $X_{test}$);
    while $acc_b - acc_s > max\_acc\_loss$ do
5       repeat Step-1 and Step-2 with smaller $\lambda$;
6   end
7   Step-3:// retraining on adversarial data.
8   if $adv\_train$ then
9       $f_s \leftarrow$retrain($f_s$, $\Lambda(T_i(X_{train}))$, $X_{test}$, $\epsilon$, $Adv = TRUE$);
            // check accuracy loss on clean test set.
10      $acc_s \leftarrow$ Accuracy($f_s$, $X_{test}$);
        while $acc_b - acc_s > max\_acc\_loss$ do
11          $f_s \leftarrow$ retrain($f_s$, $T_i(X_{train})$, $X_{test}$, $\epsilon$, $Adv = FALSE$);
12      end
13  end
```

potential to reduce the prediction accuracy of the resultant student model. Consequently, Step-1 is likely to produce student models that are less accurate than f_b. An accuracy recovery measure is necessary to ensure that each student model has acceptable accuracy close enough to f_b. To that end, we retrain the n newly created student models (line 3).

Diversity of the model pool is crucial for Morphence-2.0 's MTD core such that adversarial examples are less transferable across models (*Challenge-3*). In this regard, retraining all student models on X_{train} used for f_b results in models that are too similar to the decision function of f_b. It is, therefore, reasonable to use a distinct training set for each student model. To tackle data scarcity, we harness data augmentation techniques to perform n distinct transformations $\{T_1, ..., T_n\}$ on the

Algorithm 2: Student model retraining.

```
   Result: $f_s^{(i)}$ : student model
1  Def Retrain($f_s^{(i)}$, $X_{retrain}$, $X_{test}$, $\epsilon$, $Adv$):
       // For adversarial training we use adversarial test examples
          for validation.
2      if $Adv = TRUE$ then
3          $X_{test} \leftarrow \Lambda(X_{test})$
4      end
5      $acc_{tmp} \leftarrow$ Accuracy($f_s^{(i)}$, $X_{test}$);
        $epochs \leftarrow 0$;
        while TRUE do
6          $f_s^{(i)}$.train($X_{retrain}$, $epoch = 1$);
              $acc \leftarrow$ Accuracy($f_s^{(i)}$, $X_{test}$);
              // check training convergence.
7          if $epochs$ mod(5) $= 0$ then
8              if $|acc - acc_{tmp}| < \epsilon$ then
9                  break;
10             else
11                 $acc_{tmp} \leftarrow acc$;
12             end
13         end
14         $epochs \leftarrow epochs + 1$;
15     end
```

original training set X_{train} (e.g., translation, rotation, etc). The translation distance or the rotation degree are randomly chosen with respect to the validity constraint of the transformed set $T_i(X_{train})$. A transformed sample is valid only if it is still recognized by its original label. In our case, for each dataset, we use benchmark transformations proposed and validated by previous work [125]. We additionally double-check the validity of the transformed data by verifying whether each sample is correctly predicted by f_b.

Algorithm 2 illustrates `Retrain(`f_s`,`$X_{retrain}$`,`X_{test}`,`ϵ`,`Adv`)` the student model retraining function. It takes as inputs: a student model f_s, a retraining set $X_{retrain} = T_i(X_{train})$, a testing set X_{test}, a small positive infinitesimal quantity $\epsilon \to 0$ used for training convergence detection, and a boolean flag Adv. As indicated on line 15 of Algorithm 1, Adv is $FALSE$ since the retraining data does not include adversarial examples. The algorithm regularly checks for training convergence after a number of (e.g., 5) epochs. The retraining convergence is met when the current accuracy improvement is lower than ϵ (lines 7–13 in Algorithm 1).

The validity of the selected value of the noise scale λ used in Step-1 is decided by the outcome of Step-2 in regaining the prediction accuracy of a student model. More precisely, if retraining the student model (i.e., Step-2) does not improve the accuracy, then the optimisation algorithm that minimizes the model's loss function is stuck in a local minimum due to the significant distortion brought by weight perturbations performed in Step-1. In this case, we repeat Step-1 using a lower λ, followed by Step-2. The loop breaks when the retraining succeeds to regain the student model's accuracy, which indicates that the selected λ is within the maximum bound $\lambda < \lambda_{max}$ (Algorithm 1, lines 4-6). For more control over the accuracy of deployed models, we define a hyperparameter max_acc_loss, that is configurable by the defender. It represents the maximum prediction accuracy loss tolerated by the defender.

Step-3: Adversarial Training a Subset of p Student Models.

To motivate the need for Step-3, let us assess our design using just Step-1 and Step-2 with respect to challenges 1–3. Suppose Morphence-2.0 is deployed based only on Step-1 and Step-2, and for each input it picks the most confident student model and returns its prediction. On clean inputs, the MTD strategies introduced in Step-1 (via model weights perturbation) and Step-2 (via retraining on transformed training data) make Morphence-2.0 a moving target with nearly no loss on prediction accuracy. On adversarial inputs, an input that evades student model $f_s^{(i)}$ is less likely to also evade another student model $f_s^{(j)}$ because of the significant reduction of transferability between student model predictions because of Step-2. However, due to the exclusive usage of clean inputs in Step-1 and Step-2, an adversarial example may still fool a student model on first attempt. We note that the success rate of a repeated evasion attack is low because the randomness introduced in Step-1 disarms the adversary of a stable fixed target model that returns the same prediction for repeated queries on a given adversarial input. To significantly reduce the success of one-step attacks, we introduce selective adversarial training to reinforce the MTD strategy built through Step-1 and Step-2. More precisely, we perform adversarial training on a subset $p < n$ models from the n student models obtained after Step-2 (lines 7–13 in Algorithm 1). We note our choice of adversarial training is based on the current state-of-the-art defense. In principle, a defender is free to use a different (possibly better) method than adversarial training.

Why adversarial training on $p < n$ student models? There are three intuitive alternatives to integrate adversarial training to the MTD strategy: (a) adversarial training of f_b before Step-1; (b) adversarial training of all n student models after Step-2; or (c) adversarial training of a subset of student models after Step-2. As noted by prior work [98], (a) is bound to result in an inherited robustness for each student model, which costs less execution time compared to adversarial training of n student models. However, in this case, the inherent limitation of adversarial training, i.e., accuracy reduction on clean inputs, is also inherited by the n student models. Alternative (b) suffers from similar drawbacks. By adversarially training all n models, while making individual student models resilient against adversarial examples, we risk accuracy loss on clean data. Considering the drawbacks of (a) and (b), we pursue (c). In particular, we select $p < n$ student models for adversarial training (lines 7–13 in Algorithm 1). Consequently, the remaining $n-p$ models remain as accurate as f_b on clean data in addition to being diverse enough to reduce attack transferability among them.

Adversarial training approach. We now explain the details of adversarially training a student model with respect to Algorithm 1. Once again, the `Retrain` function illustrated in Algorithm 2 is invoked using different inputs (line 11). For instance, $X_{retrain} = \Lambda(T_i(X_{train})$, which indicates that f_s is trained on adversarial examples, is generated by performing a mixture of evasion attacks Λ on the transformed training set $T_i(X_{train})$ specified for student model $f_s^{(i)}$. To reduce accuracy decline on clean data, we shuffle the training set with additional clean samples from $T_i(X_{train})$. Furthermore, we use more than one evasion attack with different perturbation bounds $||\delta|| < \xi$ for adversarial samples generation to boost the robustness of the student model against different attacks (e.g., C&W [35], gradient-based [55], etc). Like Step-2, the training convergence is reached if the improvement of the model's accuracy on adversarial examples (i.e., the robustness) recorded periodically, i.e., after a number (e.g., 5) of epochs, becomes infinitesimal (lines 7–13 in Algorithm 2).

How to choose the values of p and n? The values p and n are defender-chosen hyper-parameters. Ideally, larger n favors the defender by creating a wider space of movement for a model's decision function (thus creating more uncertainty for repeated or correlated attacks). However, in practice n is conditioned by the computational resources available to the defender. Therefore, here we choose not to impose any specific values of n. We, however, recall that Q_{max} is proportional with the time needed to generate a pool of n models. Therefore, it is plausible that n need not be too large to lead to a long extension of the expiration time of the pool of n models due to a longer period T_n needed to generate n models that causes a large value of Q_{max}. Regarding the number of adversarially-trained models (p), there is no exact method to select an optimal value. Thus, we empirically examine all possible values of p between 0 and n to explore the performance change of Morphence-2.0 on clean inputs and adversarial examples. Additionally, we explore the impact of the value p on the transferability rate across the n student models (results are discussed in Section 5.3.5).

5.2.3 Scheduling Strategy

In the following, what we mean by *scheduling strategy* is the act of selecting the model that returns the label for a given input. There are multiple alternatives to reason about the scheduling strategy. Randomly selecting a model or taking the majority vote of all student models are both intuitive avenues. However, random selection does not guarantee effective model selection and majority vote does not consider the potential inaccuracy of adversarially trained models on clean queries.

Most Confident Model. We rather adopt a strategy that relies on the confidence of each student model. More precisely, given an input x, Morphence-2.0 first queries each student model and returns the prediction of the most confident model. Given a query x, the scheduling strategy is formally defined as: $\arg\max\{f_s^{(1)}(x), ..., f_s^{(n)}(x)\}$.

OOD-Powered Scheduling. On top of Morphence-1.0 , we extend our scheduling strategy with a pre-cursor decision component that determines whether an input needs to be predicted by the most confident model from the pool of p adversarially trained models or the most confident model of the remaining $n - p$ undefended models. Cognizant of the cost of robustness using adversarial training (i.e., accuracy loss on benign data), we aim to guide the scheduling approach to send benign queries to the $n - p$ highly accurate undefended models, while any adversarial query is sent to the remaining p models that are defended by adversarial training. To this end, as motivated in section 7.1, we leverage recent advances in OOD detection to separate between benign in-distribution queries from potential adversarial examples that are most likely OOD. Specifically, we adopt the current state-of-the-art method called ***SSD*** that trains a ***self-supervised outlier detector*** through learning a feature representation of the data distribution $\mathbb{P}_{train}$ used to train the target model f. Given an input sample x, SSD computes how far x is distant from the training data distribution using the Mahalabolis distance metric ($M(x, \mathbb{P}_{train})$) [109] (Section 5.1.1). Although, most adversarial examples are known to be OOD examples [13], in order to effectively use SSD as an adversarial examples detector, a threshold definition (τ) is necessary to separate between potential OOD adversarial examples and benign in-distribution queries that exhibit tolerable distribution shifts (i.e., $M(x, \mathbb{P}_{train}) < \tau$) from those that are far away from $\mathbb{P}_{train}$ due to potential adversarial perturbations (i.e., $M(x, \mathbb{P}_{train}) \geq \tau$).

OOD Detection Threshold Determination. The OOD distance score of an adversarial example $M(x + \delta, \mathbb{P}_{train})$ tends to be higher than the distance of benign inputs, i.e., $M(x, \mathbb{P}_{train})$, due to distribution shifts that can be caused by adversarial perturbations. In Figure 5.2, we compare the OOD scores (y-axis) of FGSM samples on CIFAR10 (red points) to the OOD scores of benign samples (green points). Figure 5.2 visually confirms that most of the adversarial samples (red) have higher OOD scores compared to the benign samples (green). However, a threshold is needed to define the maximum allowable distance $M(x, \mathbb{P}_{train})$ that a sample x can record in order to be considered benign. To this end, we refer to a subset of benign in-distribution training data that we use for threshold selection. We call it tuning data $X_\tau \sim \mathbb{P}_{train}$. Intuitively, we can select as a threshold the maximum distance score recorded by samples in the tuning data X_τ. Formally, $\tau = \max_{x \in X_\tau} M(x, \mathbb{P}_{train})$. With respect to Figure 5.2, such a threshold would fail in practice. Particularly, we observe few green points (benign) that have OOD scores similar or higher than most red points (adversarial). Such unexpected outliers in X_τ can make τ very high. Therefore, it would classify most of adversarial examples as benign. Furthermore, the threshold should be agnostic to the studied attack (e.g., FGSM) and dataset (e.g., CIFAR10). Hence, a fixed threshold is not a suitable choice. Moving forward, in Figure 5.2, we explore the k^{th} *percentile* (black curve) of X_τ as a threshold with respect to different values of k (x-axis). In other words, we select as threshold τ the lowest OOD score in X_τ that is greater than a $k\%$ of all scores recorded by $x \in X_\tau$. Formally,

$$\tau = k^{th} Percentile\{M(X_\tau), \mathbb{P}_{train})\} \tag{5.1}$$

k is a variable that can be changed with respect to the studied dataset and attacks. For instance, in Figure 5.2, k in the range $[85^{th}, 95^{th}]$ is above the mean of benign samples (X_τ) and below the mean of their adversarial counterparts (X'_τ). Consequently, one can separate between benign and adversarial samples while ignoring outliers of benign samples that have an OOD score higher than $k\%$ of all other samples in X_τ. By leveraging the $k^{th} percentile$ selection strategy we find a tradeoff between accurately detecting adversarial examples and avoiding false positives. In section 5.3.2, we investigate the effectiveness of the scheduling strategy compared to the most confident model approach.

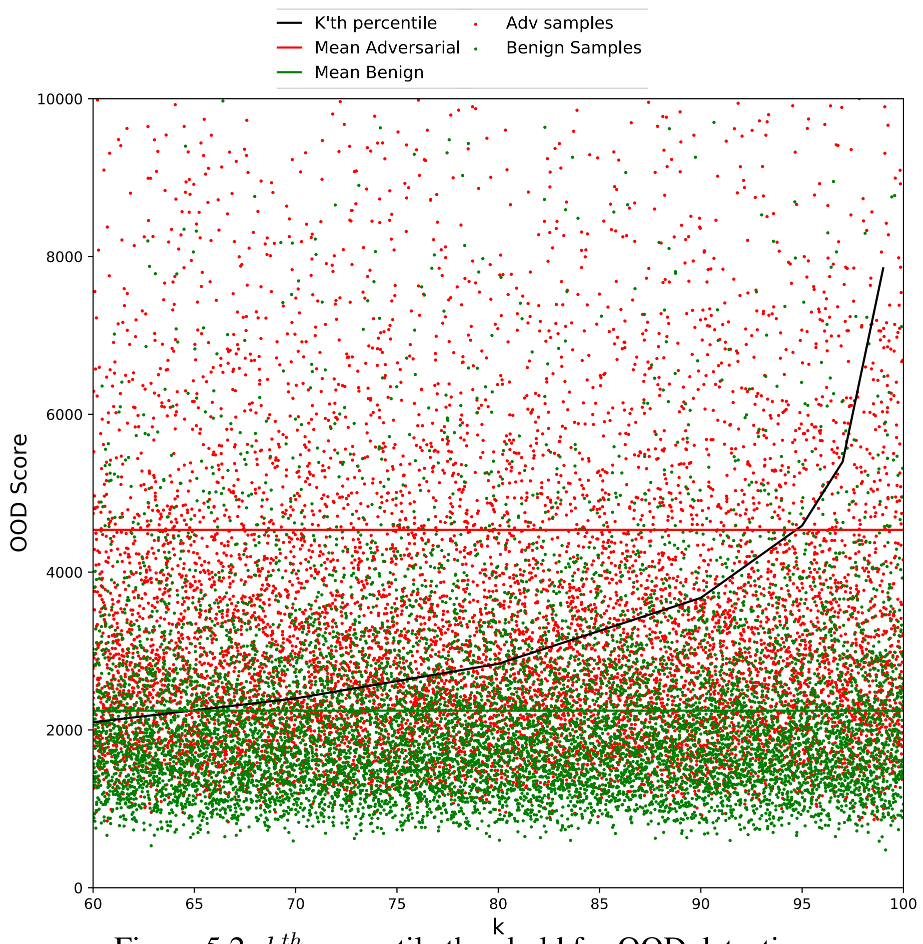


Figure 5.2: k^{th} percentile threshold for OOD detection.

5.2.4 Model Pool Renewal

Given that n finite, with enough time, an adversary can ultimately build knowledge about the prediction framework through a series of queries. For instance, if the adversary correctly guesses model pool size n, it is possible to map which model is being selected for each query by closely monitoring the prediction patterns of multiple examples. Once compromised, the whole framework becomes a sitting target since its movement is limited by the finite number n.

On way to avoid such exposure is abstaining from responding to a "suspicious" user [53]. However, given the difficulty of precisely deciding whether a user is suspect based solely on the track of their queries, this approach has the potential to result in a high abstention rate, which unnecessarily leads to denial of service for legitimate users. We, therefore, propose a relatively expensive yet effective method to ensure the continuous mobility of the target model without sacrificing the quality of service. More precisely, we actively update the pool of n models when a query budget upper bound Q_{max} is reached. To maintain the quality of service in terms of query response time, the update needs to be seamless. To enable seamless model pool update, we ensure that a buffer of batches of n student models is continuously generated and maintained on stand-by mode for subsequent deployments.

The choice of Q_{max} determines how dynamic the target model under the condition: *"the buffer of pools of models is never empty at a time of model batch renewal"*.
Suppose at time t the buffer contains K_t pools of models. A new pool is removed from the buffer after every period of Q_{max} queries and a clean-slate pool is activated. Thus, the buffer is exhausted after $K_t.Q_{max}$ number of queries. Supposing that the per-query response time is T_q and the generation of a pool of n models lasts a period of T_n, the above condition is formally expressed as: $K_t.Q_{max}.T_q > T_n$, s.t. $K_t > 0$. The inequality implies that the time to exhaust the whole buffer of pools must be always greater than the duration of creating a new pool of n models. Additionally, it shows that Q_{max} is variable with respect to the time t and the number of models in one pool n. Ideally, Q_{max} should be as low as possible to increase the mobility rate of target model. Thus, the optimal solution is $\frac{T_n}{K_t.T_q}$.

5.3 Evaluation

We now evaluate Morphence-2.0 . Section 5.3.1 presents experimental setup. Section 5.3.2 compares Morphence-2.0 with the undefended base model, adversarially trained base model, and Morphence-1.0 . Section 5.3.3 evaluates the effectiveness of the scheduling strategies. Section 5.3.4 examines the impact of dynamic scheduling and model pool renewal. Finally, Section 5.3.5 sheds light on the impact of individual Morphence-2.0 components on robustness and attack transferability.

5.3.1 Experimental Setup

To enable a fair comparison with Morphence-1.0 , we evaluate Morphence-2.0 using the same experimental setup introduced in Morphence-1.0 's paper [13]. We also experiment both approaches against additional attacks.

Datasets: We use two benchmark datasets: MNIST [80] and CIFAR10 [73]. We use 5K test samples of each dataset to perform 5K queries for evaluation.

Attacks: We use three *white-box* attacks (FGSM [55], PGD [89], and C&W [35]) and one *black-box* attack (SPSA [128]). Details of these attacks appear in Morphence-1.0 [13]. For C&W, PGD, and FGSM, we assume the adversary has white-box access to f_b. For SPSA, the adversary has oracle access to Morphence-2.0 . We carefully chose each attack to assess our contribution claims. For instance, C&W, one of the most effective white-box attacks, is suggested as a benchmark for ML robustness evaluation [33]. PGD is a widely used gradient-based attack that can reach a higher evasion rate, while FGSM is fast and scalable on large datasets and generalizes across models [78]. In addition, the relatively high transferability rate of FGSM attacks across models makes it suitable to evaluate the effectiveness of Morphence-2.0 's different components to reduce attack transferability across student models. To explore Morphence-2.0 's robustness against query-based black-box attacks, we employ SPSA since it performs multiple correlated queries to craft adversarial examples. For all attacks we use a perturbation bound $||\xi||_\infty < 0.3$.

Base Models: As base models we reuse the same models previously introduced in [13]. Notably, 6-layer CNN model that reaches a test accuracy of 99.72% (i.e.,"MNIST-CNN") and a CNN CIFAR10 model [40] that reaches an accuracy of 83.63% on a test set of 5K (i.e., "CIFAR10-CNN").

Baseline Defenses: In accordance with the baseline defenses adopted in [13], for both datasets, we compare Morphence-2.0 with an *undefended fixed* model, an *adversarially-trained fixed* model, and Morphence-1.0 .

Hyper-parameters: We refer to the same hyper-parameters in [13]. Notably, we use 5 pools of models where each of size $n = 10$, $Q_{max} = 1$K.

In Table 5.1, for MNIST, we fix $\lambda = 0.1$, $p = 5$ and for CIFAR10, we use $\lambda = 0.05$ and $p = 8$ or $p = 9$. More details about hyper-parameters tuning is explained in [13].

Metrics: As adopted in [13], we use *Accuracy* and *Average Transferability Rate (ATR)* ([13]). We compute *ATR* across all n models to evaluate the effectiveness of the data transformation measures at reducing attack transferability. To compute the transferability rate from model $f_s^{(i)}$ to another model $f_s^{(j)}$, we calculate the rate of adversarial examples that succeeded on $f_s^{(i)}$ that also succeed on $f_s^{(j)}$. Across all student models, *ATR* is computed as:

	MNIST-CNN Accuracy					CIFAR10-CNN Accuracy				
Attack	**Undefended**	**Adv-Train**	**Morphence-1.0**	**Morphence-2.0**		**Undefended**	**Adv-Train**	**Morphence-1.0** $(p = 8, p = 9)$	**Morphence-2.0** $(p = 9)$	
				$\tau = 95^{th}$	$\tau = 90^{th}$				$\tau = 95^{th}$	$\tau = 90^{th}$
No Attack	99.72%	97.17%	99.04%	**99.58%**	**99.58%**	83.63%	75.37%	**84.64%**, 82.65%	**83.34%**	79.44%
FGSM [55]	9.98%	42.38%	71.43%	**86.48%**	**86.48%**	19.98%	36.62%	36.44%, 38.78%	**46.82%**	**46.82%**
PGD [89]	0.3%	4.14%	58.02	**94.08%**	**94.08%**	10.13%	14.47%	10.14%, 10.28%	**28.19%**	**52.04%**
C&W [35]	0.0%	0.0%	**97.75%**	92.28%	96.41%	1.25%	1.34%	44.50%, 40.91%	**45.08%**	**44.67%**
SPSA [128]	29.04%	59.43%	97.77%	**98.07**	**98.62%**	38.96%	59.08%	60.85%, 62.83%	**70.06%**	**75.13%**

Table 5.1: Morphence-2.0 robustness compared to an undefended, adversarially trained fixed model, and Morphence-1.0 .

$$ATR = \frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \frac{N_{adv}(f_s^{(i)} \rightarrow f_s^{(j)})}{N_{adv}(f_s^{(i)})},$$

where $N_{adv}(f_s^{(i)}) = \{x' \in X'_{test}; f_s^{(i)}(x') \neq y_{true}\}$ is the number of adversarial examples that fooled $f_s^{(i)}$ and $N_{adv}(f_s^{(i)} \rightarrow f_s^{(j)}) = \{x' \in N_{adv}(f_s^{(i)}); f_s^{(j)}(x') \neq y_{true}\}$ is the number of adversarial examples that fooled $f_s^{(i)}$ that also fool $f_s^{(j)}$.

5.3.2 Robustness Against Evasion Attacks

Based on results summarized in Table 5.1, we now evaluate the robustness of Morphence-2.0 compared with the undefended base model, adversarially trained base model, and Morphence-1.0 across the 4 reference attacks. Note that we are particularly interested in the difference in robustness between Morphence-1.0 (scheduling: based on most confident model) and Morphence-2.0 (scheduling: powered by OOD detector).

Robustness in a nutshell: Across all attacks and threat models, both Morphence-1.0 and Morphence-2.0 are more robust than *adversarial training* for both datasets. On MNIST, across all four attacks, Morphence-1.0 and Morphence-2.0 significantly outperform adversarial training by an average of $\approx 55\%$ and $\approx 67\%$, respectively. On CIFAR10, Table 5.1 suggests similar results. On CIFAR10, Morphence-1.0 improves robustness by $\approx 2\%$ on FGSM and $\approx 4\%$ on SPSA when $p = 9$. With regards to C&W, it drastically improves robustness compared to the baseline models (i.e., $\approx 41\%$) while we observe a small decrease against PGD (more explanation in 5.3.3). Additionally, Morphence-2.0 , further improves our results. It outperforms adversarial training across all attacks by an average of 22% for both OOD detection threshold configurations. These findings show that Morphence-2.0 is more robust than Morphence-1.0 when powered by OOD detection for input scheduling.

Accuracy loss on clean data: Table 5.1 indicates that, unlike adversarial training on a fixed model, Morphence-2.0 does not sacrifice accuracy to improve robustness. For instance, while adversarial training drops the accuracy of the undefended MNIST-CNN by $\approx 3\%$, both Morphence-1.0 and Morphence-2.0 maintain it close to its original value ($> 99\%$). Similar results are observed on CIFAR10. On one hand, even after using 9 adversarially-trained models ($p = 9$) of $n = 10$ student models for each pool, the accuracy loss is $\leq 1\%$ for Morphence-1.0 . As for Morphence-2.0 , it becomes marginal when the threshold is set to the 95^{th} percentile ($\tau = 95^{th}$). On another hand, adversarial training sacrifices 8.26% of the original accuracy of CIFAR10-CNN. These results are linked to the effectiveness of the adopted scheduling strategy to assign clean queries mostly to student models that are not adversarially trained, thus, accurate on clean data (more details in 5.3.3). Furthermore, Morphence-2.0 can improve the original accuracy for lower values of p. For instance, Table 5.1 shows an improvement of 1% in the accuracy on CIFAR10 clean data when using Morphence-1.0 with $p = 8$, compared to the undefended baseline model. In conclusion, our findings indicate that *Morphence-2.0 is much more robust compared to adversarial training on fixed model and Morphence-1.0 while preserving the original accuracy of the undefended f_b.*

Robustness against C&W: Inline with the state-of-the-art, Table 5.1 shows that C&W is highly effective on the baseline fixed models. For both datasets, even after adversarial training, C&W attack can maintain its high attack accuracy for both datasets (100% on MNIST and $\approx 99\%$ on CIFAR10). However, it significantly fails to achieve the same attack accuracy on Morphence-2.0 . For instance, Morphence-1.0 increases the robustness against C&W data by $\approx 97\%$ for MNIST and $\approx 40\%$ for CIFAR10 compared to adversarial training on fixed model. This significant improvement in robustness is brought by the moving target aspect included in Morphence-2.0 . More precisely, given the low transferability rate of C&W examples across different models, C&W queries that are easily effective on f_b are not highly transferable to the student models, hence less effective on Morphence-2.0 . More discussion that reinforces this insight is presented in Section 5.3.5.

Robustness against FGSM and PGD: Although known to be a less effective attack than C&W on the fixed models, FGSM performs better on Morphence-2.0 . For instance, Morphence-1.0 accuracy on FGSM data is 26.32% less for MNIST and $\approx 3\%$ less for CIFAR10, compared to C&W. These findings are explained by the high transferability rate of FGSM examples across student models and f_b. Similar behavior is observed for PGD. However, despite the transferability issue, Morphence-1.0 still improves robustness on FGSM and PGD data. More discussion about the transferability effect on Morphence-2.0 are provided in Section 5.3.5. It is noteworthy that, once again, Morphence-2.0 further improves the robustness against FGSM and PGD by an average margin of $\approx 25\%$ on MNIST and $\approx 20\%$ on CIFAR10 (averaged across FGSM and PGD). These results confirm the importance of introducing the OOD detector to separate between benign queries and potential adversarial queries.

Robustness against SPSA: We now turn to a case where the adversary has a black-box prediction API access to Morphence-2.0 to issue multiple queries. The adversary performs iterative perturbations of an input to reach the evasion goal. Inline with the adversary's goal here, SPSA performs multiple queries using different variations of the same input to reduce the SPSA loss function. Table 5.1 shows that Morphence-2.0 is more robust on SPSA than the two baseline fixed models. Due to its dynamic characteristic, Morphence-2.0 is a moving target. Hence, it can derail the iterative query-based optimization performed by SPSA. More analysis into the impact of the dynamic aspect is discussed in Section 5.3.4. Consistent with results against white-box attacks, Morphence-2.0 is also better than Morphence-1.0 against the studied black-box attack, SPSA. In Section, 5.3.3, analyze scheduling history of Morphence-1.0 vs Morphence-2.0 over benign queries and adversarial queries of different attacks to verify highlight the improved robustness by Morphence-2.0 over Morphence-1.0 .

Robustness across datasets: Morphence-2.0 performs much better on MNIST than CIFAR10. This observation is explained by various factors. First, CNN models are highly accurate on MNIST ($> 99\%$) than CIFAR10 ($\approx 84\%$). Second, across all attacks, on average, *adversarial training* is more effective on MNIST; it not only leads to higher robustness (i.e., $\approx +31\%$ for MNIST compared to $\approx +19\%$ for CIFAR10) it also sacrifices less accuracy on clean MNIST data (i.e., -2.55% for MNIST compared to -8.26% for CIFAR10). Consequently, the p adversarially-trained student models on MNIST are more robust and accurate than the ones created for CIFAR10. Finally, CIFAR10 adversarial examples are more transferable across student models than MNIST. More results about the transferability factor are detailed in Section 5.3.5.

Observation 1: Compared to adversarial training, both Morphence-1.0 and Morphence-2.0 improve robustness to adversarial examples on both benchmark datasets (MNIST, CIFAR10) for both white-box and black-box attacks. This is achieved without sacrificing accuracy on clean data.

5.3.3 Effectiveness of Scheduling Strategy

As illustrated in Section 5.2.3, given an input query, a scheduling strategy selects the most suitable model for the prediction task. We recall that in Morphence-1.0 the scheduler simply picks the most confident model from the active batch of all the n models regardless of the nature of the input, while in Morphence-2.0 we extend the scheduling approach with another decision layer that separates OOD adversarial inputs from in-distribution benign ones. Our approach aims to precisely assign adversarial examples to the most confident model from the p adversarially trained models, while attributing benign queries to the remaining $n - p$ undefended models that are more accurate on benign data (explained in section 7.2). In Figure 5.3, we keep track of the scheduling history of different Morphence-1.0 and Morphence-2.0 with respect to clean (benign) queries and adversarial queries. In the following, as we interpret the scheduling history recorded by each design on both datasets, we refer to Table 5.1 to explain the impact of the scheduling precision on improving the robustness of our approach.

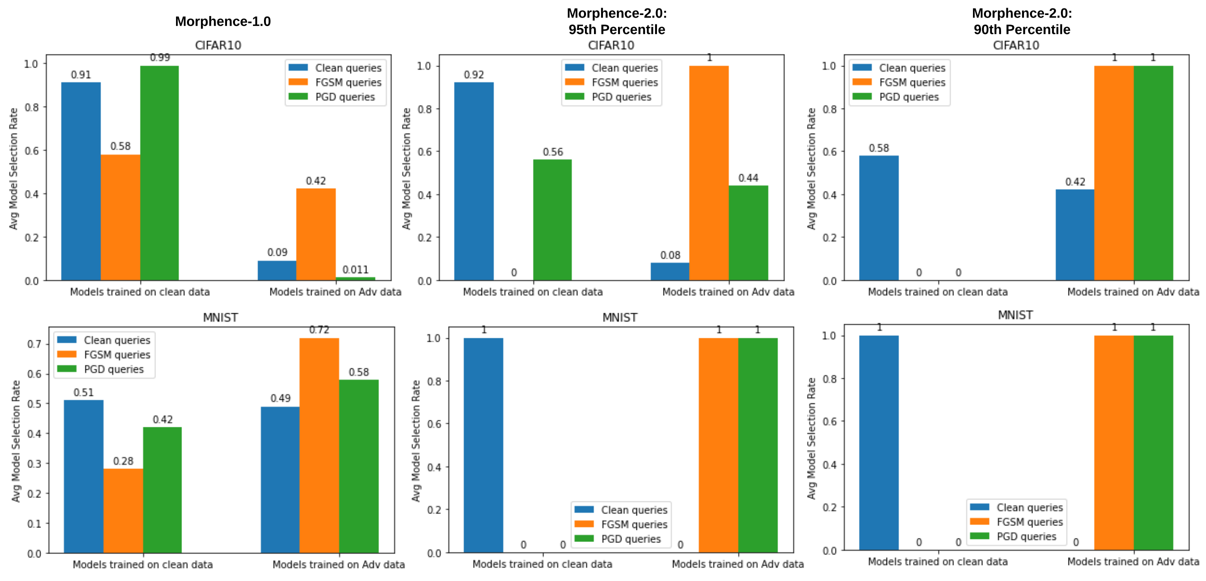


Figure 5.3: Scheduling results of Morphence-1.0 compared to Morphence-2.0 .

On CIFAR10, Morphence-1.0 successfully assigned 91% of the clean queries (blue bar) to undefended models, which explains its high accuracy on benign data compared to adversarial training that sacrifices $\approx 8\%$ of the original accuracy (Table 5.1). However, it fails to correctly handle adversarial queries. Almost all PGD queries (green bar) are falsely assigned to undefended models which explain the very low accuracy of Morphence-1.0 against PGD attack recorded in Table 5.1. As for the FGSM queries (orange bar), Morphence-1.0 exhibits a better scheduling precision, however, it assigns only 42% of FGSM queries to the adversarially-trained models. On the contrary, powered by the OOD detection, Morphence-2.0 succeeds to assign all FGSM queries to the adversarially-trained models which reflects the improvement in robustness observed in Table 5.1. Same results are observed for the PGD queries when $k = 90^{th}$ is used as threshold. We recall that using a lower threshold τ favors more the detection of adversarial examples while it tolerates some false positives on benign data. This tradeoff is observed on CIFAR10 results. More precisely, while Morphence-2.0 with $\tau = 95^{th}$ leads to better scheduling precision on benign data (92%) it scores only 44% on PGD data. A reduction of the threshold to $\tau = 90^{th}$ leads to 100% precision on PGD queries while it scores worse on clean data. These findings explain why, in Table 5.1, Morphence-2.0 with $\tau = 90^{th}$ records better robustness but it sacrifices more accuracy on CIFAR10 benign data.

On MNIST, Morphence-2.0 scores a perfect scheduling precision for both threshold configurations. In other words, all adversarial queries are successfully fed to adversarially trained models, while all clean queries are assigned to undefended models. Particularly, Figure 5.3 shows that, for both threshold configurations, unlike Morphence-1.0 , Morphence-2.0 assigns 100% of clean

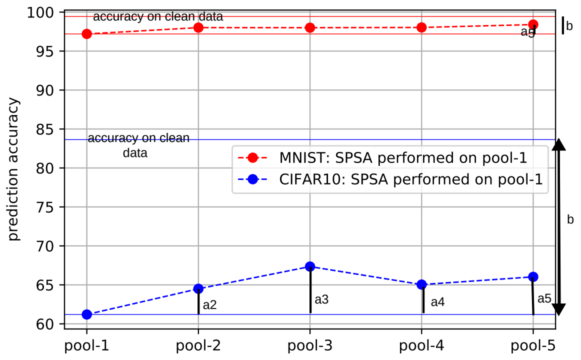
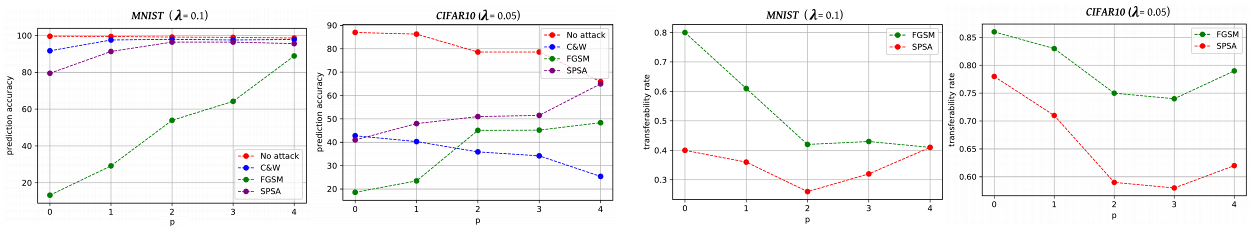


Figure 5.4: Impact of model pool renewal on repeating previously successful SPSA queries: Prediction accuracy of pools 2-5 of models on adversarial examples is generated through multiple queries on pool-1.



(a) p vs. accuracy.

(b) p vs. average transferability rate.

Figure 5.5: # adversarially trained models p with respect to accuracy (left) and transferability rate (right).

queries only to undefended models, while it attributes 100% of FGSM and PGD queries to adversarially trained models. Such high scheduling precision on MNIST explains the significant improvement in robustness by Morphence-2.0 in Table 5.1.

Observation 2: The OOD-powered scheduling in Morphence-2.0 is more effective than the scheduling strategy in Morphence-1.0 which relies on the most confident model.

Observation 3: In Morphence-2.0 , a careful configuration of the percentile order k is mandatory to find the defender-desired trade-off between adversarial example detection (higher robustness) and benign example detection (high accuracy on benign data).

5.3.4 Model Pool Renewal vs. Repeated Attacks

To diagnose the impact of the model pool renewal on the effectiveness of repeated adversarial queries, we perform the SPSA attack by querying only pool-1 of Morphence-2.0 . Then we test the generated adversarial examples on the ulterior pools of models (i.e., pools 2–5). For this experiment, we adopt the notation *Failed Repeated Queries (FRQ)* that represents the number of ineffective repeated adversarial queries ("a" in Figure 5.4) from the total number of repeated previous adversarial queries ("b" in Figure 5.4). Results are shown in Figure 5.4. With respect to the baseline evasion results (i.e., accuracy of pool-1 on SPSA), we observe for both datasets an increase of the accuracy (hence robustness) of ulterior pools (i.e., pools 2–5) on SPSA data generated by querying pool-1. These findings indicate that some of the adversarial examples that were successful on pool-1 are not successful on ulterior pools (i.e., *FRQ*> 0). More precisely, on average across different pools, $\approx 87\%$ of the previously effective adversarial examples failed to fool ulterior pools on MNIST ($FRQ = \frac{a}{b} \approx 87\%$). As for CIFAR10, $\approx 21\%$ of previously effective adversarial examples are not successful on ulterior pools ($FRQ = \frac{a}{b} \approx 21\%$). These results reveal the impact of the *model pool renewal* on defending against repeated adversarial queries. However, we note that unlike MNIST, repeated CIFAR10 adversarial queries are more likely to continue to be effective on ulterior pools ($\approx 79\%$ are still effective), which indicates the high transferability rate of SPSA examples across different pools.

Observation 4: Morphence-2.0 significantly limits the success of *repeating previously successful adversarial examples* due to the *model pool renewal* step that regularly and seamlessly updates the model pool to invalidate patterns in adversary's observations.

5.3.5 Impact of Model Pool Generation Components

We now focus on the impact of each component of the student model generation steps on Morphence-2.0 's *robustness* and *transferability rate across student models*. To that end, we generate different pools of $n = 4$ student models using $0 < \lambda < \lambda_{max}$ and $0 \leq p \leq n$. We monitor changes in *accuracy* and *ATR* across the n student models for different values of λ until the maximum bound λ_{max} is reached. Additionally, we perform a similar experiment where we try all different possible values of $0 \leq p \leq n$. Finally, we evaluate the effectiveness of training each student model on a distinct set and its impact on the reduction of the *ATR* compared to using the same X_{train} to train all student models.

Retraining on adversarial data. For this experiment, we fix a λ value that offers acceptable model accuracy and try all possible values of p (i.e., $0 \leq p \leq 4$).

Impact on robustness against adversarial examples: Figure 5.5a shows accuracy of Morphence-2.0 with respect to $0 \leq p \leq 4$, for both datasets. For all attacks on MNIST, a $0 \rightarrow n$ increase in p leads to a higher robustness. Similar results are observed for SPSA and FGSM on CIFAR10. However, the accuracy on C&W data is lower when p is higher which is consistent with CIFAR10 results of Morphence-1.0 in Table 5.1 ($p = 8$ vs. $p = 9$). As stated earlier (Section 5.3.2), adversarial training on CIFAR10 leads to a comparatively higher accuracy loss on clean test data. Consequently, the accuracy on clean data decreases when we increase p ("No attack" in Figure 5.5a). We conclude that, retraining student models on adversarial data is a crucial step to improve the robustness of Morphence-2.0 . However, p needs to be carefully picked to reduce accuracy distortion on clean data caused by adversarial training (especially for CIFAR10), while maximizing Morphence-2.0 's robustness. From Figure 5.5a, for both datasets, $p = 3$ serves as a practical threshold for $n = 4$ (it balances the trade-off between reducing accuracy loss on clean data and increasing accuracy on adversarial data).

For $p = 0$, although no student model is adversarially trained, we observe an increase in Morphence-2.0 's robustness. For instance, compared to the robustness of the undefended model on MNIST reported in Table 5.1, despite $p = 0$ (Figure 5.5a), the accuracy using a pool of 4 models is improved on FGSM data ($9.98\% \rightarrow\approx 18\%$). Similar results are observed for C&W on both datasets (MNIST: $0\% \rightarrow\approx 91\%$, CIFAR10: $0\% \rightarrow\approx 42\%$). These findings, once again, indicate the impact of the MTD aspect on increasing Morphence-2.0 's robustness against evasion attacks.

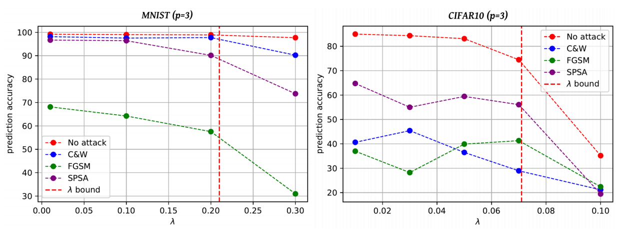
Impact on the evasion transferability across student models: Adversarially training a subset of student models leads to more diverse student models compared to those trained on just clean data – this might reduce *ATR* across models. Figure 5.5b shows *ATR* of SPSA and FGSM adversarial examples across student models for $0 \leq p \leq 4$. We choose SPSA and FGSM in view of their high transferability across ML models. Figure 5.5b shows that, for both datasets, *ATR* of both attacks is at its highest rate when $p = 0$. Then, it decreases for larger values of p (i.e., $p = 1 \rightarrow 3$), until it reaches a minimum (i.e., $p = 2$ for MNIST and $p = 3$ for CIFAR10). Finally, *ATR* of both attacks increases again for $p = n = 4$. In this final case, all student models are adversarially trained, therefore they are less diverse compared to $p \in \{1, 2, 3\}$. We conclude that the choice of p has an impact, not only on the overall performance of Morphence-2.0 , but also on the *ATR* across student models.

Noise Scale λ. We begin with $p = 3$ and incrementally try different configurations of $\lambda > 0$ until we reach a maximum bound λ_{max}. In Figures 5.6a and 5.6b, with respect to different values of $0 < \lambda < \lambda_{max}$, we investigate the impact of the *weights perturbation* step on accuracy and on *ATR* across student models. For both datasets, λ_{max} is presented as a vertical bound (i.e., red vertical line).

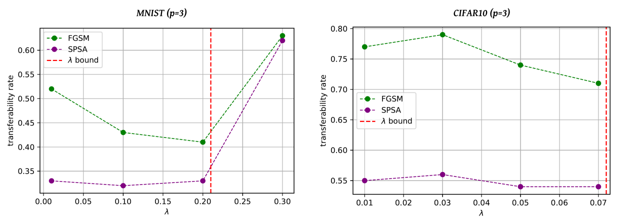
Impact on ATR: Figure 5.6b indicates that an increase in the noise scale λ, generally, leads to the decrease of the transferability rate of adversarial examples across student models, for both datasets. This observation is consistent with our intuition (in Section 7.2) that higher distortions on f_b weights lead to the generation of more diverse student models.

Impact on the accuracy: For MNIST dataset, we observe that higher model weights distortion (i.e., higher λ) leads to less performance on clean data and on all the studied adversarial data (e.g., C&W, FGSM and SPSA). As for CIFAR10, Figure 5.6a indicates different results on SPSA and FGSM. For instance, unlike C&W examples, which are much less transferable, the accuracy on FGSM data reaches its highest when $\lambda = 0.05$. Similarly, we observe an increase of the accuracy on SPSA data for $\lambda = 0.05$. Next, we further discuss the difference in accuracy patterns on FGSM and SPSA between MNIST and CIFAR10.

Best λ configuration: On MNIST, $\lambda = 0.1$ represents a tradeoff that balances the reduction



(a) λ vs. accuracy.



(b) λ vs. average transferability rate.

Figure 5.6: Noise scale λ vs. accuracy (left) and average transferability rate (right).

	MNIST-CNN		**CIFAR10-CNN**	
	FGSM	**SPSA**	**FGSM**	**SPSA**
Retraining on X_{train}	0.88	0.52	0.95	0.84
Retraining on $T_i(X_{train})$	**0.80**	**0.40**	**0.86**	**0.78**

Table 5.2: Comparison of *ATR* of FGSM and SPSA when student models are retrained on X_{train} vs. on $T_i(X_{train})$ ($p = 0$).

of the tansferability rate and the reduction of the accuracy loss, which are conflicting. As for CIFAR10, we choose $\lambda = 0.05$ to balance between Morphence-2.0 performance against non-transferable attacks (e.g., C&W) and transferable attacks (e.g., FGSM and SPSA).

Observation 5: Morphence-2.0 's performance is influenced by the values of its hyper-parameters (e.g., λ, n and p). Empirically estimating the optimal configuration of Morphence-2.0 contributes to the reduction of the *ATR* across student models, which leads to an increased robustness against adversarial examples.

Retraining on transformed data: We now evaluate to what extent using data transformation reduces the transferability rate of adversarial examples. To that end, we compute the *ATR* of FGSM and SPSA, and we compare it with the baseline case where all student models are retrained on the same training set X_{train}. To precisely diagnose the impact of data transformations on transferability, we exclude the effect of Step-3 by using $p = 0$, in addition to the same λ configuration adopted before. Therefore, we note that the following results do not represent the actual transferability rates of Morphence-2.0 student models (covered in previous discussions). Results reported in Table 5.2 show that performing different data transformations T_i on the training set X_{train} before retraining leads to more diverse student models. For instance, we observe $\approx -8,5\%$ less transferable FGSM examples on average across both datasets and an average of $\approx -9\%$ less transferable SPSA examples.

Observation 6: *Training data transformation* is effective at reducing *average transferability rate*.

Despite Morphence-2.0 advances to reduce ATR, the transferability challenge still has room for improvement. Prior work examined the adversarial transferability phenomenon [95, 46]. Yet, rigorous theoretical analysis or explanation for transferability phenomenon is a work in-progress in the literature.

5.4 Conclusion

While prior defenses against adversarial examples aim to defend a fixed target model, Morphence takes a moving target defense strategy that sufficiently randomizes information an adversary needs to succeed at fooling ML models with adversarial examples. In Morphence, model weights perturbation, data transformation, adversarial training, and dynamic model pool scheduling, and seamless model pool renewal work in tandem to defend adversarial example attacks. Our extensive evaluations across white-box and black-box attacks on benchmark datasets suggest Morphence significantly outperforms adversarial training in improving robustness of ML models against adversarial examples. It does so while maintaining (at times improving) accuracy on clean data and reducing attack transferability among models in the Morphence pool. By tracking the success/-failure of repeated attacks across batches of model pools, we further validate the effectiveness of the core MTD strategy in Morphence in thwarting repeated/correlated adversarial example attacks. Looking forward, we see great potential for moving target strategies as effective countermeasures to thwart attacks against machine learning models.

CHAPTER 6

Out2In: Towards Machine Learning Models Resilient to Adversarial and Natural Distribution Shifts

6.1 Introduction

Despite their compelling performance in experimental benchmarks, Machine learning (ML) models struggle to maintain their accuracy when deployed in practice, specifically, when facing inputs that lie *out of distribution* of their train/test data. The main reason for the poor performance of ML models in practice has to do with the fundamental assumptions that govern model training and testing: *the IID assumptions* [53]. Under the IID assumptions, training and test examples are all generated *independently* and from an *identical* probability distribution. Given a training data and a test data, both of which are drawn IID from an identical distribution, the goal of the training is to learn a model that generalizes well on test data, i.e., *in-distribution generalization*.

When a ML model is deployed in the wild, such as in a ML-as-a-Service (MLaaS) setting, inputs to the model may no more respect the IID assumptions. Under such a situation, to remain useful, the model needs to generalize on inputs that lie far away from the model's natural distribution, i.e., *out of distribution (OOD) generalization*. Otherwise, the model's accuracy degrades to a level that makes it *unreliable* for real-life deployment. Relating to ML unreliability in practice, prior works have revealed that ML models are vulnerable to *adversarial examples* (i.e., maliciously perturbed inputs) that successfully deplete prediction accuracy [55, 89, 35].

Early defenses [29, 24] pointed out that adversarial examples can be detected (e.g., using estimators) as samples from "unknown class" that should be rejected. In this work, we investigate the *link* between both problems (i.e., OOD generalization and adversarial examples) by, first, highlighting the observation that the *adversarial examples problem* is in fact a part of the wider *OOD generalization problem*, following the intuition that adversarial inputs may not conform to the IID setting due to adversarial perturbations. Hence, addressing primarily the *OOD generalization* *problem* as a major cause of the emergence of *adversarial examples*. Second, re-purposing OOD detection approaches for adversarial example detection. Third, exploring different in-distribution mapping strategies on OOD examples (particularly adversarial examples) instead of rejecting them.

Though not all OOD samples are adversarial, we observe that most adversarial examples are OOD samples [53], which leads to the speculation that, the lack of OOD generalization is a potential major cause of the susceptibility to adversarial examples. Recognizing such *cause-effect* link, we aim to rethink existing approaches to defend against adversarial examples *(effect)* by focusing on the OOD generalization problem *(cause)* towards ML models robust to not only adversarial examples but also to natural distribution shifts.

Early attempts [57, 65] to harden ML models against adversarial examples provided only marginal robustness improvements. Heuristic defenses [94, 42, 25, 141, 135, 88, 58, 31, 119] were subsequently broken [35, 34, 61, 22]. Although *adversarial training* [126] remains effective against known attacks, robustness comes at a cost of accuracy penalty on clean data. *Certified defenses* [81, 37, 82] are limited in the magnitude of robustness guarantee they offer and operate on a restricted class of attacks constrained to LP-norms [81, 134]. Apart from adversarial training, current defenses share the focus on answering the question *"how to defend against adversarial examples?"* instead of exploring the question *"why are ML models vulnerable to adversarial examples?"*. We argue that tackling the cause, i.e., lack of OOD generalization, is essential for building ML models robust to OOD inputs, both adversarial and benign. In this context, adversarial training techniques [126] can be perceived as attempts to generalize the ML model on adversarial examples distribution, which makes them share the same motivation as our approach. We, therefore, compare our approach with adversarial training and ensemble adversarial training.

We propose Out2In to enable a model trained under the IID assumptions and generalizes well on in-distribution test data to also generalize well on OOD data, both adversarial and benign. Given an OOD input $(x, \mathbb{P}_{ood})$ and a model f trained with the IID assumptions on a training data X_{train} sampled from distribution $\mathbb{P}_{train}$, Out2In's intuition is to detect and perform *OOD to in-distribution mapping* of an OOD sample x drawn from $\mathbb{P}_{ood}$ to the training distribution $\mathbb{P}_{train}$ of the model. We denote the OOD to in-distribution mapping scheme as $\mathcal{M}$ and the act of mapping an OOD sample x to its in-distribution equivalent as $\tilde{x} = \mathcal{M}(x)$. Although OOD to in-distribution mapping is adaptable to other settings, we focus on image classification. We note that, despite the technical resemblance, the purpose of this work completely differs from *input transformation* defenses that use variational autoencoders (VAEs) [140] to defend against adversarial examples. More precisely, Out2In tackles the OOD generalization problem while it is particularly resilient against adversarial examples.

To implement $\mathcal{M}$, we leverage recent advances in *Image-to-Image translation* [142, 67] that aim to transfer images from a source distribution to a target distribution while preserving content representations. To this end, we explore alternative designs of $\mathcal{M}$ to map an OOD sample to its in-distribution equivalent that is correctly classified by f (in effect f is robust to benign or adversary-induced distributional shifts). In §7.2, we thoroughly describe the components of Out2In, including the need for an OOD detector to put apart OOD and IID inputs. We note that previous efforts in the OOD generalization literature (e.g., [76, 8]) may possibly be adapted to also achieve generalization on adversarial examples. However, in this work, we choose to leverage the impressive performance of Image-to-Image translation methods in a wide range of applications (e.g., generating photographs from sketches [108], from attribute and semantic layouts [70], converting horses to zebras, etc) to perform in-distribution mapping of OOD samples, particularly adversarial examples.

We extensively evaluate Out2In across multiple benchmark datasets (MNIST, CIFAR10, ImageNet), multiple reference attacks (FGSM [55], PGD [89], C&W [35], SPSA [129]), and alternative designs of $\mathcal{M}$. When deployed as defense, Out2In outperforms prior related defenses by a margin of $\approx 27.64\%$ on MNIST, $\approx 40.25\%$ on CIFAR10, and $\approx 40.23\%$ on ImageNet, averaged across all studied attacks. All this is achieved while preserving the *exact* original accuracy on benign data. Furthermore, it generalizes well on non-adversarial OOD examples. Specifically, it improves the accuracy of a CIFAR10 model on OOD *darker images* ($\mathbb{P}_{ood} = \mathbb{P}_{dark}$) by a margin of 35.64% and that of an ImageNet model on OOD *sharper images* ($\mathbb{P}_{ood} = \mathbb{P}_{sharp}$) by a margin of 34.16%.

In summary, we make the following main contributions:

1. We empirically verify that the adversarial examples problem is a special case of the OOD generalization problem.

2. We propose Out2In, an OOD Generalization approach able to defend against adversarial examples, while also generalizing on non-adversarial OOD inputs.

3. We propose the first defense that preserves *the exact* accuracy on benign data.

Our code is available at `https://github.com/OOD2IID/Out2In`, along with our pre-trained models and detailed configurations of different experiments.

6.2 Background

6.2.1 Image-to-Image Translation

Overview: In *Image-to-Image Translation* the goal is to map an input image to an output image that represents its translation to a different distribution [142]. The emergence of GANs [54] inspired several GAN-based approaches to build accurate models that learn the Image-to-Image translation. The first class of approaches are supervised methods that require a *paired* training set where each image from the source distribution *(X)* has a corresponding image in the target distribution *($\tilde{X}$)* [86, 67]. One of the most notable works is *pix2pix* which uses a conditional GAN (cGAN). The generation of target images (translated) is conditional on a given input image [67]. *pix2pix* is generic and can be applied to different tasks (e.g., generating photographs from sketches [108], from attribute and semantic layouts [70]). However, obtaining paired training data can be difficult and expensive for some applications where datasets are not largely available (e.g., semantic segmentation [38]). Consequently, *unpaired* Image-to-Image translation ideas emerged (e.g., cycleGAN [142], CoGAN [85], Auto-Encoding Variational Bayes [84]), where the goal is to relate two data domains: X and $\tilde{X}$. CycleGAN [142] builds on *pix2pix* by proposing *cycle consistency loss* as a way of using transitivity to supervise training.

Pix2Pix [67]: Pix2Pix is a fully supervised paired image-to-image translation approach and it employs a conditional GAN (cGAN) architecture that requires specifying a generator model $G : X \rightarrow \tilde{X}$, discriminator model D, and model optimization procedure. Unlike the traditional GAN, the generator G does not take a point from the latent space as input. Instead, noise is provided only in form of dropout applied on several layers of the generator at both training and testing time. The generator model is a U-net that takes as an input an image from the source domain X and is trained to return a translated image in the target domain $\tilde{X}$. A U-net is similar to an encoder-decoder model, it involves down-sampling to a bottleneck and up-sampling again to an output image. U-net is a convolutional network for image segmentation [103], that is particularly suitable for experiments like Cityscapes labels2photos, and Edges2photo. In Out2In, we consider a 6-block ResNet generator instead, given that ResNet models have been particularly successful on ImageNet and CIFAR10 [60]. The discriminator model takes an image from the source domain and an image from the target domain and distinguishes whether the image from the target domain is a real translation or a generated version (fake) of the source image.

CycleGAN [142]: CycleGAN is an unpaired image-to-image translation approach. It trains two generators, $G : X \rightarrow \tilde{X}$ and $F : \tilde{X} \rightarrow X$, using a forward cycle-consistency loss: $F(G(x)) \approx x$ and a backward cycle-consistency loss: $G(F(\tilde{x})) \approx \tilde{x}$. The cycle-consistency losses are used to further reduce the space of possible mapping functions, by eliminating the ones that are not cycle-consistent. As a result, for each image x from domain X, the image translation cycle should be able to bring x back to the original image (i.e., forward cycle consistency). Similarly, G and F should also satisfy backward cycle consistency. Additionally, two adversarial discriminators D_X and $D_{\tilde{X}}$ are introduced where D_X distinguishes between images $\{x\}$ and translated images $\{F(\tilde{x})\}$ and $D_{\tilde{X}}$ distinguishes between images $\{\tilde{x}\}$ and translated images $\{G(x)\}$. The objective is the sum of two types of terms: *adversarial losses* (L_{GAN}) for matching the distribution of generated images to the data distribution in the target domain; and *cycle consistency losses (L_{cyc})* to prevent the learned mappings G and F from contradicting each other. Formally, the full objective is defined as follows:

$$\begin{aligned} L(G, F, D_X, D_{\tilde{X}}) = L_{GAN}(G, D_{\tilde{X}}, X, \tilde{X}) + \\ L_{GAN}(F, D_X, \tilde{X}, X) + \lambda L_{cyc}(G, F) \end{aligned} \tag{6.1}$$

where,

- $L_{GAN}(G, D_{\tilde{x}}, X, \tilde{X}) = \mathbb{E}_{\tilde{x} \sim \mathbb{P}_{data}(\tilde{x})}[logD_{\tilde{X}}(\tilde{x})]$
  $+ \mathbb{E}_{x \sim \mathbb{P}_{data}(x)}[log(1 - D_{\tilde{X}}(G(x))]$
- $L_{cyc}(G, F) = \mathbb{E}_{x \sim \mathbb{P}_{data}(x)}[||F(G(x)) - x||_1]$
  $+ \mathbb{E}_{\tilde{x} \sim \mathbb{P}_{data}(\tilde{x})}[||G(F(\tilde{x})) - \tilde{x}||_1]$
- λ controls the relative importance of the two objectives.

CycleGAN uses 9 blocks of ResNet models as generators G and F and a PatchGAN classifier as discriminators (D_G and D_F). It classifies whether 70×70 overlapping patches are real or fake. Such a patch-level discriminator architecture has fewer parameters than a full-image discriminator and works well on arbitrarily sized images in a fully convolutional fashion [142].

6.3 Motivation and Threat Model

6.3.1 Motivation

Our goal is to make ML models resilient to adversarial inputs by addressing a more general problem: *the OOD generalization problem*. Hence, the main motivation of Out2In is that adversarial examples include malicious feature perturbations that causes distribution shifts. By performing FGSM and PGD attacks on CIFAR10 and MNIST inputs we verify that evasion attacks actually produce OOD malicious samples. Particularly, we compute how distant FGSM, PGD and benign samples are from the training data distribution, using a state-of-the-art OOD detector (SSD [109], explained in 5.1.1). Figure 6.1 illustrates the different density ranges of *OOD distance* returned by the SSD model for each data distribution (i.e., benign, FGSM and PGD). We observe that the benign test data is very close to the training distribution, while the majority of adversarial data tend to be much distant from training distribution due to adversarial perturbations. These findings confirm our intuition that our quest to make ML models generalize on adversarial examples is linked to the wider problem of generalizing ML models on OOD data.

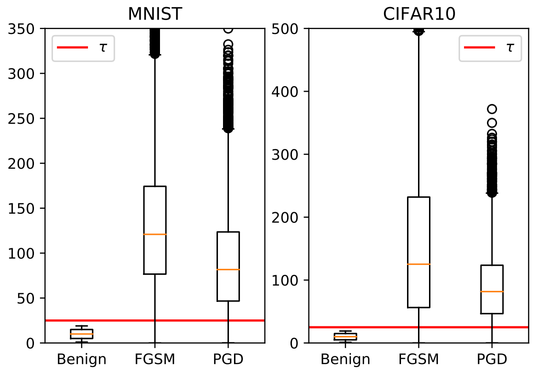


Figure 6.1: OOD scores of adversarial and benign data on MNIST and CIFAR10. τ is the adversarial threshold that we define in §7.2.

6.3.2 Threat Model

Our work is developed and tested with respect to a threat model that governs the target ML model and the assumptions about the adversary's goals and capabilities.

Target ML model: As a target ML model, we consider a supervised classification model f trained under the IID assumption to accurately classify input images within a limited set of labels Y. Thus, it does not deal with unrelated data with labels $y \notin Y$. Out2In does not make any changes on f, it rather ensures that a received input x is within the training distribution of f, independently of what f is intended to do on x. Consequently, Out2In is applicable on any ML task in the image domain (e.g., segmentation, object detection, etc).

Adversarial Goals and Capabilities: The main goal is to evade f and confuse it to make wrong classification on a visibly legitimate image x. To that end, the adversary performs adversarial manipulation on x by carefully adding noise $x + \delta$. The added perturbation has to preserve the original visual human interpretation of the image x. For instance, if x is an image of a cat, $x+\delta$ has to be visually interpretable as an image of the same cat. Consequently, the studied evasion attacks are all subject to a perturbation limit $\epsilon > 0$, that we choose to satisfy the condition $||\delta||_p < \epsilon$. When deployed as a defense, Out2In is anticipated to face strong adversaries that have access to the target model f and knowledge about its architecture and training data, and weaker but more realistic attackers that have no access to the model and can only attack it as a black-box prediction API.

6.4 Out2In Design

6.4.1 Overview

We consider a model f trained under the IID assumption on a training data (X_{train}, $\mathbb{P}_{train}$). Given an OOD test sample $(x, \mathbb{P}_{ood})$, where $\mathbb{P}_{ood} \neq \mathbb{P}_{train}$ (or $\mathbb{P}_{ood} \neq \mathbb{P}_{test}$), f is likely to produce an incorrect prediction on x, which makes it unsuitable for real-word and high-stakes tasks. We denote the data distribution used to independently draw training and test samples by $\mathbb{P}_{data} = \mathbb{P}_{train} = \mathbb{P}_{test}$. Thus, an unknown test input $(x, \mathbb{P}_{unknown})$ can be either OOD ($\mathbb{P}_{unknown} = \mathbb{P}_{ood} \neq \mathbb{P}_{data}$) or IID ($\mathbb{P}_{unknown} = \mathbb{P}_{data}$).

As illustrated in Figure 6.2, Out2In involves two key steps: 1) detecting whether or not an input $(x, \mathbb{P}_{unknown})$ is OOD and 2) mapping an OOD input $(x, \mathbb{P}_{ood})$ to its in-distribution equivalent. In particular, a test input $(x, \mathbb{P}_{unknown})$ is first received by an *OOD detector* that decides whether or not it is an OOD sample. If x is an OOD sample $(x, \mathbb{P}_{ood})$, then $\mathcal{M}$ intervenes to convert x into its in-distribution equivalent $(\tilde{x}, \mathbb{P}_{data})$ that can be correctly recognized by f with a confidence as high as the model's generalization power on samples from $\mathbb{P}_{test}$. Otherwise, if x turns out to be already drawn from the same data distribution $\mathbb{P}_{data}$, f computes its prediction as in the IID case.

OOD Detection: To rule out in-distribution samples from the OOD to in-distribution mapping goal, we leverage *SSD* [109], the current state-of-the-art OOD detection method (introduced in §5.1.1). SSD trains a *self-supervised outlier detector* through learning a feature representation of $\mathbb{P}_{data}$. SSD only uses the training data that f was trained on, it does not require training on OOD data, including adversarial data. Given an input sample drawn from $\mathbb{P}_{unknown}$, SSD returns what we call an *OOD score* (denoted $S_{ood}(x)$): it is the distance of x from $\mathbb{P}_{data}$. In order to effectively deploy SSD into our framework, a threshold definition (τ) is necessary to separate between queries that exhibit tolerable distribution shift (i.e., $S_{ood}(x) < \tau$) from those that are far away from $\mathbb{P}_{data}$ (i.e., $S_{ood}(x) \geq \tau$). Although most of adversarial examples are OOD, when Out2In is deployed to defend against adversarial examples the threshold is carefully selected to re-purpose the OOD detector to be an adversarial input detector regardless of whether an input is OOD or not (red line in Figure 6.1). Hence, in this case, the OOD score can be called adversarial score (i.e., S_{adv}). This allows benign input $(x, \mathbb{P}_{data})$ to be directly classified by f, without considering any defense measures that could sacrifice the model's accuracy on benign data, while taking defensive measures on adversarial examples. As a result, in adversarial settings, the OOD detector plays the role of an adversarial examples detector that guides Out2In to maintain the exact accuracy on benign data. SSD reaches a detection accuracy $> 99\%$ on benign OOD data and adversarial examples. In Section 6.5.4, we empirically confirm that, due to the OOD detector, Out2In does not sacrifice accuracy on benign data.

OOD to In-Distribution Mapping $\mathcal{M}$: Given an OOD sample $(x, \mathbb{P}_{ood})$, the design of $\mathcal{M}$ is

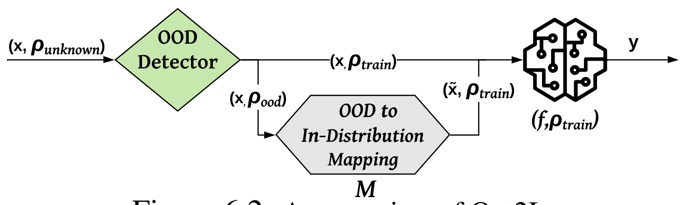


Figure 6.2: An overview of Out2In.

dictated by the data domain. As stated earlier, the purpose of $\mathcal{M}$ is to translate an OOD sample $(x, \mathbb{P}_{ood})$ to its in-distribution equivalent $(\tilde{x}, \mathbb{P}_{data})$, where $\tilde{x} = \mathcal{M}(x)$. To this end, *Image-to-Image translation* methods (Section 6.2.1) are naturally suitable for realizing $\mathcal{M}$, because our goal here is inline with that of translating an image from a source distribution to its equivalent in a target distribution. In the following sections, we explore design alternatives of $\mathcal{M}$. In the remainder of this chapter, a translator $\mathcal{M}$ will have the form **[Src-Dist]2[Target-Dist]**, where **Src-Dist** corresponds to source/input distribution and **Target-Dist** to target/output distribution. For adversarial examples, Src-Dist represents the distribution of adversarial examples crafted using attacks such as FGSM, PGD, C&W, or SPSA. On the other hand, Target-Dist is $\mathbb{P}_{data}$ of the model at hand. In the following, we examine two alternative approaches to realize $\mathcal{M}$: (i) a standalone translator and (ii) ensemble of translators.

6.4.2 Standalone Translator

We first discuss the case where we consider only one translator: $\mathcal{M} = \mathcal{M}_1$ = Src-Domain2Target-Domain.

Source Domain = One Distribution: A standalone translator is trained to solely translate inputs from a specific source distribution (e.g., $\mathbb{P}_{PGD}$) into $\mathbb{P}_{data}$. In practice, an input query may be drawn from any unknown distribution $\mathbb{P}_{unknown}$. As a result, there is no direct approach to selecting the source distribution to train $\mathcal{M}$. For instance, if deployed as a defense against adversarial examples, there is no guarantee that a standalone translator, trained to map FGSM examples into in-distribution samples (i.e., FGSM2Benign), is able to additionally translate other attack examples (e.g., PGD, C&W). This issue is similarly faced by adversarial training techniques (e.g., [126]). For instance, a model trained on FGSM samples is not necessarily robust against samples from other attacks (e.g., C&W). Hence, in addition to exhaustively examining different attacks as a source domain (§6.5.2), in the following sections, we discuss other designs to mitigate this issue.

Source Domain = Mixture of Distributions: Intuitively, one way to enhance the translation capabilities of a standalone translator $\mathcal{M} : X \to \tilde{X}$ on different distributions is to train it on a mixture of distributions as a source domain, i.e., source domain $X = \cup_i(X_i, \mathbb{P}_i)$. For instance, in adversarial settings, we consider as a source domain $X = \cup_i(X_i, \mathbb{P}_{Attack_i})$, composed of the union of distinct subsets X_i from different attack distributions $\mathbb{P}_{Attack_i}$ (i.e., $Attack_i \neq Attack_j$ if $i \neq j$). We note that depending on the defender's preference and priorities (e.g., the strength of attacks), the proportion of each attack distribution may vary. In particular, each attack distribution $Attack_i$ is represented by a subset of adversarial examples in the source domain X.

Although this design is likely to result in a better generalization of the translation capabilities, it might also decrease the performance compared to separate standalone translators trained on each attack as one source distribution (e.g., FGSM2Benign, PGD2Benign, etc). Thus, we further investigate another alternative design that is likely to maintain the performance of individual translators while also generalizing the translation capabilities to any attack. We call it *Ensemble of Translators*.

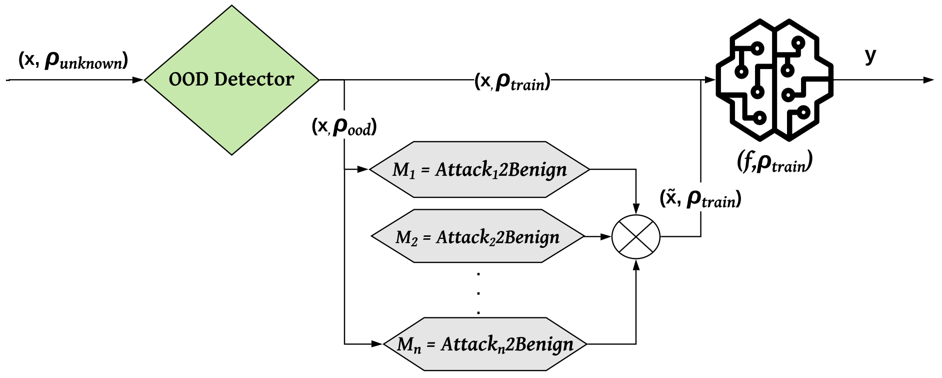
6.4.3 Ensemble of Translators

Multiple Standalone Translators: For this design, multiple standalone translators are deployed together. In adversarial settings, each standalone translator is trained on a source domain of only one attack distribution. Consequently, the overall in-distribution mapping model is defined as $\mathcal{M} = (\mathcal{M}_1, ..., \mathcal{M}_n)$, where each translator $\mathcal{M}_i$ is trained to translate test samples from $\mathbb{P}_{Attack_i}$ into $\mathbb{P}_{data}$. As illustrated in Figure 6.3, given an adversarial query $(x, \mathbb{P}_{adv})$, each scheme $\mathcal{M}_i$ generates a candidate translation $\tilde{x}_i$. Hence, a selection approach is needed to pick the best input translation and feed it to f for classification. Two selection approaches stand out:
Majority Vote: First, we consider the majority vote approach. Particularly, f predicts the labels of all possible input translations $\{\tilde{x}_i\}_{i \leq n}$. The label $y_i = f(\tilde{x}_i)$ with the highest number of occurrences across all predictions is returned as the final prediction.
Highest Confidence: Another method is to pick the predicted label that has the highest confidence of the model f, i.e., prediction probability $P_f(y)$. For each possible input $\tilde{x}_i$, f attributes a prediction probability to each label $y \in Y$. The selected label is the one that has the highest prediction probability, in total across all inputs $\{\tilde{x}_i\}_{i \leq n}$. Formally, $y = \max_{y \in Y} \{\sum_{\{\tilde{x}_i\}_{i \leq n}} P_{f|\tilde{x}_i}(y)\}$.

We note that the *ensemble translators* design can be extended to, additionally, include other translators trained to translate *non-adversarial* OOD samples $(x, \mathbb{P}_{OOD})$ where $\mathbb{P}_{OOD} \neq \mathbb{P}_{adv}$ (e.g., blur2clear, sharp2normal, etc). Hence, realizing the goal of OOD generalization, on adversarial and non-adversarial inputs.

6.4.4 Translator Model Architecture

We leverage Image-to-Image translation methods as tools to implement $\mathcal{M}$. We particularly study CycleGAN [142] and pix2pix [67], that currently are Image-to-Image translation benchmarks.



(x, $\rho_{unknown}$)
OOD Detector
(x, ρ_{train})
(x, ρ_{ood})
M_1 = Attack$_1$2Benign
M_2 = Attack$_2$2Benign
M_n = Attack$_n$2Benign
($\tilde{x}$, ρ_{train})
(f, ρ_{train})
y

Figure 6.3: Out2In with ensemble of translators.

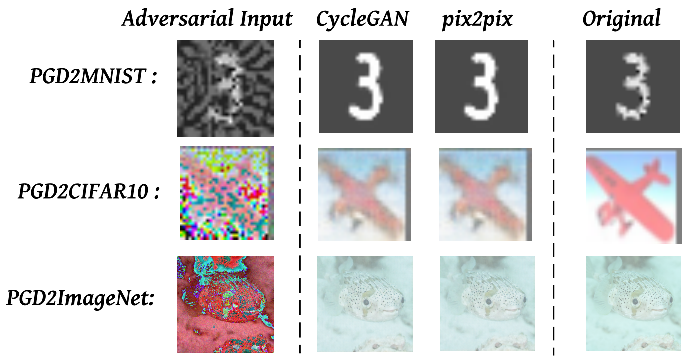


Figure 6.4: An illustration of Image-to-Image translation of PGD-adversarial examples on normalized samples, to $\mathbb{P}_{OOD}$ using CycleGAN and pix2pix on MNIST, CIFAR10, and ImageNet.

$\mathcal{M}$ = **CycleGAN:** As described in §6.2, Most Image-to-Image translation methods require a dataset comprised of *paired* examples [86, 67]. For instance, a summer2winter landscape translator model needs a large dataset of many examples of summer landscapes and the same number of their winter counterparts. Gathering paired datasets is challenging for several real-world applications. In CycleGAN [142], however, there is no need to have paired images, it can be trained using source training images X that are unrelated to the target training images $\tilde{X}$ (i.e., unpaired dataset).

As a defense against adversarial examples, Out2In can be deployed with CycleGAN as in-distribution mapping architecture by training a generator G: $(X_{adv}, \mathbb{P}_{ood}) \rightarrow (X_{benign}, \mathbb{P}_{data})$ that generates a benign version of an adversarial input. With respect to the cycle-consistency losses defined in §6.2.1, a reverse generator F: $(X_{benign}, \mathbb{P}_{data}) \rightarrow (X_{adv}, \mathbb{P}_{ood})$ is additionally trained.

$\mathcal{M}$ = **pix2pix:** We additionally cover the implementation of a supervised translator. In case paired datasets can be gathered, pix2pix could be more suitable than cycleGAN. Our adaptation to the original implementation of pix2pix is provided along with its background in §6.2.

Figure 6.4 shows three examples of translating adversarial images from the PGD attack distribution (on normalized data) to benign images close to the training distribution in three datasets

(MNIST, CIFAR10, and ImageNet) using $\mathcal{M}$ = CycleGAN = PGD2Benign, and $\mathcal{M}$ = pix2pix = PGD2Benign. For all three datasets, PGD training samples are used as the source distribution $\mathbb{P}_{PGD}$ to train $\mathcal{M} : (X, \mathbb{P}_{PGD}) \rightarrow (\tilde{X}, \mathbb{P}_{benign})$. Thus, $\mathcal{M}$ can successfully map an OOD input (PGD image in this case) to the distribution of benign data.

6.5 Evaluation

We evaluate Out2In on three image datasets across alternative designs of $\mathcal{M}$ in two OOD problem settings: adversarial and benign OOD examples.

6.5.1 Experimental Setup

Datasets: We use three benchmark image classification datasets of different scales (i.e., MNIST [80], CIFAR10 [73], and ImageNet [75]). Details of these datasets are in Appendix A.

Models: On MNIST, we train a CNN model that reaches a test accuracy of 98.96% (on 10K samples). For CIFAR10, we adopt a SimpleDLA network that reaches one of the best performances compared to the state-of-the-art accuracy of 95.19% originally proposed in [77]. For ImageNet, we use a pretrained ResNet50 model available via PyTorch [136]. On the first 100 classes of the validation data (i.e., 5K samples), it reaches an accuracy of 83.58%. Details of each model's architecture appear in our publicly available code.

Translator Model Architectures: As described in §6.4.4, we leverage *CycleGAN* for unpaired datasets and *pix2pix* for paired datasets. For MNIST, all models are trained on 500 images from the source distribution and their equivalent in the target distribution. As for CIFAR10 and ImageNet, 1000 training images in each distribution (i.e., source and target distributions) were sufficient to reach visually accurate image translation. In order to reduce the train/test time of translation models, we consider only the first 100 of the 1000 classes of ImageNet to train and test $\mathcal{M}$, f and adversarial training models.

Source Distributions: We test all the definitions of $\mathcal{M}$ discussed in §6.4.2 and §6.4.3 which include a standalone translator with a single source distribution (PGD, FGSM or SPSA), a standalone translator with a mixture of source distributions (FGSM, PGD, and C&W), and an ensemble of translators $\mathcal{M} = \{\mathcal{M}_1 = PGD2Benign, \mathcal{M}_2 = FGSM2Benign, \mathcal{M}_3 = SPSA2Benign\}$. To test Out2In against naturally OOD samples, we conduct two experiments with a standalone $\mathcal{M}$ and one experiment with an ensemble of translators. To train different designs of $\mathcal{M}$, we consider dark images and images with high sharpness as OOD source distributions. More details about each experiment are provided in §6.5.2 and §6.5.5 as we discuss the results.

Target Distribution: Inline with the purpose of our approach, the target distribution is always

$\mathbb{P}_{data}$ used to train the model f. It is one of MNIST, CIFAR10, or ImageNet training distributions. Hence, to train $\mathcal{M}$ we use a subset of the training data X_{train}.

Evaluation Metrics: Our evaluation relies on two complementary metrics, prediction *Accuracy* and *Relative Robustness*.

Accuracy: it computes the rate of correct predictions out of the total number of test samples. We use this metric to compare the model's prediction accuracy on OOD data before and after using Out2In (§6.5.5).

Relative Robustness (RR): The robustness of a ML model on adversarial data is relative to its performance on benign data. For instance, the best neural network trained on CIFAR10 can only reach a maximum of 95.19% accuracy on benign data [77]. In order to ensure a fair evaluation of robustness, we evaluate the model's performance under attack, relative to its performance before the attack. In particular, we compute a metric that compares the number of correct predictions made by f under attack with the number of correct predictions on benign data. We call this metric the *Relative Robustness (RR)*. Formally, it is defined as: $RR(\%) = \frac{\sum_{x \in X}\{f(x+\delta)=y_{true}\}}{\sum_{x \in X}\{f(x)=y_{true}\}} \times 100$,where $x \in X$ is the test sample, y_{true} is the true label of x, and δ is the attack's perturbation.

On benign data ($\delta = 0$), $RR = 100\%$. On adversarial data, if RR is close to 100%, then accuracy on adversarial data is close to accuracy on benign data which reflects high robustness. Otherwise, the model is less robust.

Studied Attacks: We use three white-box attacks (i.e., FGSM [55], PGD [89], and C&W [35]), where the adversary has knowledge of model f's architecture, and a black-box attack (i.e., SPSA [129]) to cover complementary threat models and diverse attack strengths. In a white-box setting, we use FGSM, PGD, and C&W. In a black-box setting, we use SPSA as one of the representative attacks.

We select $\epsilon = 0.3$ for MNIST and $\epsilon = 0.2$ for CIFAR10. For ImageNet, Out2In maintains its robustness for $\epsilon = 0.2$. However, we choose to follow the state-of-the-art by using a maximum of $\epsilon = 8/255 \approx 0.031$ so as to conduct a fair comparison with prior defenses [106]. For all attacks, we use $||.||_\infty$ norm except C&W which supports $||.||_2$.

Studied Defenses: So far, the closest the state-of-the-art got to OOD generalization for adversarial robustness is through *adversarial training*. An adversarially-trained model is in fact an improved model that generalizes to adversarial examples, which makes it more robust, albeit its penalty on benign accuracy. As a result, we consider adversarial training techniques as benchmark defenses to compare Out2In with the state-of-the-art.

Adversarial training: A training [106, 78] scheme of exposing the model f to adversarial examples during the training phase in order to learn the correct labels of adversarial test samples. It has been considered as one of the most effective and practical defenses against evasion attacks, but with a caveat of sacrificing accuracy on clean data.

Ensemble Adversarial training: Another technique of adversarial training is to train the model

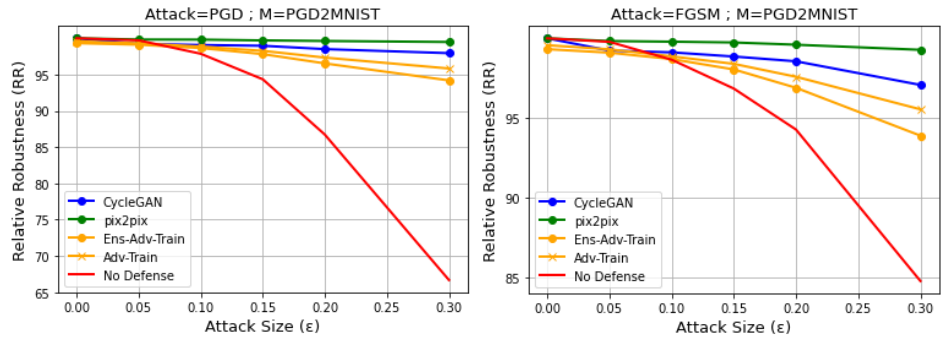


Figure 6.5: PGD (left) and FGSM (right) against Out2In with $\mathcal{M}$=PGD2MNIST and adversarial training techniques.

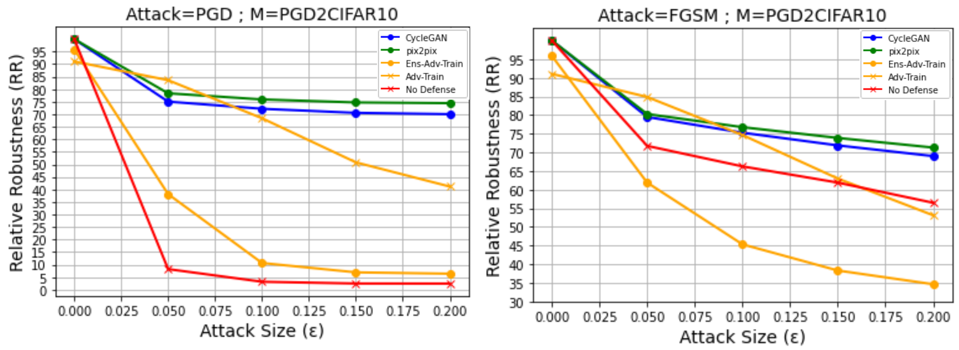


Figure 6.6: PGD (left) and FGSM (right) against Out2In with $\mathcal{M}$=PGD2CIFAR10 and adversarial training techniques.

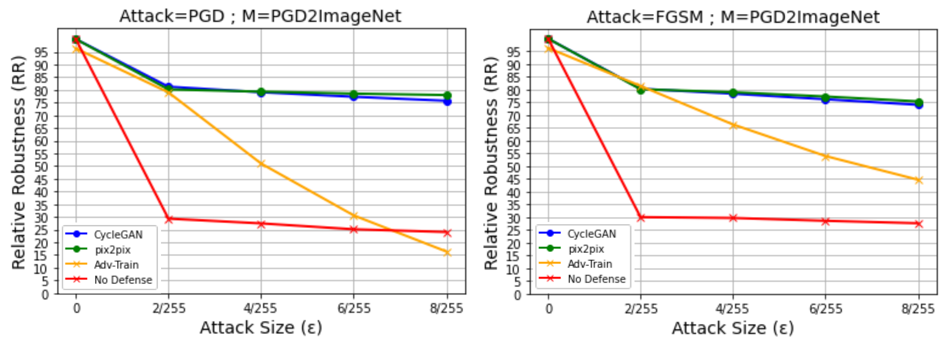


Figure 6.7: PGD (left) and FGSM (right) against Out2In with $\mathcal{M}$=PGD2ImageNet and adversarial training techniques.

on adversarial examples that are generated using a set of different models as target models. It is more effective against multiple-step attacks, especially black-box attacks [126]. In all subsequent figures, tables, and discussions we use "Adv-Train" for adversarial training and "Ens-Adv-Train" for ensemble adversarial training.

Notations: In all the next figures and tables, we use the abbreviated notation "MIX" to refer to the source distribution of a mixture of attacks, "Ens-$\mathcal{M}$-MV" to denote ensemble translators with majority vote, and "Ens-$\mathcal{M}$-HC" to denote ensemble translators with highest prediction confidence.

Evaluation plan: First, we compare Out2In with adversarial training. For a fair comparison, we perform adversarial training and Out2In using *the same training attack distribution.* To that end, we select PGD as it has a fairly reasonable evasion success compared to FGSM and a lower performance overhead compared to SPSA and C&W [89] (§6.5.2). Second, we compare between different designs of Out2In (§6.5.3). Additionally, we highlight the utility of OOD detection in Out2In robustness and we evaluate its performance (§6.5.5). Finally, in §6.5.4 we evaluate Out2In when deployed to generalize on natural distribution shifts (e.g., darker or sharper images). In Appendix B and C, we also discuss how to counter adaptive attack against Out2In and extend it to other domains.

6.5.2 Comparison with Benchmark Defenses

For this experiment, we consider the case where the defender deploys a standalone translator trained to translate PGD data into in-distribution (e.g., $\mathcal{M}$=PGD2MNIST, $\mathcal{M}$=PGD2CIFAR10, or $\mathcal{M}$=PGD2ImageNet). For each dataset, we use both CycleGAN and pix2pix. To ensure a fair comparison with the benchmark adversarial training techniques, we similarly use PGD attack to perform Adv-Train and ens-adv-train on the target model f. Figures 6.5, 6.6, and 6.7 show the RR values using Out2In compared to adversarial training techniques against PGD (left) and FGSM (right) attacks, respectively, for MNIST, CIFAR10, and ImageNet. We additionally refer to Table 6.1 to examine the RR values of different defenses against C&W and SPSA.

Robustness against PGD and FGSM: Figure 6.5 shows that, on average, unlike Adv-Train ($RR \approx 95\%$) and Ens-Adv-Train ($RR \approx 93\%$), Out2In with $\mathcal{M}$=PGD2MNIST maintains $RR > 98\%$ on PGD and FGSM attacks, even for higher attack size ($\epsilon = 0.3$). These results suggest that using Out2In, the accuracy of f under attack is almost the same as its accuracy on benign data. Comparable findings are observed on CIFAR10 and ImageNet (Figures 6.6 and 6.7). In particular, with $\mathcal{M}$=PGD2CIFAR10 and $\mathcal{M}$=PGD2ImageNet, Out2In outperforms adversarial training techniques by a significant margin. More precisely, considering the worst-case scenario of both attacks (i.e., CIFAR10: $\epsilon = 0.2$, ImageNet: $\epsilon = 8/255 \approx 0.031$), Out2In outperforms Adv-Train and Ens-Adv-Train, respectively by a margin of 25.72% and 49.19% on CIFAR10, averaged over FGSM and PGD. It reaches $RR \approx 75\%$ (Figure 6.6). Additionally, it beats Adv-Train on ImageNet by a margin of 46.14% averaged over FGSM and PGD to reach $RR = 77.96\%$ against PGD and $RR = 75.44\%$ against FGSM (Figure 6.7 and Table 6.1). We note that for ImageNet we consider only Adv-Train as benchmark defense due to the high training time of adversarial training techniques (especially Ens-Adv-Train).

Takeaway 1: Out2In generalizes to adversarial examples far better than previous adversarial generalization methods, which suggests its suitability for real-world applications.

			No Attack	FGSM	PGD	C&W	SPSA
1	**No Defense**	MNIST	100%	84.79%	66.69%	0.07%	85.13%
2		CIFAR10	100%	56.48%	2.41%	0.01%	47.70%
3		ImageNet	100%	27.63%	24.06%	0.0%	65.79%
4	**Ens-Adv-Train**	MNIST	99.32%	93.90%	94.19%	1.99%	94.14%
5		CIFAR10	95.87%	34.63%	6.35%	0.71%	41.11%
6	**Adv-Train**	MNIST	99.57%	95.55%	95.83%	0.86%	95.59%
7		CIFAR10	91.06%	53.13%	41.15%	0.52%	68.41%
8		ImageNet	96.33%	44.69%	16.39%	0.0%	82.01%
9	**Out2In with CycleGAN**	$\mathcal{M}$ = PGD2MNIST	100%	97.08%	97.94%	98.20%	97.44%
10		$\mathcal{M}$ = PGD2CIFAR10	100%	69.00%	70.05%	48.90%	64.67%
11		$\mathcal{M}$ = PGD2ImageNet	100%	74.06%	75.71%	**46.03%**	80.27%
12	**Out2In with Pix2Pix**	$\mathcal{M}$ = PGD2MNIST	100%	**99.28%**	**99.48%**	**99.12%**	**99.27%**
13		$\mathcal{M}$ = PGD2CIFAR10	100%	71.29%	74.43%	49.50%	67.58%
14		$\mathcal{M}$ = SPSA2CIFAR10	100%	71.26%	74.37%	45.58%	72.69%
15		$\mathcal{M}$ = FGSM2CIFAR10	100%	**74.96%**	**78.95%**	47.44%	**72.70%**
16		$\mathcal{M}$ = MIX2CIFAR10	100%	60.41%	60.51%	**68.69%**	56.54%
17		$\mathcal{M}$ = Ens-$\mathcal{M}$-MV (CIFAR10)	100%	72.72%	76.48%	48.44%	70.10%
18		$\mathcal{M}$ = Ens-$\mathcal{M}$-HC (CIFAR10)	100%	73.53%	77.33%	49.81%	70.38%
19		$\mathcal{M}$ = PGD2ImageNet	100%	**75.40%**	**77.96%**	25.40%	**83.10%**

Best result per attack on:
🟦 MNIST 🟩 CIFAR10 🟫 ImageNet

Table 6.1: Relative Robustness (RR) of variations of $\mathcal{M}$ and adversarial training (Adv-Train) and ensemble adversarial training (Ens-Adv-Train) for MNIST, CIFAR10, and ImageNet. Attack_size $\epsilon = 0.3$ for MNIST, 0.2 for CIFAR10, and $8/255 \approx 0.031$ for ImageNet. All attacks are performed with $||.||_\infty$ except C&W (with $||.||_2$).

Accuracy preservation on benign data: On top of its convincing robustness over benchmark defenses, Figures 6.5, 6.6, and 6.7 show that our defense does not sacrifice accuracy on benign data ($RR = 100\%$ for $\epsilon = 0$), unlike adversarial training techniques that sacrifice the accuracy on benign data by an average margin (over Adv-Train and Ens-Adv-Train) of $\approx 1\%$ on MNIST, 6.53% on CIFAR10, and 3.67% on ImageNet. This preservation of benign accuracy is attributed to the adversarial input detector that successfully differentiates between adversarial and benign test data, and feeds benign (in-distribution) data directly to f without involving $\mathcal{M}$. Additional experimental insights on the OOD detection performance are discussed later in §6.5.4.

Takeaway 2: Unlike prior methods, Out2In generalizes to adversarial examples without penalizing accuracy on benign test data.

CycleGAN vs. pix2pix: Although both methods allow Out2In to stand against adversarial examples and surpass the state-of-the-art results, we note that, on all three datasets, pix2pix is more effective than CycleGAN in the translation of OOD examples to in-distribution equivalents. It outperforms CycleGAN by an average margin of $\approx 2\%$, $\approx 5\%$, and $\approx 3\%$, respectively, on MNIST, CIFAR10, and ImageNet, over both attacks (i.e., FGSM and PGD). We recall that pix2pix is a supervised learning approach, while CycleGAN is unsupervised (i.e., relies on cycle consistency) which explains the observed advantage of pix2pix.

Takeaway 3: pix2pix offers better OOD to in-distribution translation success than CycleGAN on all three datasets.

Robustness against unexpected attacks (C&W and SPSA): We recall that Adv-Train and Ens-Adv-Train are performed using PGD-generated training samples. We further recall that, for a fair comparison with Out2In, we, similarly, consider only translators trained on PGD samples for this experiment. Using attacks that were not incorporated in the training of $\mathcal{M}$, we now evaluate Out2In against C&W and SPSA as "unexpected" attacks.

From Table 6.1, we focus only on results for $\mathcal{M}$ = PGD2MNIST (rows 9 and 12), $\mathcal{M}$ = PGD2CIFAR10 (rows 10 and 13), and $\mathcal{M}$ = PGD2ImageNet (rows 11 and 19). Compared to Adv-Train and/or Ens-Adv-Train results (rows 4-8), for all datasets, we confirm our previous findings on C&W and SPSA. More precisely, while adversarial training techniques are basically defeated by the C&W attack (i.e., $RR \approx 1\%$), Out2In delivers a robust model on MNIST (row 12: $RR = 99.12\%$) and a significant improvement on CIFAR10 (row 13: $RR = 49.50\%$) and ImageNet (row 11: $RR = 46.03\%$) against C&W. Similar results are observed against a black-box attack (SPSA), as our defense (MNIST: $RR = 99.27\%$ in row 12, CIFAR10: $RR = 67.58\%$ in row 13, and ImageNet: $RR = 83.10\%$ in row 19) outperforms Adv-Train (MNIST: $RR = 95.59\%$, CIFAR10: $RR = 68.41\%$, ImageNet: $RR = 82.01\%$) and Ens-Adv-Train (MNIST: $RR = 94.14\%$, CIFAR10: $RR = 41.11\%$). Finally, it is noteworthy that, although $\mathcal{M}$ is only trained to translate adversarial images generated using PGD attack subject to $||.||_\infty$, it can also translate FGSM and SPSA images. Furthermore, it is able to maintain the same translation performance on C&W test images even though they are generated subject to $||.||_2$ perturbations, especially on MNIST.

We attribute this observation to our design choices in Out2In compared to adversarial training. In particular, adversarial training changes the model f to another model f' trained to recognize the distribution of attack samples (e.g., PGD). In a white-box setting, just by re-performing the attack against f' the adversary can drastically decrease the gained robustness of adversarial training. However, in Out2In we do not retrain f. Thus, f still only recognizes the true distribution of the data. Consequently, performing attacks on f again will not lead to better evasion results against Out2In. Furthermore, even though an Image-to-Image translator PGD2DATA is trained to accurately translate PGD samples to the training distribution, it is still capable to also approximate a translation of any other sample (e.g., from another distribution) close to the training distribution. In effect, f can make a more accurate response on the approximated translation than f' can predict the true label of an adversarial example that it has not been trained on, which explains our results in Table 6.1.

Takeaway 4: On MNIST, a standalone translator trained with PGD attack distribution is sufficient to successfully generalize to other attacks such as FGSM, SPSA and C&W.

We note, however, that these findings are not consistent across the three datasets. In particular, from Table 6.1, using a standalone translator trained only on PGD (row 13), Out2In performs less against other attacks, especially on CIFAR10 (i.e., SPSA: $\approx 6.85\%$ less and C&W: $\approx 24.93\%$

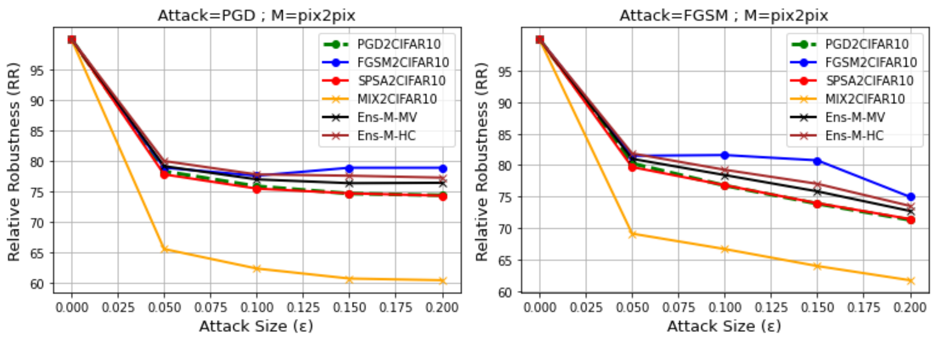


Figure 6.8: PGD (left) and FGSM (right) against alternative designs of $\mathcal{M}$.

less) compared to its performance against PGD. However, compared to adversarial training (C&W $\approx$ 41% less than their performances against PGD), Out2In generalizes much better on unseen attacks. In §6.5.3, we evaluate $\mathcal{M}$ as an ensemble of translators.

Prediction Response Time: As Out2In is composed of two models (i.e., OOD detection and $\mathcal{M}$) in addition to the target model f, one might argue about its performance overhead and deployability in practice as a prediction API. Thus, we compute its response time (in seconds) to return a prediction of an input sample. Averaged on 10K samples, we observe that Out2In takes 0.0039 seconds to respond to an MNIST input, while it takes 0.1102 seconds to respond to CIFAR10 input. Compared to adversarial training techniques, Out2In adds, respectively, 0.0033 and 0.1076 seconds to make a more accurate and robust response on MNIST and CIFAR10. We note that this small delay is caused by the OOD/adversarial detector. Particularly, without the OOD detection layer, we noticed that $\mathcal{M}$+f together have similar response time to adversarial training models. In §6.5.4, we experimentally validate the utility of the OOD detector to improve Out2In, with respect to its added overhead. Our response time results are recorded using a GPU engine: NVIDIA GeForce GTX 1060, 6GB.

6.5.3 Comparison Among Different Designs of $\mathcal{M}$

We now evaluate alternative designs of $\mathcal{M}$ with respect to the options discussed in §6.4.2 and §6.4.3. We have already established that pix2pix leads to better results than CycleGAN. Hence, for the next experiment, we focus only on $\mathcal{M}$ = pix2pix. Additionally, we exclude MNIST, since $\mathcal{M}$ = PGD2MNIST is sufficient to reach more than 99% robustness on MNIST. Specifically, we focus on CIFAR10.

Figure 6.8 shows the RR results of $\mathcal{M}$= PGD2CIFAR10, FGSM2CIFAR10, SPSA2CIFAR10, and MIX2CIFAR10 as different combinations of the source distribution for a standalone translator, and Ens-$\mathcal{M}$-HC, Ens-$\mathcal{M}$-MV as ensemble translators using, respectively, highest-confidence (HC) or majority vote (MV) prediction. MIX2CIFAR10 is trained on the union of three distributions of attacks (i.e., FGSM, PGD, and C&W) such that each attack distribution represents $\approx \frac{1}{3}$ of the training set of $\mathcal{M}$. For the ensemble translators method, we use an ensemble of PGD2CIFAR10, FGSM2CIFAR10, and SPSA2CIFAR10.

Robustness against PGD and FGSM: From Figure 6.8, compared to PGD2CIFAR10 (green line) most other translators, including $\mathcal{M}$ = Ens-$\mathcal{M}$, improve the robustness against both attacks (i.e., PGD and FGS) by a margin of $\approx 4\%$, especially for higher attack size $\epsilon > 0.05$. This result is true except for $\mathcal{M}$ =MIX2CIFAR10 where RR drops to $\approx 61\%$ compared to PGD2CIFAR10 ($RR = 74.43\%$ against PGD and $RR = 71.29\%$ against FGSM). We note that, using the highest confidence (Ens-$\mathcal{M}$-HC) prediction leads to slightly better results than using majority vote (Ens-$\mathcal{M}$-MV). The majority vote (MV) is not effective especially when the number of translators is very high, since multiple translators are equally voting for the label of an input that is most likely from an unseen distribution. However, if we consider a FGSM input, the translator that is trained to translate FGSM examples is most likely to lead to the highest confidence prediction, which leads to its selection when using HC. Additionally, it is noteworthy that using $\mathbb{P}_{FGSM}$ as source distribution to train a translator $\mathcal{M}$ is the best design to defend against gradient-based attacks on CIFAR10.

Robustness against CW and SPSA (Table 6.1): Against C&W, $\mathcal{M}$ =MIX2CIFAR10 (row 16) reaches the best robustness $RR = 68.69\%$ (row 16) compared to an average $RR \approx 48\%$ over all other designs (PGD2CIFAR10: row 13, FGSM2CIFAR10: row 15, SPSA2CIFAR10: row 14, and Ens-$\mathcal{M}$ translators: rows 17-18). This observation is explained by including C&W samples to train MIX2CIFAR10 translator. Both designs of Ens-$\mathcal{M}$ do not improve robustness against C&W compared to standalone translator designs, since only SPSA2CIFAR10, FGSM2CIFAR10, and PGD2CIFAR10 translators are adopted for Ens-$\mathcal{M}$ designs (note: no C&W2CIFAR10 translator is considered). Against SPSA, all designs similar results ($\approx 71\%$) on average, except MIX2CIFAR10 which shows $\approx -15\%$ drop in robustness.

Takeaway 5: Among alternative designs of $\mathcal{M}$, ensemble-translators design is most likely to lead to better robustness to OOD examples with enough number of translators and diverse attack distributions, while FGSM2CIFAR10 is the best standalone design.

6.5.4 The Utility of the OOD Detector for Adversarial Detection

As previously reported in [109], SSD can detect OOD samples with high accuracy ($\approx 99\%$). However, in this section, we explore to what extent our adaptation of the SSD approach is effective to filter out adversarial input from benign input. Thus we record its false positive rate (FPR) and false negative rate (FNR) in adversarial detection on FGSM, PGD and Benign input.

FPR and FNR: Examining OOD distance scores of FGSM and PGD we noticed that Figure 6.1 might have ignored very few outliers with scores closer to the training data. By computing FPR and FNR, we find that SSD makes negligible false negative rates and no false positives

			Benign	FGSM	PGD
1	**OOD detection + Ens-Adv-Train**	MNIST	**100%**	93.81%	94.32%
2		CIFAR10	**100%**	33.97%	6.92%
3	**OOD detection + Adv-Train**	MNIST	**100%**	95.89%	95.83%
4		CIFAR10	**100%**	53.13%	41.17%
5	**Out2In w/o OOD detection** (only pix2pix)	$\mathcal{M}$ = pix_PGD2MNIST	*99.88%*	**99.26%**	**99.50%**
6		$\mathcal{M}$ = FGSM2CIFAR10	*83.93%*	**75.96%**	**78.91%**

Best result per attack on:
🟦 MNIST 🟩 CIFAR10

Table 6.2: Relative robustness of other designs: OOD detection + Adversarial training and Out2In w/o OOD detection.

on MNIST and CIFAR10. More precisely, on MNIST, only 1% and 2% of FGSM and PGD respective inputs were undetected as adversarial. On CIFAR10, only 2% and 3% of FGSM and PGD respective inputs were undetected as adversarial. Furthermore, *all* benign inputs of both datasets were effectively recognized as in-distribution and excluded from $\mathcal{M}$ translation. These findings are the major reasons behind Out2In's capability to preserve the exact performance of the target ML model on benign data.

Out2In performance without OOD detection: To further accentuate on the utility of the OOD detector in the effectiveness of Out2In, we exclude it from the prediction pipeline to explore how $\mathcal{M}$ would perform without the guidance of the OOD detector. Our results reported in Table 6.2 (rows 9 and 10) suggest that Out2In no longer preserves the accuracy on benign data, since all samples, whether they are "in" or "out" of the distribution are automatically translated by $\mathcal{M}$. However, Out2In keeps its previously reported high robustness against evasion attacks, since the absence of OOD detector does not affect the prediction pipeline of adversarial examples within Out2In.

Adversarial training guided by OOD detection: Another intriguing experiment is to guide adversarial training by the OOD detector. We deploy a defense pipeline that alternates between the target model f and the adversarially trained model f', according to whether an input is adversarial (i.e., goes to f') or benign (i.e., goes to f). Table 6.2 (rows 5-8) shows that deploying adversarial training models within such pipeline can also preserve the accuracy on benign data while introducing the robustness of adversarial training. However, we note that, even in this setting, Out2In without OOD detection is still more robust than adversarial training with OOD detection against evasion attacks.

OOD detection overhead: The contribution of the OOD detector to Out2In comes with a small additional overhead. We notice that Out2In is slightly slower to respond to test inputs when

	CIFAR10			ImageNet	
Test Distribution	**Original**	**Darker**	**Sharper**	**Original**	**Sharper**
Baseline model f	95.19%	50.33%	55.21%	83.58%	47.18%
Out2In using $\mathcal{M}$=sharp2normal	95.19%	-	**87.09%**	83.58%	**81.34%**
Out2In using $\mathcal{M}$=Ens-$\mathcal{M}$-HC	95.19%	85.12%	86.95%	-	-
Out2In using $\mathcal{M}$ =dark2clear	95.19%	**85.97%**	-	-	-

Table 6.3: Accuracy on benign OOD test data on CIFAR10 (darker and sharper images) and ImageNet (sharper images). "-" represents an unrelated experiment.

deployed with OOD detection module. More precisely, we observe, an additional 0.0029 seconds is taken by the SSD detector to classify an MNIST sample and 0.1073 seconds for a CIFAR10 sample classification made by a more complex SSD model architecture.

Takeaway 6: While it incurs a tolerable delay on response time, OOD detection plays a major role to preserve the accuracy on Benign data.

6.5.5 Generalization to OOD Benign Data

We now evaluate Out2In for generalization to benign OOD samples so as to account for a non-adversarial setting. To that end, we perform three experiments, two on CIFAR10 and the other on ImageNet. On CIFAR10, the distribution shift is caused by changes in *brightness* and *sharpness* of test data, while on ImageNet changes in *sharpness* of test data is the factor. We use pix2pix as image-to-image translator (i.e., for CIFAR10: dark2clear and sharp2normal, for ImageNet: sharp2normal). Each translator is trained using 1000 images.

Experiment 1: OOD input due to lower brightness: Table 6.3 shows that the accuracy of CIFAR10 model drops from 95.19% to 50.33% on OOD dark images while using $\mathcal{M}$ =dark2clear, Out2In can keep 85.97% accuracy despite the distribution shift of test data. Furthermore, once again, Out2In does not sacrifice accuracy on original data (in-distribution) due to the use of an OOD detector as an input filter. Figure 6.9 (top row) illustrates that $\mathcal{M}$ =dark2clear effectively translates dark OOD images (Src-Dist) into its in-distribution equivalent (Target-Dist), which is very similar to the original sample (i.e., deer), which explains its effectiveness to enable more accurate classification on OOD dark data ($+35.64\%$). In conclusion, despite being trained only on clear CIFAR10 data, using Out2In, f can precisely classify OOD dark test input.

Experiment 2: OOD input due to higher sharpness: Table 6.3 shows that the accuracy of the ImageNet and CIFAR10 models on the same test set, but with higher sharpness, decreases respectively from 83.58% and 95.19% to 47.18% and 55.21%. However, using $\mathcal{M}$ = sharp2normal (illustrated in Figure 6.9: bottom row), Out2In enables an accuracy recovery to reach 81.34% and 87.09% respectively on sharper ImageNet and CIFAR10 data.

Experiment 3: OOD input includes both distributions: We consider the case where Out2In

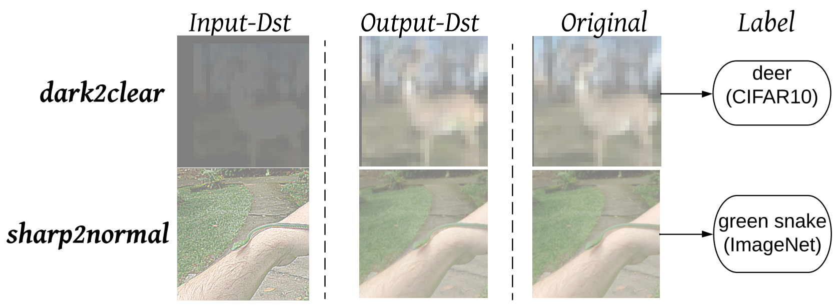


Figure 6.9: Image-to-Image translation of benign OOD to in-distribution equivalents using pix2pix on CIFAR10 (dark2clear) and ImageNet (sharp2normal).

receives OOD test data that covers different distributions (e.g., both sharp and dark). In this case, we leverage the *Ensemble translators* deployment using the highest confidence as a selection approach between $\mathcal{M}_1$ = sharp2normal and $\mathcal{M}_2$ = dark2normal. Results show that Out2In can effectively translate inputs from both distributions to add respectively $+31.74\%$ and $+34.79\%$ more classification accuracy, which is very close to the rates recorded by the standalone translators in Experiment 1 and Experiment 2 (Table 6.3).

Takeaway 7: Out2In not only generalizes on adversarial examples but also on OOD benign inputs generated because of legitimate distribution shifts.

6.6 Conclusion

ML models struggle when they face adversarial or benign OOD inputs. This work presented a framework to systematically study the cause-effect connection between the OOD problem and adversarial examples. We leverage image-to-image translation methods to build an OOD generalization framework for image classifiers. Through extensive evaluation on three benchmark datasets, we show that our approach consistently outperforms state-of-the-art defenses (adversarial training and ensemble adversarial training) in its generalization to both adversarial examples and OOD benign inputs that result from natural distribution shifts. Contrary to adversarial training-based defenses, our approach does not sacrifice accuracy on benign data. We also demonstrate the consistent performance of our approach on alternative designs of the OOD-to-IID mapping approach based on single, mixed, and ensemble of image-to-image translation-based OOD generalization models. Finally, we show the resilience of our approach to an adaptive adversary that constrains adversarial perturbations to produce IID adversarial inputs.

CHAPTER 7

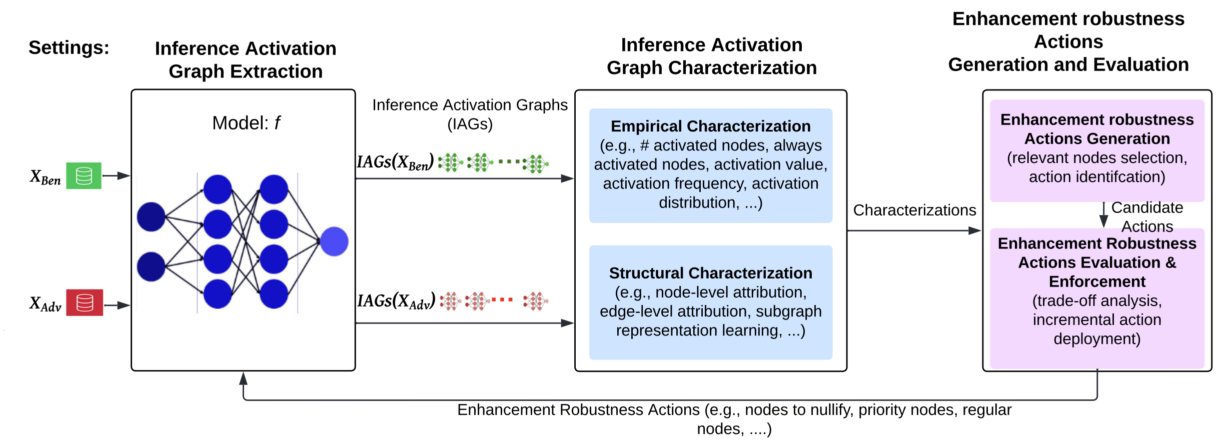
Behavioral Characterization of Neural Networks via Activation Graphs

7.1 Introduction

The impressive performance of DNNs in multiple domains [124, 105, 112, 7, 99] is largely due to their ability to learn complex patterns from vast amounts of data. However, despite their success, DNNs often exhibit puzzling and unpredictable behaviors when deployed in real-world environments. These behaviors can undermine the safety and reliability of high-stakes DNN-based systems, highlighting the need for fine-grained behavioral characterization and subsequent model repair.

Previous work aimed at characterizing DNN inference through explanation frameworks (e.g., [101, 87, 59]) offers localized, sample-centric, and retrospective attribution of a DNN's inference to contributing features. However, the fast-paced adoption of DNNs in high-stakes applications necessitates integrating capabilities into the ML pipeline that enable the capture, analysis, and characterization of DNN inference. These capabilities would in turn enable systematic analysis of *inference provenance*: quantitative (empirical) and qualitative (structural) artifacts that describe how *information flows within a DNN during inference* on a given input. In this chapter we will focus mainly on DNN characterization from security perspective under benign vs. adversarial settings.

To achieve a deeper understanding of a DNN's behavior, particularly under benign versus adversarial conditions, it is crucial to characterize its runtime behavior over a distribution of inputs using its underlying graph structure, typically a directed acyclic graph (DAG). Our hypothesis is that systematically capturing inference provenance can enhance this understanding. We introduce *Inference Activation Graphs ($IAGs$)*, where nodes represent activation values and edges correspond to model weights during inference. These IAGs are used for both empirical and structural inference characterization: empirically, by measuring activations under benign and adversarial conditions (§7.2.3); structurally, by employing a feature extraction model f_{IAG}, such as graph



Settings:
Inference Activation Graph Extraction
X_{Ben}
X_{Adv}
Model: *f*
Inference Activation Graphs (IAGs)
$IAGs(X_{Ben})$
$IAGs(X_{Adv})$
Inference Activation Graph Characterization
Empirical Characterization (e.g., # activated nodes, always activated nodes, activation value, activation frequency, activation distribution, ...)
Structural Characterization (e.g., node-level attribution, edge-level attribution, subgraph representation learning, ...)
Characterizations
Enhancement robustness Actions Generation and Evaluation
Enhancement robustness Actions Generation (relevant nodes selection, action identifcation)
Candidate Actions
Enhancement Robustness Actions Evaluation & Enforcement (trade-off analysis, incremental action deployment)
Enhancement Robustness Actions (e.g., nodes to nullify, priority nodes, regular nodes, ...)

Figure 7.1: Framework overview.

neural networks (f_{IAG}s), to learn graph representations for adversarial and benign inputs. Node- and edge-level attribution is performed to identify components of IAG that contribute to inference provenance. This IAG-driven characterization then guides the automated discovery of model repair actions for robustness enhancement, evaluated through a systematic node-level action generation mechanism (§7.2.4).

Our findings indicate that the proposed DNN graph-based characterization effectively differentiates the distinct runtime behaviors of DNNs in benign and adversarial settings. Using the characterization-based action generation mechanism, we identified IAG-driven actions that significantly enhance robustness (§8.2.1.3).

7.2 Approach

7.2.1 Overview

Figure 7.1 illustrates an overview of our approach. We consider a DNN f trained on a dataset X_{train} and tested on aversarial and benign test sets (X_{ben}, X_{adv}) to perform a classification task. The DNN is represented as a directed acyclic graph (DAG). We first extracts inference provenance graphs (IAGs) for each sample in each setting, resulting in 2 sets of $IAGs : IAGs(X_{ben})$, $IAGs(X_{adv})$. Each IAG is a subgraph of the DAG representation of f, capturing the activation graph instances of f for inputs across both settings. These IAGs represent the runtime behavior of f under different conditions.

Next, we characterize the runtime behavior of f using empirical and structural (§7.2.3) inference provenance from the IAGs across both settings. By leveraging the complementary strengths of empirical and structural inference provenance artifacts, we then identify robustness enhancement actions. These actions are then evaluated for their feasibility and effectiveness in enhancing f's reliability for the repair goal at hand (§7.2.4).

7.2.2 Graph Extraction

IAGs capture the lineage of computations during inference by illustrating how input data propagates through the layers of a DNN to produce output. In IAG, each node represents an intermediate computation or transformation, while edges indicate the flow of data between these computations. For instance, in convolutional neural networks (CNNs) used for image classification, IAG depicts the journey of pixels through convolutional, pooling, and fully connected layers, culminating in the final predicted class. Since the model itself is a directed acyclic graph (DAG), we leverage this structure to abstract the model's *computational dynamics*, analogous to a system's runtime behavior. When a model f computes an inference for input x, we extract activation values and edge weights at each layer, where node activation is typically determined by the activation function, such as ReLU (activated if non-zero). By systematically capturing and analyzing these IAGs, we reveal the computational dynamics of the DNN, providing crucial insights for effective model robustness enhancement.

7.2.3 Characterization of Inference Provenance Graphs

Empirical Characterization: First, we adopt an empirical approach that is computed based on metrics by aggregating *activation values* across different levels of the IAG: the entire network, individual layers, or specific nodes. To achieve this, we investigate several IAG-based metrics, including the number of activated nodes, consistently activated nodes, the average activation value per node, the average activation frequency per node, and the differences in activation values across different settings of f. These characterization metrics serve as empirical proxies to capture f's inference provenance.

`Number of Activated Nodes.` This metric assesses whether the number of activated nodes in IAG differs between benign and adversarial inputs. For an input $x \in X_{ben/adv}$, we calculate the number of activated nodes in its inference graph IAG`(f(x))`, and then average the results for each setting.

`Always Activated Nodes.` This metric identifies nodes that are consistently activated in all IAGs of one setting but never in another (e.g., $benign$). We also record the number of such nodes across all samples of the same class to provide a detailed per-label analysis of IAGs for each setting.

`Average Node Activation Value.` This metric quantifies the variation in node activation values across settings by computing the average activation value of each node. Comparing these averages allows for the identification of outlier IAGs within or between settings (e.g., benign and adversarial).

`Node Activation Frequency`: This metric tracks how often each node is activated across IAGs in each setting. It helps identify correlations between input transformations (e.g., adversarial perturbations) and changes in node activation frequency. For adversarial robustness, these frequencies can highlight nodes that are more frequently activated in adversarial data but less so in benign data, or vice versa.

`Activation Distribution Metrics`. To characterize inference across settings, we capture distributional differences in node activations. One metric is the dispersion index (DI), which measures how spread out or clustered node activations are across IAGs, calculated as $DI = \frac{\sigma^2}{\mu}$, where σ is the variance and μ the mean. Another metric, the entropy index, quantifies the level of disorder in activation patterns, such as the variability in adversarial IAGs.

Structural Characterization: While the empirical IAG artifacts provide quantitative insights into inference provenance, they are inherently oblivious to the underlying graph structure of IAGs. This oversight can lead to gaps in inference characterization. Therefore, to achieve a comprehensive characterization, we develop a complementary characterization scheme based on the graph structure of IAGs. For this purpose, we use a feature extraction model f_{IAG} to capture the structural characterization of IAG. The model is typically a graph neural network which allow us to analyze IAGs locally through message passing functions and globally at the graph level. f_{IAG} is trained on $\{IAG(X_{Ben})\} \cup \{IAG(X_{Adv})\}$ to predict the label of an $IAG(x)$ as f_{IAG}`(`IAG`(f(x)))`. Particularly, f_{IAG} learns IAG subgraphs to identify correlations between IAG substructures and their inference results (i.e., benign or adversarial).

Once we obtain f_{IAG}`(`IAG`(f(x)))`, next we leverage inference attribution techniques [71] for f_{IAG} model outputs. Across benign and adversarial settings, by extracting the structural attributions of the f_{IAG}'s predictions on IAGs, we identify nodes and edges that are exclusively relevant to a particular setting. Additionally, we examine the typical activation behavior (e.g., activation value) of highly attributed nodes for IAGs in one setting (e.g., benign) compared to other settings (e.g., adversarial) and how these differences among settings guide robustness enhancement.

Combining empirical and structural inference provenance characterization, we empower model deployers to utilize comprehensive IAG-based inference provenance for targeted model security enhancement.

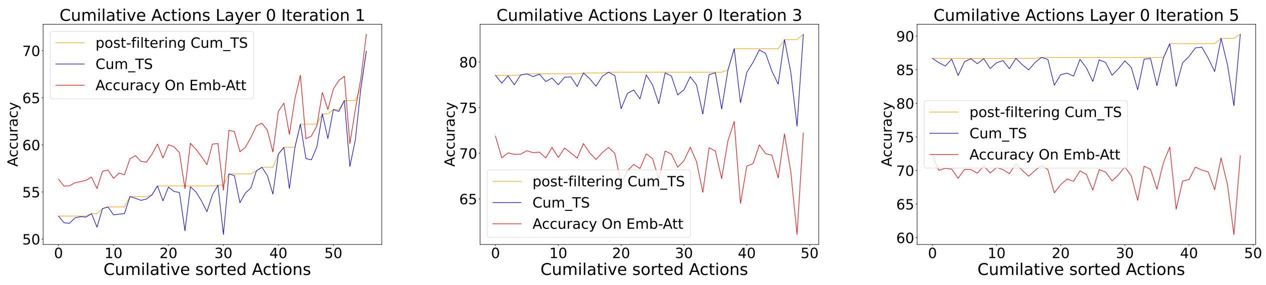
7.2.4 Robustness Enhancement Actions Generation and Selection

Actions Generation: The proposed framework systematically enhances a model's robustness to adversarial attacks by focusing on node- and edge-level actions. First, relevant nodes are identified based on their activation frequency, activation values, and attribution scores across adversarial and benign settings. Nodes that are only activated in adversarial scenarios are prioritized for repair and categorized as adversarial nodes (N_p), while those activated in both adversarial and benign settings are classified as regular candidate nodes (N_r). Nodes that are only triggered in adversarial settings are tagged for nullification (N_n). Once relevant nodes are selected, the next step involves identifying suitable repair actions. The goal is to guide each selected node's activation behavior in adversarial conditions to mimic its behavior in a benign setting. A reference activation value is computed for each node based on its behavior in benign conditions, and a sensitivity parameter α is used to control how closely the node's new activation value should align with this reference. For nodes in N_n, nullification is performed regardless of their activation distribution in the benign setting. This method ensures that adversarial influences are mitigated while maintaining the model's desirable behavior in benign settings.

Actions Selection: To systematically evaluate and enforce actions aimed at improving model robustness, each action A_i generated by the algorithm is assessed for its effectiveness in enhancing the model repair process. In the context of robustness-focused repairs, the action must improve the model's accuracy on adversarial data without sacrificing performance on benign data. To quantify this trade-off, a metric called *Tradeoff_Score* (TS) is introduced, which measures the difference between the gain in accuracy on adversarial data and the loss of accuracy on benign data. The TS for a candidate action A_i is formally defined as:

$$TS(A_i) = A_i(Ben) - A_0(Ben) + \frac{1}{n}\sum_{j=1}^{n}(A_i(Adv_j) - A_0(Adv_j))$$

Here, $A_i(Ben)$ is the model's accuracy on benign data after applying A_i, and $A_i(Adv_j)$ is the accuracy on adversarial data crafted with attack j. The baseline accuracy, without any action, is represented by A_0. A positive $TS(A_i)$ suggests that the action A_i enhances accuracy on adversarial data more than it reduces accuracy on nominal data.



(a) Step 1 (b) Step 2 (c) Step 3

Figure 7.2: Ember: cumulative actions analysis and filtering.

Guided by TS scores, actions are sorted per layer, starting with the highest TS, and incrementally applied to the model f, while tracking cumulative trade-off scores (Cum_TS), as illustrated in Figure 7.2. Actions with negative trade-offs ($TS \leq 0$) are ignored, as they lead to a greater degradation in nominal accuracy than improvements on adversarial data. The process continues iteratively, filtering out actions that negatively affect performance, until either the cumulative TS reaches its maximum or all candidate actions have been tested. This approach is illustrated in the Ember dataset case study, where the Cum_TS curve is analyzed to optimize the set of actions, ensuring improved performance on adversarial data while maintaining benign accuracy.

7.3 Evaluation

To evaluate the effectiveness of our approach, we conduct comprehensive case studies on two ML systems, one for image classification (CIFAR10 [74]) and one for malware detection (Ember [17]). These case studies encompass a variety of attacks and DNN architectures, guided by the following research questions:

•**RQ1:** How effective are the empirical and structural characterizations of IAGs in identifying distinct runtime behaviors of a DNN on benign and adversarial settings across attacks?

•**RQ2:** How effective is our approach in identifying graph-specific robustness enhancing repair actions with minimal impact on accuracy in benign setting?

To address **RQ1**, we visualize the empirical and structural characterizations introduced in §7.2.3. Our findings, detailed in §7.3.2, highlight significant differences in the runtime behavior of the studied models in adversarial and benign settings. For **RQ2**, we rely on the repair actions selection and evaluation described in §7.2.4.

7.3.1 Setup

Dataset and Models: We run the proposed framework on two models covering image classification (CIFAR10[74]) and Windows PE malware detection (Ember[17]). The models used are: CIFAR10-Resnet18 (accuracy = 87.94 %) and Ember-DNN (accuracy = 94.56%). To reduce computational overhead in model repair, we compressed each ResNet block into a single layer for ResNet18 models. This simplified graph representation allows the signal to pass through each block as if it were a single layer. While the internal behavior of the block isn't analyzed, any unwanted activity within it is reflected in the final output, which can then be addressed by our repair actions.

Attacks: For CIFAR10, we use three white-box attacks: FGSM[55], PGD [90], and APGD-DLR [39] and two black-box attacks: SPSA [128] and Square [20]. For all attacks, we maintain a perturbation bound $\epsilon \leq 0.3$. For Ember, we assume the adversary lacks knowledge about the target model but is aware of the features used to train it (e.g., API calls, DLL files). We call the attack on Ember as *Emb-Att*. As in prior work [63, 122, 12], we incrementally perform additive perturbations until the model flips its label to benign.

7.3.2 IAG-Based Characterization Results

Empirical Characterization: Figure 7.3 presents box plots of node *Activation Values* per layer, averaged across all studied samples of benign IGPs (green) and adversarial IAGs (other colors). It also shows node *Activation Frequency* per layer. We observe a clear distinction between benign and adversarial settings. More precisely, activation value and activation frequency ranges on adversarial data differ from their benign counterparts. These distinctions are more significant in Ember model, as it serves the task of binary classification. For CIFAR10, it is noteworthy that such distinction is sometimes observable across attacks, reflecting different activation patterns from one attack to another. For instance, layer 3 of CIFAR10-Resnet18 has different ranges of activation values on test samples generated with FGSM (red) and APGD-DLR (orange).

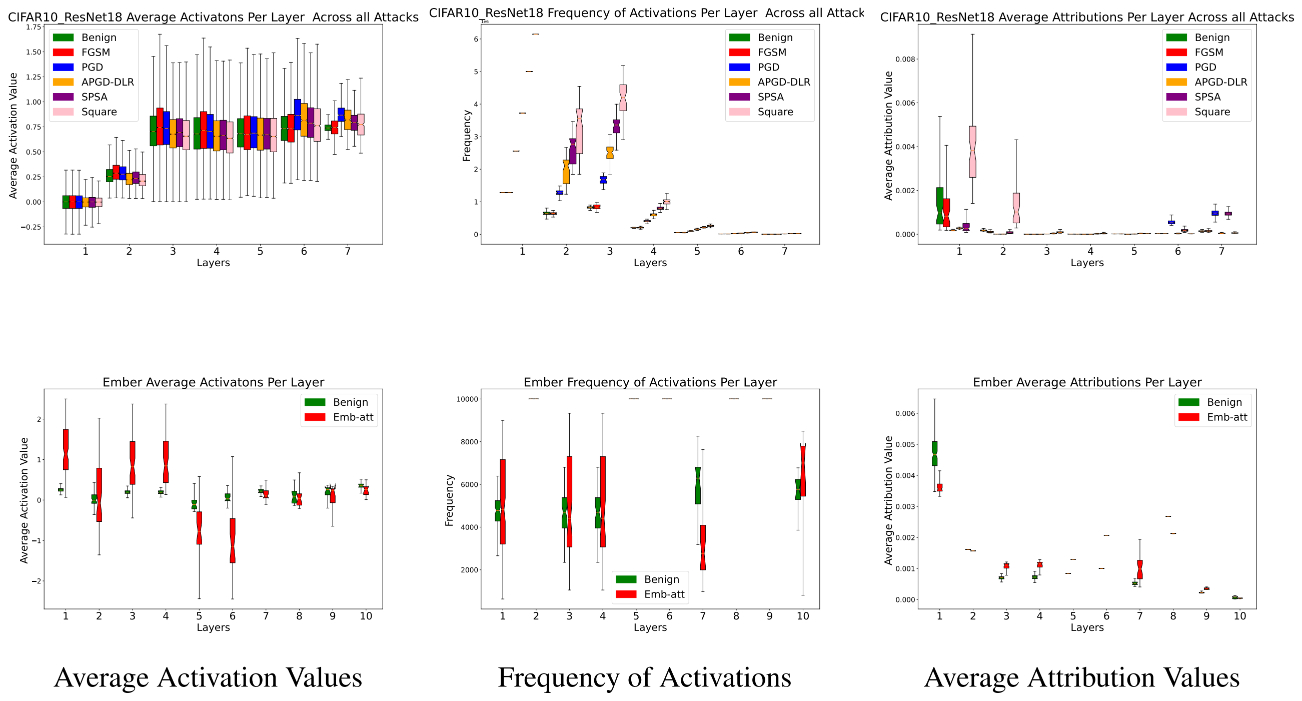
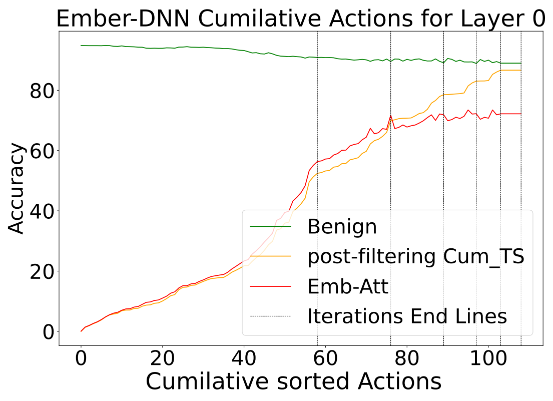


Figure 7.3: Benign vs. adversarial characterization insights. From top-to-bottom, each row refers to a studied model with plots for Average Activation per layer, Frequency of Activations per layer, and Average Attribution Values per layer, respectively.

Structural Characterization: To assess the f_{IAG}-based structural characterization at node level, we analyze the average attributions per layer across attacks (Figure 7.3). The node-level



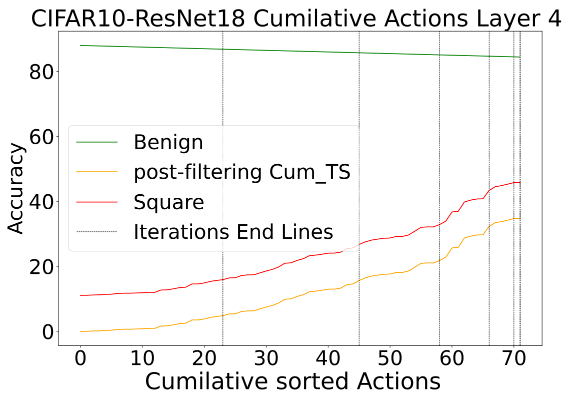


Figure 7.4: Cumulative robustness enhancement actions per model. Vertical lines denote the end of a step (iteration) in the evaluation methodology (described in §7.3.3). All plots showcase the post-filtering and ordering cumulative actions of each step, for the most performing layer.

analysis, presented in the form of box plots, reveals distinct node attribution patterns between benign and adversarial IAGs, highlighting the impact of evasion attacks on the contribution of each node to activation patterns.

7.3.3 Robustness Enhancement Results

To address **RQ2** we assess our framework's effectiveness in identifying model repair actions for robustness enhancement. We use the actions evaluation approach (§7.3.3) to determine the most effective sequence of cumulative actions

Figure 7.4 shows the post-filtering and ordering cumulative actions (x-axis) and their impact on the model's accuracy (y-axis) after every iteration (vertical line). For the sake of simplified visualization, we focus on the most performing layer (i.e., layer 0 for Ember-DNN and layer 3 for CIFAR10-Resnet18). The charts indicate increasing curves of accuracy (red lines) on adversarial data as we incrementally perform graph-specific cumulative actions, along with a slight decrease in the accuracy on benign data. Inline with these results, the post-filtering cumulative trade-off curves (orange lines) follow a similar increasing pattern as the accuracy on adversarial data, reflecting that, by leveraging our repair action generation mechanism, the proposed characterization can lead to identify effective actions for robustness enhancement.

Furthermore, we records in table 7.1 the overall accuracy results of each model before and after performing robustness enhancement actions over all layers. The table highlights how the applied robustness-enhancement actions significantly improve the model's accuracy under various adversarial attacks while maintaining a minimal decrease in accuracy in benign settings. Across all attacks, the ”With Actions” results consistently outperform the ”No Actions” results, demonstrating substantial gains in adversarial robustness. At the same time, the decrease in benign accuracy is kept under 4%, indicating that these actions introduce only a small trade-off in everyday performance.

Model	CIFAR10-ResNet18						Ember	
Test Attack	FGSM	PGD	APGD-DLR	SPSA	Square	Benign	Emb-Att	Benign
No Actions	11.08%	3.14%	4.14%	24.05%	4.14%	**87.94%**	0.05%	**94.88%**
With Actions	**64.72%**	**65.92%**	**23.80%**	**42.29%**	**17.33%**	84.46%	**42.15%**	91.03%

Table 7.1: **Robustness Analysis Results (Accuracy)**. Accuracy of each model before vs. after performing robustness enhacement actions

7.4 Conclusion

We introduced a customizable framework that empowers DNN deployers to capture the computational information flow of a DNN's inference via inference provenance graphs (IAGs), (empirically and structurally) characterizes IAGs across settings, and finally leverages the IAG-based characterizations towards systematic model robustness enhancement. Our evaluation showed that significant differences between benign and adversarial settings allow for identifying key nodes for security repair. Our framework improves robustness with minimal accuracy loss on benign inputs.

CHAPTER 8

Discussion and Future works

8.1 Introduction

Our multi-faceted approach to characterizing and addressing machine learning (ML) security particularly in relation to adversarial examples—has led to a deeper understanding of the problem and significant progress in mitigating attacks. However, no one has yet succeeded in deploying an ML system that is completely robust against evasion attacks. As discussed in Chapter 6, ML models are inherently trained under the i.i.d. (independent and identically distributed) assumption, which contributes to their lack of generalization when encountering out-of-distribution (OOD) test data. This inherent vulnerability is a major factor in the success of evasion attacks.

Given these limitations, the current state of AI security prompts us to question whether AI security threats are, in fact, unsolvable. Should we accept these vulnerabilities and focus on mitigating them to the greatest extent possible?

An equally intriguing idea is whether adversarial examples could be seen not merely as threats, but as potential features. Could adversarial examples, for instance, serve as a solution to another ML-related problem? In Section 8.2, we explore a real-world scenario where this possibility becomes a reality.

8.2 Adversarial Examples: Are they a curse or a blessing?

Adversarial examples are by definition the result of malicious input crafting that aims to fool a target ML model. As explained throughout most of our previous works, when the ML model is deployed as a service (e.g., self-driving car, camera surveillance, etc), adversarial examples present a serious security threat. However, we argue that in some cases it might be to our benefit to confuse a malicious ML model. For instance, internet freedom technologists are constantly fighting to evade censored regimes (e.g., India, China, Iran, etc) which have led to a heated arms race that catalyzed the production of more sophisticated censoring techniques along with more improved censorship measurement tools (e.g., Geneva [30]). In our latest published work [16], we show that censors can use ML to detect the most advanced censorship measurement tools with high confidence. In this situation, it is to the benefit of Internet users within censored regimes to evade detection and ironically adversarial examples can play a major role to achieve that. Furthermore, in [16], we have proposed a tool called DeResistor that can extend any censor-probing based censorship evasion tool to protect it from ML detection. The core idea of this approach is taking advantage to ML vulnerability to adversarial examples.

8.2.1 DeResistor: Leveraging Adversarial Examples for Detection Evasion Against Internet Censorship

First, we introduce a ML detection system of one of the latest automated tool for censorship-evasion called Geneva [30]. It was developed to automate the creation of packet manipulation strategies for censorship evasion, addressing the manual evade-detect cycle seen in prior systems like INTANG [131], liberate [83], and brdgrd [133]. Using a genetic algorithm, Geneva derives evasion strategies from four basic packet manipulation techniques: drop, tamper headers, duplicate, and fragment. It has successfully re-derived many strategies from earlier manual efforts and even discovered new strategies for bypassing censorship in countries such as China, India, Iran, and Kazakhstan.

8.2.1.1 ML Detection Approach

To demonstrate how probing traffic of automated evasion tools can be easily detected, we introduce a two-step approach to detect Geneva clients at the censor side with high confidence.

Flow-Level ML-Based Detection By running Geneva against the censor, middlebox operators can collect Geneva traces and train a ML model that distinguishes Geneva traffic from normal traffic. Figure 8.1 shows feature analysis of our ML-based Geneva detection model. From the density plots, we notice that Geneva TCP packets have several corrupt data-offset fields and tend to have smaller size compared to normal traffic. Furthermore, Geneva may tamper with other TCP header fields like checksum or TTL, as part of its probing design to locate filtering middleboxes. We also notice that overlapping TCP segments are more likely to occur in Geneva traffic due to the tampering of packet payloads. Using these distinctive features, a fairly simple ML model (e.g., Decision Trees, Random Forests) is able to accurately distinguish Geneva flows from normal flows. Figure 8.2 shows that all four models (Decision Trees (DT), Random Forests (RF), Logistic Regression (LR) and Support Vector Machines (SVM)) are able to detect almost all Geneva flows in the test set, with negligible false positives ($AUC > 0.99$).

To enable a more reliable user IP-blocking decision, we adopt a multi-observation (multi-flow)

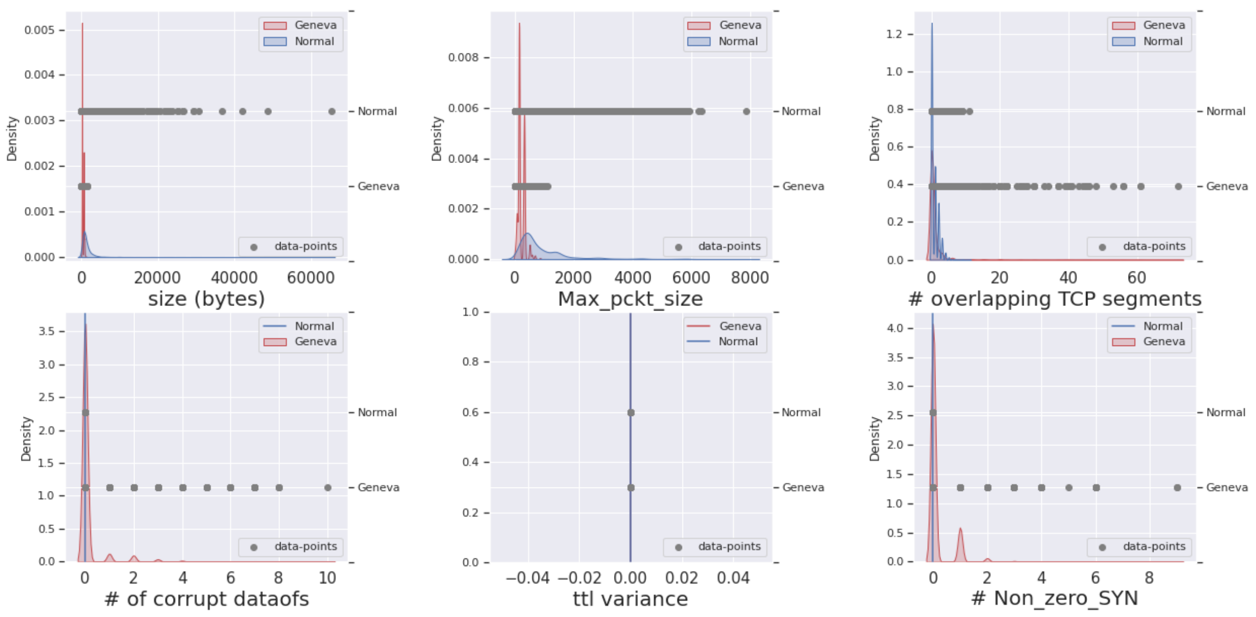


Figure 8.1: Geneva training feature analysis. Scatter plots show data-points of Geneva vs. normal traffic. Curves show data distribution density of each traffic type.

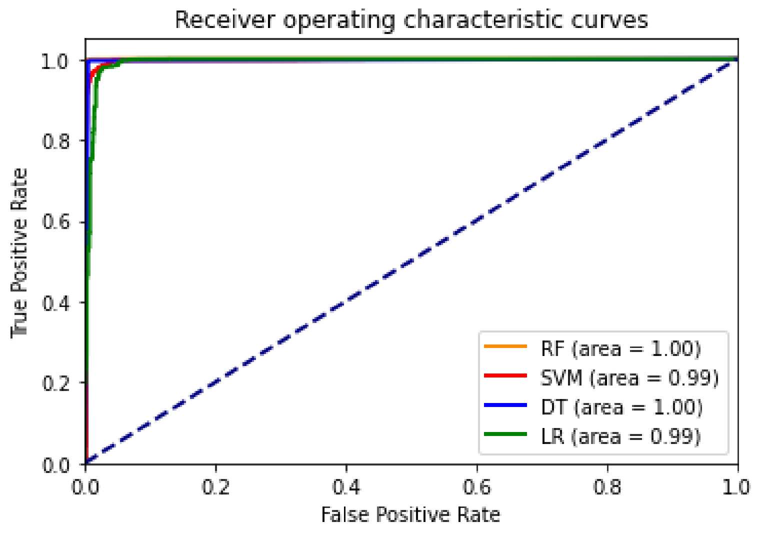


Figure 8.2: ML model performance for flow-level detection.

detection method that relies on a sequential hypothesis-testing approach based on the popular TRW algorithm [69] used in port-scan detection.

IP-Level TRW-Based Detection: we use the Threshold Random Walk (TRW) algorithm (Figure 8.3) to identify whether a given source IP is running Geneva, a traffic obfuscation tool. For each flow initiated by the source, a machine learning model f predicts the outcome as either Geneva ($Y_i = 0$) or benign ($Y_i = 1$). The goal is to quickly and accurately classify the source IP based on two hypotheses: $\mathcal{H}_0$, where the source is running Geneva, and $\mathcal{H}_1$, where it is benign. The detection is based on Bernoulli-distributed random variables Y_i under each hypothesis, with prob-

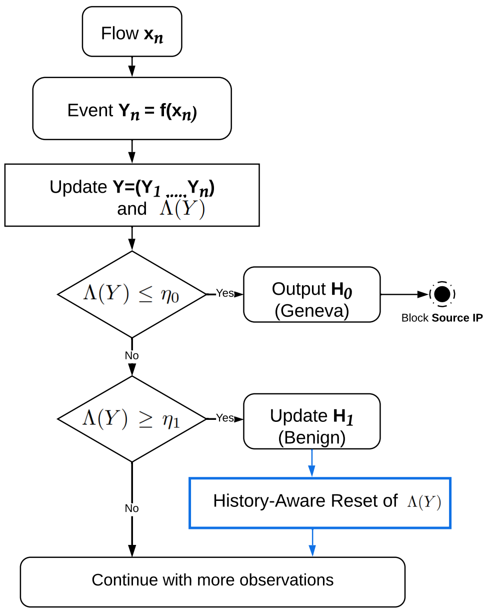

Flow $\mathbf{x}_n$
Event $\mathbf{Y}_n = \mathbf{f}(\mathbf{x}_n)$
Update $\mathbf{Y}=(\mathbf{Y}_1,\dots,\mathbf{Y}_n)$ and $\Lambda(Y)$
$\Lambda(Y) \leq \eta_0$
Yes
Output $\mathbf{H}_0$ (Geneva)
Block Source IP
No
$\Lambda(Y) \geq \eta_1$
Yes
Update $\mathbf{H}_1$ (Benign)
History-Aware Reset of $\Lambda(Y)$
No
Continue with more observations

Figure 8.3: Flow diagram of TRW for real-time Geneva detection. The blue box represents an extension to the original TRW algorithm.

abilities θ_0 and θ_1 calculated from the model's true positive (TP) and false positive (FP) rates. The algorithm aims to minimize false positives ($P_F \leq \alpha$) and maximize detection accuracy ($P_D \geq \beta$), typically with $\alpha = 0.01$ and $\beta = 0.99$. The likelihood ratio Λ is computed from observed flows, and decisions are made based on thresholds η_0 and η_1. If $\Lambda \leq \eta_0$, the source is blocked as Geneva; if $\Lambda \geq \eta_1$, it is considered benign, though history-aware resets are applied if there were previous Geneva detections. If the likelihood ratio is between these thresholds, further observations are made. The thresholds are set using TRW-specific formulas to ensure detection reliability.

Using the proposed ML-TRW-based detection approach we were able to detect Geneva probes (training) after it tests only 2 strategies of the first generation. This result is consistent across all simulated censors and China's Great Firewall. Considering that Geneva can derive previous manipulation strategies proposed in other censor probing-based tools (e.g. INTANG, liberate) [30], we believe that our detection approach is adaptable to be performed against them. In particular, the ML model used for flow-level detection can be trained on the network traces of any other probing-based tool.

8.2.1.2 DeResistor

Revisiting Figure 8.1, we observe that Geneva traffic exhibits distinct features compared to normal traffic, which contributes to the high flow-level detection accuracy by the studied machine learning models. The DeResistor approach is motivated by the fact that Geneva can evade detection by crafting adversarial flows through feature perturbation, making them resemble normal traffic. To

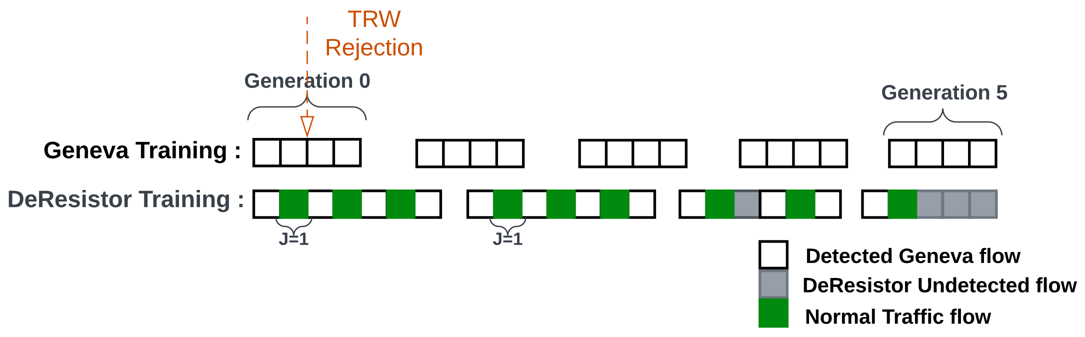


Figure 8.4: An illustration of Geneva genetic evolution traces when trained with DeResistor design vs. standalone training.

implement this strategy, we adopt two complementary methods. First, we modify Geneva's fitness function to guide its training towards detection-resistant strategies. Second, we introduce regular pauses in Geneva's training, allowing the user to engage in normal internet activity, thereby adding noise to the detection system and further complicating its ability to identify Geneva traffic. Next, we illustrate these two methods.

Two-Objective Fitness Function: Geneva initially employs a canary strategy that would be detected by the ML model. To enhance detection resilience, DeResistor introduces a two-objective fitness function, defined as:

$$fitness(s) = a.G(s) - b.P(s), \tag{8.1}$$

where s is the manipulation strategy. The term $G(s)$ measures the effectiveness of the evasion strategy based on censor feedback, while $P(s)$ calculates the probability of detection by the ML model. Since the objectives conflict, the second term is subtracted, aiming to maximize effectiveness and minimize detectability. The system optimizes the trade-off between these objectives using parameters a and b, where $a + b = 1$, reflecting the user's preferences.

Background Traffic Generation: DeResistor uses a two-objective fitness function to learn detection-resilient strategies over time. Early on, its strategies resemble Geneva's and are easily detected by the ML-based evasion detector. To counter this, DeResistor introduces benign background traffic between the client's IP and uncensored websites, pausing censor probing if a strategy is detected and resuming it once enough benign flows are observed (Figure 8.4). These delays detection, giving more time for undetectable strategies to evolve. Adjusting the benign flow jump size (J) helps confuse the ML detector but must be carefully managed to avoid triggering IP blocking.

8.2.1.3 Evaluation

Our evaluation of DeResistor is guided by these questions:

RQ1: Is DeResistor effective at making Geneva's probing traffic more resilient to detection?

RQ2: How effective is our approach in toning down the detectability of packet manipulation features?

Experimental Setup: To ensure a fair comparison, we ran DeResistor and Geneva under identical experimental conditions, closely following the setup outlined in the Geneva paper [30]. We focused on evaluating GFW's HTTP censorship from three locations in China (Qingdao, Beijing, and Shanghai). In each location, we ran both tools three times independently, collecting the fittest strategies. Each training session started with 500 individuals and was capped at 20 generations. Our results showed that this parameter tuning successfully trained both tools against the studied censors. We also tested their detection-resilience against a real-time detection system.

`In-Situ Validation`: Following the validation approach used in the Geneva experiments [30], we first performed a Dockerized evaluation of DeResistor against 11 mock censors proposed by Geneva's authors. Each strategy was tested in an isolated environment with four containers: a client, a mock censor, a forbidden server, and a legitimate server. The legitimate server simulated normal background traffic via Harpoon [117], which DeResistor uses when pausing training. Harpoon continuously sent TCP packets between the client and legitimate server using the same source IP involved in the training evaluation.

`Real-World Evaluation`: We evaluated circumvention strategies against the Great Firewall (GFW) of China, testing each strategy multiple times across different vantage points. The GFW censors HTTP requests by injecting RST packets and employs residual censorship, blocking connections for a limited time. To bypass DNS poisoning, which redirects users to fake IP addresses, we manually resolved correct IPs and successfully accessed certain websites using tools like Geneva. However, websites with more advanced censorship, such as null-routing, remained inaccessible. Similar tests were conducted in India and Kazakhstan on other censored websites, yielding comparable results.

`Detection Resilience`: We evaluated DeResistor's detection-resilience against GFW, India, Kazakhstan, and mock censors, using a real-time Geneva detection system (ML+TRW). The system monitored training, blocking the client if censor-probing was detected with 99% confidence ($\alpha = 0.99$, $\beta = 0.01$). To balance success and detection-resilience, DeResistor used a two-objective fitness function with equal weights ($a = 0.5$, $b = 0.5$) and started with a jump size of $J = 1$. Detection-resilience was achieved if DeResistor completed training without IP-blocking or reduced detection rates over time.

Detection-Resilience Results:

A strategy evaluation against the censor is tracked by a network flow between the client and the server. We recall that, our detection approach uses a ML model that performs a preliminary flow-level detection. Each flow detection is marked as another observation for the TRW algorithm to make a more confident IP-level detection. Figure 8.5 shows generations (x-axis) vs. flow-level

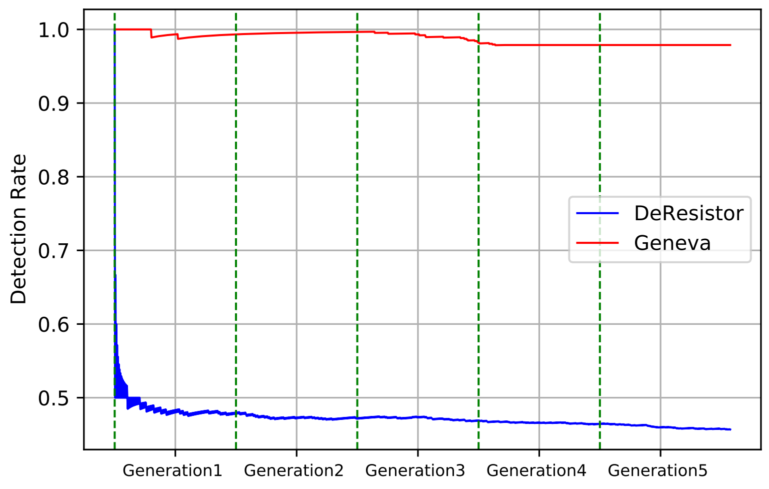


Figure 8.5: Flow-level detection rate evolution during Geneva and DeResistor training against China's GFW. We consider the 5 first generations.

Censors	**IP-Level Detection** (Geneva→DeResistor)	**Flow-Level Detection** (Geneva→DeResistor)	**Jump Size** J
China's GFW	Detected after 2 flows → **Undetected**	96.27% → **45.06%**	1
India	Detected after 2 flows → **Undetected**	99.50% → **34.93%**	1
Kazakhstan	Detected after 2 flows → **Undetected**	99.50% → **49.22%**	1
Censor 1-4,7,9	Detected after 2 flows → **Undetected**	99.4% → **32.46%**	1
Censor 5,10.	Detected after 2 flows → **Undetected**	99.4% → **31.21%**	1
Censor 6	Detected after 2 flows → **Undetected**	99.4% → **34.05**	1
Censor 8	Detected after 2 flows → **Undetected**	99.4%→ **30.93%**	1
Censor 11	Detected after 2 flows → **Undetected**	99.4%→ **29.91%**	1

Table 8.1: Geneva vs. DeResistor detection results using history-aware TRW. Details about mock censors can be found in [30].

detection rate trend (y-axis) during Geneva and DeResistor training. Furthermore, in Table 8.1, we report the IP-level detection results recorded by the TRW on Geneva and DeResistor when trained against different censors.

`Flow-Level Detection Resilience`: Figure 8.5 confirms that DeResistor traces are way less detectable compared to Geneva. In particular, as DeResistor training advances, the detection rate continues to drop until it reaches 45.06% after 5 generations while it stays very high

(96.27%) during Geneva training. The immediate drop observed at the beginning of generation 1 (i.e., $1 \rightarrow 0.5$) is caused by the guided pausing of the genetic algorithm training when a flow is detected that permits the client to engage with a number J of normal benign flows. The same drop is observed when we run Geneva with guided pauses for normal traffic injection (without multi-objective optimization). However, due to the proposed two-objective fitness function, the observed decrease in the detection rate from generation 0 to generation 5 shows that DeResistor is learning to generate less detectable strategies as it advances to higher generations, compared to ”Geneva+normal traffic”.

To further explain our findings, we investigate the features of DeResistor traces (adversarial examples) compared to Geneva traces (normal examples) and normal traffic in Figure 8.6. Overall, we observe that the feature values density of DeResistor (green curve) are closer to Normal traffic (blue curve) compared to Geneva (red curve). Furthermore, Looking into the data-points of each traffic type (gray scatter plots), it seems that DeResistor traces exhibit less overlapping TCP segments, less corrupt data-offset fields, and less corrupt SYN packets compared to Geneva. We conclude that DeResistor is able to tone down detectable features exhibited by Geneva, which leads to lower flow-level detection rate (answers **RQ2**). However, we acknowledge that DeResistor traces are still different from normal traces, which is natural considering that its main objective is still to manipulate packets and evade censorship. It is noteworthy that DeResistor can be even less detectable if the user chooses to give advantage to the detection-resilience objective at the expense of the strategy success objective (e.g., a=0.3, b=0.7). In-line with the GFW results, Table 8.1 (3^{rd} column) shows that DeResistor was able to reduce the flow-level detection rate ($99.4\% \rightarrow \approx 32\%$) when trained against the 11 mock censors as well.

`IP-Level Detection Resilience:` As reported before, using the proposed detection approach we were able to detect with high confidence a source IP address running Geneva after the TRW receives only 2 observations (i.e., 2 Geneva flows/probes). According to results reported in Table 8.1, we observe that, against all the studied censors, DeResistor was able to complete its training without being rejected by TRW (2^{nd} column), which answers **RQ1**. A jump size $J = 1$ was sufficient to reach these results (3^{rd} column). Particularly, as we illustrated in Figure 8.4, after each flow-level detection of a strategy test, the injected normal flow restores the likelihood ratio $\Lambda(Y)$ of the TRW to its initial value. This pattern encapsulates the $\Lambda(Y)$ between η_0 and η_1, which makes it longer for the TRW to converge to a decision. Setting the jump size to $J = 1$ is not only sufficient to avoid detection, but also recommended to avoid triggering TRW resets. We recall that, our implementation of TRW algorithm is powered by a history-aware reset of the likelihood ratio in case the TRW is converging to a decision that the source IP is benign. Thus, it is in favor of the client to avoid pushing the TRW to make a reset that considers all previous detection of packet manipulations.

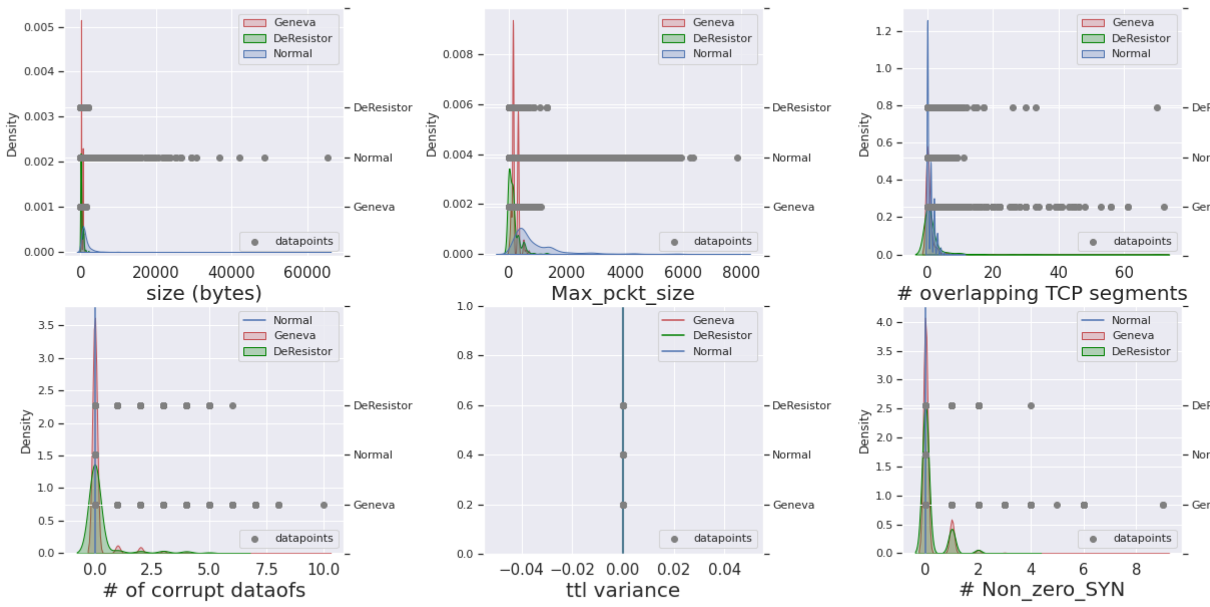


Figure 8.6: DeResistor training feature analysis. The scatter plot displays data-points of Geneva, DeResistor, and normal traffic. The curves show the data distribution density of each traffic type.

Conclusion: AI security is a paramount research problem to explore, particularly adversarial examples. Moreover, through DeResistor we demonstrate that it can be useful in the broader application of ML in multiple tasks, specifically in cases where evading a ML system can provide a greater good (e.g., censor-side detector of censorship evasion tools). DeResistor factors in the likelihood of detectability of censorship strategies when it generates evasion strategies and guides Geneva's strategy generator to rely on less-detectable features. By doing so, it not only enables the generation of a successful censorship evasion strategy, but also shields Geneva from being detected by the censor. We evaluated DeResistor first using 11 mock censors and then against real-world censors, China's GFW, India and Kazakhstan.

8.3 Future Work: Security and Privacy of Foundation Models

Foundation models, particularly large language models (LLMs) like GPT, PaLM, and LLaMA, have revolutionized artificial intelligence (AI) by exhibiting remarkable capabilities in language comprehension, generation, and reasoning. However, the increasing power and accessibility of these models raise significant concerns about security and privacy. Their deployment spans across sensitive domains such as healthcare, finance, and governance, which exacerbates the need to address vulnerabilities, misuse, and privacy breaches.

As AI research and development increasingly pivot toward foundation models, our research is naturally following this trend, addressing the unique security and privacy challenges associated with large language models (LLMs). These challenges include vulnerabilities in training data, the risks of adversarial manipulation, and potential data leakage. Furthermore, we explore emerging mitigation strategies and regulatory frameworks aimed at resolving these issues.

8.3.1 Security Threats in LLMs

Foundation models, particularly large language models (LLMs), introduce unique security challenges due to their ability to autonomously and adaptively generate text. Key threats include adversarial attacks, where crafted prompts manipulate outputs to produce harmful or biased content, and poisoning attacks, in which malicious data injected during online learning distorts model behavior. Evasion attacks further exploit vulnerabilities, tricking models into leaking sensitive information like personally identifiable data (PII). LLMs also face risks of model theft and extraction attacks, where adversaries replicate proprietary models, undermining their economic value and exposing proprietary algorithms. Additionally, LLMs can be misused to generate spam, phishing messages, or large-scale misinformation, posing significant social, political, and economic risks through automated disinformation and hyper-personalized manipulations.

8.3.2 Privacy Challenges in LLMs

LLMs, often trained on large datasets scraped from public sources, can unintentionally ingest private or sensitive information, leading to several privacy risks. One major issue is *data leakage and memorization*, where models retain sensitive details, such as passwords or private conversations, which attackers can extract through repeated queries or membership inference attacks. Additionally, the *lack of transparency and data governance* complicates efforts to determine if training data contains personally identifiable information (PII) or copyrighted material, potentially breaching regulations like GDPR or CCPA. Even with anonymization efforts, *re-identification risks* persist, as LLMs can infer identities by correlating seemingly anonymous data with other available datasets.

8.3.3 Ongoing Efforts and Future Research Needs

While several mitigation strategies have been introduced to address the security and privacy challenges of LLMs, further research is needed to enhance their effectiveness. Techniques like *differential privacy* are being employed to obscure sensitive data points, and *federated learning* offers a decentralized approach to limit data exposure. *Prompt filtering and output moderation* help block adversarial inputs, while *encryption techniques* such as homomorphic encryption and secure multi-party computation protect data during inference. Additionally, *regulatory frameworks* like the EU's AI Act and NIST's AI Risk Management Framework are emerging to promote ethical and secure AI practices.

However, these efforts are only the beginning. Future research must focus on developing *AI-powered auditing tools* for real-time threat detection, improving *explainable AI (XAI)* for greater transparency, and fostering *collaborative regulatory ecosystems* that involve academia, industry, and government. Continuous adversarial testing and *privacy-preserving innovations* will be essential to maintaining trust and ensuring that LLMs remain secure and privacy-conscious as they evolve. Balancing innovation with robust *security and privacy* measures will be critical for sustainable AI development in the future.

CHAPTER 9

Related Works

In this chapter, we position our contributions with respect to the related works. Since, we have already covered prior Evasion Attacks in 3.2, Here we focus on studying the previously proposed Defenses.

9.1 Defenses against ML Evasion Attacks

A handful of defense techniques were proposed as countermeasures against evasion attacks. Among the notable defense methods are *defensive distillation* [94], *adversarial training* [79], *adversarial example detection*[137], *monotonic models* [66], and *certified defenses*.

Defensive distillation. Papernot et al. [94] propose a method that generates a new model whose gradients are much smaller than the original undefended model. If gradients are very small, attack techniques like FGSM [55] or BIM [79] are no longer useful, as the attacker would need great distortions of the input image to achieve a sufficient change in the loss function. The defense enhances DNN-based image classifiers against gradient descent-based attacks. This defense mechanism has been broken later on by Carlini & Wagner (CW) attack [35] as the attack does not rely on the gradient function.

Adversarial Training. One of the relatively effective defense techniques is training the model on adversarially-perturbed examples using their true labels. This method is originally proposed by Kurakin et al. [79] and has been proven to reduce the evasion rate of adversarial examples. However, merging the training data with adversarial samples would make the model less accurate on benign input.

Monotonic Models. The output of a monotonic ML model can increase/decrease only if there is an increase/decrease in its monotonic features since, by definition, monotonic features are the only features that are highly correlated with the model's output. This idea has been harnessed to build robust malware detectors [66]. Such classifiers whose monotonic features are only malware features are robust against any benign feature injection to a malware file. Monotonic modeling can be an effective defense only in binary classification, typically in malware detection.

Detection of Adversarial Examples. Numerous prior works in this area attempted to detect adversarial examples by performing transformations on input examples (e.g., via spatial smoothing, rotation) instead of changing the ML model. These simple and inexpensive transformations on images have been proven to be effective to detect adversarial inputs generated by famous attacks (e.g., PGD, CW). However, it can confuse the model to incorrectly predict the label of clean samples [137, 125]

As shown in Table 9.1, several other defense techniques were proposed to counter specific attacks such as *Random Feature Nullification (RFN)* on Windows PE [130] that enhances DNN-based malware detectors against the FGSM attack [55] by nullifying (or dropping) features randomly in both training and testing phases. This offers a probabilistic assurance in preventing a white-box attacker from deriving adversarial files by using gradients of the loss function with respect to input. *One-and-half-class classifiers* as well enhance the PDF malware detector against the gradient-based attack in the feature space, given that its decision boundary is often tighter than that of two-class classifiers.

Certified Defenses. Another promising research direction that has been recently explored [100, 81, 134] suggests going beyond best-effort defense measures by exploiting the norm bound constraint (i.e., $||\delta|| < \epsilon$) that limits the possible perturbations of most of previously proposed attacks. The goal is to offer a *certificate* of robustness that guarantees a maximum of evasion rate which by design cannot be surpassed by the attack. Prior work managed to reach a certificate of robustness that bounds the evasion rate to a maximum of 60%. Although it is still not an acceptable guarantee of ML robustness, such a promising outcome encourages future improvements in this direction.

Moving Target Defenses. Network and software security has leveraged numerous flavors of MTD including randomization of service ports and address space layout randomization. Recent work has explored MTD for defending adversarial examples. Song et al. [118] proposed a fMTD where they create fork-models via independent perturbations of the base model and retrain them. Fork-models are updated periodically whenever the system is in an idle state. The input is sent to all models and the prediction label is decided by majority vote.

Sengupta et al. [110] proposed MTDeep, which uses different DNN architectures (e.g., CNN, HRNN, MLP) in a manner that reduces transferability between model architectures using a measure called *differential immunity*. Through Bayesian Stackelberg game, MTDeep chooses a model to classify an input. Despite diverse model architectures, MTDeep suffers from a small model pool size.

Qian et al. [98] propose EI-MTD, a defense that leverages the Bayesian Stackelberg game for dynamic scheduling of student models to serve prediction queries on resource-constrained edge

Table 9.1: Systematization of defenses against test-time evasion.

Domains	Works	Adv. Knowledge	Defense Strategy	Target Attack
Image	**Kurakin *et al.* [79]**	white-box	adversarial training	all attacks
	Madry *et al.* [89]	white-box	robust optimization	all attacks
	Papernot *et al.* [94]	white-box	defensive distillation	gradient-based
	[137, 125]	white-box+ black-box	Adv example detection	all attacks
	[100, 81, 134]	white-box	certified defense	Lp-norm attacks
Windows PE	**Incer *et al.* [66]**	gray-box	Monotonic Models	adding benign features
	Wang *et al.* [130]	white-box	random feature nullification	FGSM
Android PDF	**Smutz and Stavrou [116]**	white-box	ensemble learning	gradient-based
PDF	**Biggio *et al.* [26]**	black-box+white-box	One-and-half-class classifier	gradient-based

devices. The student models are generated via differential distillation from an accurate teacher model that resides on the cloud.

Compared to EI-MTD [98], Morphence-2.0 avoids inheritance of adversarial training limitations by adversarially training a subset of student models instead of the base model. Unlike EI-MTD that results in lower accuracy on clean data after adversarial training, Morphence-2.0 's accuracy on clean data after adversarial training is much better since the accuracy penalty is not inherited by student models. Instead of adding regularization term during training, Morphence-2.0 uses distinct transformed training data to retrain student models and preserve base model accuracy. Instead of the Bayesian Stackelberg game, Morphence-2.0 uses the most confident model for prediction.

With respect to MTDeep [110], Morphence-2.0 expands the pool size regardless of the heterogeneity of individual models and uses average transferability rate to estimate attack transferability. On scheduling strategy, instead of Bayesian Stackelberg game Morphence-2.0 uses the most confidence model.

Unlike fMTD [118], Morphence-2.0 goes beyond retraining perturbed fork models and adversarially trains a subset of the model pool to harden the whole pool against adversarial example attacks. In addition, instead of majority vote, Morphence-2.0 picks the most confident model for prediction. For pool renewal, instead of waiting when the system is idle, Morphence-2.0 takes a rather safer and transparent approach and renews an expired pool seamlessly on-the-fly.

CHAPTER 10

Conclusion

The sudden emergence of the field of Adversarial Machine Learning has proven that ML is not completely reliable for real-world deployment. Despite the exponential growth of the number of research papers in this area, our study in this report shows that the adversarial ML field has a long way to go given the various open problems discussed earlier. In this dissertation, we presented our contributions to this area of research focusing on four dimensions. We drew meaningful insights and diagnosed the ML evasion attacks through explanation-guided analysis (Chapter 3). Additionally, we advanced the security-related evaluation process of ML models by proposing an explanation-guided booster of ML evasion attacks that tests a ML model under a more serious evasion threat (Chapter 4). After that, we advanced the defense landscape by suggesting a moving target deployment of ML as a service that aims to dodge adversarial threats (i.e., Morphence in Chapter 5). Furthermore, we addressed the adversarial examples problem from the root cause by studying its cause-effect link with the OOD-Generalization problem (Chapter 6). In particular, we build a framework that generalizes the ML model on natural OOD inputs and adversarial inputs. Lastly, we focus on examining Neural Networks at the granularity of the activation patterns given different inputs (Chapter 7) to enable model robustness enhancement and we explore a different perspective where ML models' vulnerability to adversarial examples might be 'monetized' to solve real-world problems such as Internet censorship (Chapter 8).

Bibliography

[1] Cnet freeware site. https://download.cnet.com/s/software/windows/?licenseType=Free, 2019.

[2] Virus share. https://virusshare.com, 2019.

[3] Virus total. https://www.virustotal.com/gui/home/upload, 2019.

[4] CNN-CIFAR10 model. https://github.com/jamespengcheng/PyTorch-CNN-on-CIFAR10, 2020.

[5] Cuckoo sandbox. https://cuckoosandbox.org, 2020.

[6] Lief project. https://github.com/lief-project/LIEF, 2020.

[7] Zeeshan Ahmad, Adnan Shahid Khan, Cheah Wai Shiang, Johari Abdullah, and Farhan Ahmad. Network intrusion detection system: A systematic study of machine learning and deep learning approaches. *Trans. Emerg. Telecommun. Technol.*, 32(1), 2021.

[8] Kartik Ahuja, Karthikeyan Shanmugam, Kush R. Varshney, and Amit Dhurandhar. Invariant risk minimization games. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 145–155. PMLR, 2020.

[9] Abdullah Al-Dujaili, Alex Huang, Erik Hemberg, and Una-May O'Reilly. Adversarial deep learning for robust detection of binary encoded malware. In *2018 IEEE Security and Privacy Workshops, SP Workshops 2018, San Francisco, CA, USA, May 24, 2018*, pages 76–82, 2018.

[10] Abdullah Ali and Birhanu Eshete. Best-effort adversarial approximation of black-box malware classifiers. In *Security and Privacy in Communication Networks - 16th EAI International Conference, SecureComm 2020, Washington, DC, USA, October 21-23, 2020, Proceedings, Part I*, volume 335 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 318–338. Springer, 2020.

[11] Abderrahmen Amich and Birhanu Eshete. Explanation-Guided Diagnosis of Machine Learning Evasion Attacks. In *Security and Privacy in Communication Networks - 17th EAI International Conference, SecureComm 2021*, 2021.

[12] Abderrahmen Amich and Birhanu Eshete. Explanation-guided diagnosis of machine learning evasion attacks. In Joaquin Garcia-Alfaro, Shujun Li, Radha Poovendran, Hervé Debar, and Moti Yung, editors, *Security and Privacy in Communication Networks*, pages 207–228, Cham, 2021. Springer International Publishing.

[13] Abderrahmen Amich and Birhanu Eshete. Morphence: Moving target defense against adversarial examples. In *ACSAC '21: Annual Computer Security Applications Conference*, pages 61–75, 2021.

[14] Abderrahmen Amich and Birhanu Eshete. Eg-booster: Explanation-guided booster of ml evasion attacks. In *Proceedings of the Twelfth ACM Conference on Data and Application Security and Privacy*, CODASPY '22, page 16–28, New York, NY, USA, 2022. Association for Computing Machinery.

[15] Abderrahmen Amich and Birhanu Eshete. Rethinking machine learning robustness via its link with the out-of-distribution problem. *CoRR*, abs/2202.08944, 2022.

[16] Abderrahmen Amich, Birhanu Eshete, Vinod Yegneswaran, and Nguyen Phong Hoang. DeResistor: Toward Detection-Resistant probing for evasion of internet censorship. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 2617–2633, Anaheim, CA, August 2023. USENIX Association.

[17] H. S. Anderson and P. Roth. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. *ArXiv e-prints*, April 2018.

[18] Hyrum S. Anderson, Anant Kharkar, Bobby Filar, David Evans, and Phil Roth. Learning to evade static PE machine learning malware models via reinforcement learning. *CoRR*, abs/1801.08917, 2018.

[19] Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. Globally normalized transition-based neural networks. In *ACL 2016*. The Association for Computer Linguistics, 2016.

[20] Maksym Andriushchenko, Francesco Croce, Nicolas Flammarion, and Matthias Hein. Square attack: a query-efficient black-box adversarial attack via random search, 2020.

[21] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society, 2014.

[22] Anish Athalye, Nicholas Carlini, and David A. Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *ICML 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 274–283, 2018.

[23] Ulrich Aïvodji, Alexandre Bolot, and Sébastien Gambs. Model extraction from counterfactual explanations, 2020.

[24] Abhijit Bendale and Terrance Boult. Towards open set deep networks, 2015.

[25] Arjun Nitin Bhagoji, Daniel Cullina, Chawin Sitawarin, and Prateek Mittal. Enhancing robustness of machine learning systems via data transformations. In *52nd Annual Conference on Information Sciences and Systems, CISS 2018*, pages 1–5, 2018.

[26] B. Biggio, I. Corona, Zhi-Min He, P. Chan, G. Giacinto, D. Yeung, and F. Roli. One-and-a-half-class multiple classifier systems for secure learning against evasion attacks at test time. In *MCS*, 2015.

[27] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Srndic, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III*, pages 387–402, 2013.

[28] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, 2018.

[29] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, dec 2018.

[30] Kevin Bock, George Hughey, Xiao Qiang, and Dave Levin. Geneva: Evolving Censorship Evasion. 2019.

[31] Jacob Buckman, Aurko Roy, Colin Raffel, and Ian J. Goodfellow. Thermometer encoding: One hot way to resist adversarial examples. In *6th International Conference on Learning Representations, ICLR 2018*, 2018.

[32] Nicholas Carlini. A Complete List of All (arXiv) Adversarial Example Papers. `https://nicholas.carlini.com/writing/2019/all-adversarial-example-papers.html`, 2020.

[33] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, Aleksander Madry, and Alexey Kurakin. On evaluating adversarial robustness, 2019.

[34] Nicholas Carlini and David A. Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2017*, pages 3–14, 2017.

[35] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 39–57, 2017.

[36] Jianbo Chen, Michael I. Jordan, and Martin J. Wainwright. Hopskipjumpattack: A query-efficient decision-based attack. In *2020 IEEE Symposium on Security and Privacy, SP 2020*, pages 1277–1294, 2020.

[37] Jeremy M. Cohen, Elan Rosenfeld, and J. Zico Kolter. Certified adversarial robustness via randomized smoothing. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019*, volume 97 of *Proceedings of Machine Learning Research*, pages 1310–1320, 2019.

[38] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding, 2016.

[39] Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks, 2020.

[40] Jacson Rodrigues Correia da Silva, Rodrigo Ferreira Berriel, Claudine Badue, Alberto Ferreira de Souza, and Thiago Oliveira-Santos. Copycat CNN: stealing knowledge by persuading confession with random non-labeled data. In *2018 International Joint Conference on Neural Networks, IJCNN 2018*, pages 1–8, 2018.

[41] G. E. Dahl, Dong Yu, Li Deng, and A. Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1):30–42, January 2012.

[42] Nilaksh Das, Madhuri Shanbhogue, Shang-Tse Chen, Fred Hohman, Li Chen, Michael E. Kounavis, and Duen Horng Chau. Keeping the bad guys out: Protecting and vaccinating deep learning with JPEG compression. *CoRR*, abs/1705.02900, 2017.

[43] Luca Demetrio, Battista Biggio, Giovanni Lagorio, Fabio Roli, and Alessandro Armando. Explaining vulnerabilities of deep learning to adversarial malware binaries. In *Proceedings of the Third Italian Conference on Cyber Security, Pisa, Italy, February 13-15, 2019.*, 2019.

[44] Luca Demetrio, Battista Biggio, Giovanni Lagorio, Fabio Roli, and Alessandro Armando. Efficient Black-box Optimization of Adversarial Windows Malware with Constrained Manipulations. *CoRR*, abs/2003.13526, 2020.

[45] Ambra Demontis, Marco Melis, Battista Biggio, Davide Maiorca, Daniel Arp, Konrad Rieck, Igino Corona, Giorgio Giacinto, and Fabio Roli. Yes, machine learning can be more secure! A case study on android malware detection. *IEEE Trans. Dependable Secur. Comput.*, 16(4):711–724, 2019.

[46] Ambra Demontis, Marco Melis, Maura Pintor, Matthew Jagielski, Battista Biggio, Alina Oprea, Cristina Nita-Rotaru, and Fabio Roli. Why Do Adversarial Attacks Transfer? Explaining Transferability of Evasion and Poisoning Attacks. In Nadia Heninger and Patrick Traynor, editors, *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, pages 321–338. USENIX Association, 2019.

[47] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting adversarial attacks with momentum. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 9185–9193, 2018.

[48] Logan Engstrom, Andrew Ilyas, Hadi Salman, Shibani Santurkar, and Dimitris Tsipras. Robustness (python library), 2019.

[49] Ivan Evtimov, Kevin Eykholt, Earlence Fernandes, Tadayoshi Kohno, Bo Li, Atul Prakash, Amir Rahmati, and Dawn Song. Robust physical-world attacks on machine learning models. *CoRR*, abs/1707.08945, 2017.

[50] M. Fan, W. Wei, X. Xie, Y. Liu, X. Guan, and T. Liu. Can we trust your explanations? sanity checks for interpreters in android malware analysis. *IEEE Transactions on Information Forensics and Security*, 16:838–853, 2021.

[51] Feng Gao, Wei Wang, Miaomiao Tan, Lina Zhu, Yuchen Zhang, Evelyn Fessler, Louis Vermeulen, and Xin Wang. DeepCC: a novel deep learning-based framework for cancer molecular subtype classification. *Oncogenesis*, 8(9), August 2019.

[52] Amirata Ghorbani, Abubakar Abid, and James Zou. Interpretation of neural networks is fragile. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33, 10 2017.

[53] Ian Goodfellow. A research agenda: Dynamic models to defend against correlated attacks, 2019.

[54] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[55] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[56] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick D. McDaniel. Adversarial examples for malware detection. In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II*, pages 62–79, 2017.

[57] Shixiang Gu and Luca Rigazio. Towards deep neural network architectures robust to adversarial examples. In *3rd International Conference on Learning Representations, ICLR 2015*, 2015.

[58] Chuan Guo, Mayank Rana, Moustapha Cissé, and Laurens van der Maaten. Countering adversarial images using input transformations. In *6th International Conference on Learning Representations, ICLR 2018*, 2018.

[59] Wenbo Guo, Dongliang Mu, Jun Xu, Purui Su, Gang Wang, and Xinyu Xing. LEMNA: explaining deep learning based security applications. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 364–379. ACM, 2018.

[60] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[61] Warren He, James Wei, Xinyun Chen, Nicholas Carlini, and Dawn Song. Adversarial example defense: Ensembles of weak defenses are not strong. In *11th USENIX Workshop on Offensive Technologies, WOOT*, 2017.

[62] Juyeon Heo, Sunghwan Joo, and Taesup Moon. Fooling neural network interpretations via adversarial model manipulation. 02 2019.

[63] Weiwei Hu and Ying Tan. Generating adversarial malware examples for black-box attacks based on GAN. *CoRR*, abs/1702.05983, 2017.

[64] Weiwei Hu and Ying Tan. Black-box attacks against RNN based malware detection algorithms. In *The Workshops of the The Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*, pages 245–251, 2018.

[65] Ruitong Huang, Bing Xu, Dale Schuurmans, and Csaba Szepesvári. Learning with a strong adversary. *CoRR*, abs/1511.03034, 2015.

[66] Inigo Incer, Michael Theodorides, Sadia Afroz, and David Wagner. Adversarially robust malware detection using monotonic classification. pages 54–63, 03 2018.

[67] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2018.

[68] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. High accuracy and high fidelity extraction of neural networks, 2020.

[69] Jaeyeon Jung, V. Paxson, A.W. Berger, and H. Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*, pages 211–225, 2004.

[70] Levent Karacan, Zeynep Akata, Aykut Erdem, and Erkut Erdem. Learning to generate images of outdoor scenes from attributes and semantic layouts, 2016.

[71] Narine Kokhlikyan, Vivek Miglani, Miguel Martin, Edward Wang, Bilal Alsallakh, Jonathan Reynolds, Alexander Melnikov, Natalia Kliushkina, Carlos Araya, Siqi Yan, and Orion Reblitz-Richardson. Captum: A unified and generic model interpretability library for pytorch, 2020.

[72] Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, and Fabio Roli. Adversarial malware binaries: Evading deep learning for malware detection in executables. In *26th European Signal Processing Conference, EUSIPCO 2018, Roma, Italy, September 3-7, 2018*, pages 533–537, 2018.

[73] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

[74] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). 2009.

[75] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, 2017.

[76] David Krueger, Ethan Caballero, Jörn-Henrik Jacobsen, Amy Zhang, Jonathan Binas, Dinghuai Zhang, Rémi Le Priol, and Aaron C. Courville. Out-of-distribution generalization via risk extrapolation (rex). In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 5815–5826. PMLR, 2021.

[77] kuangliu. Cifar10 best models. https://github.com/kuangliu/pytorch-cifar, 2020.

[78] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale, 2017.

[79] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *CoRR*, abs/1611.01236, 2016.

[80] Yan LeCun, Corinna Cortes, and Christopher J.C. Burges. The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/, 2020.

[81] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy, 2019.

[82] Bai Li, Changyou Chen, Wenlin Wang, and Lawrence Carin. Certified adversarial robustness with additive noise. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019*, pages 9459–9469, 2019.

[83] Fangfan Li, Abbas Razaghpanah, Arash Molavi Kakhki, Arian Akhavan Niaki, David Choffnes, Phillipa Gill, and Alan Mislove. Lib•erate, (n): A library for exposing (traffic-classification) rules and avoiding them efficiently. In *Proceedings of the 2017 Internet Measurement Conference*, IMC '17, page 128–141, New York, NY, USA, 2017. Association for Computing Machinery.

[84] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. Unsupervised image-to-image translation networks, 2018.

[85] Ming-Yu Liu and Oncel Tuzel. Coupled generative adversarial networks, 2016.

[86] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation, 2015.

[87] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 4765–4774, 2017.

[88] Yan Luo, Xavier Boix, Gemma Roig, Tomaso A. Poggio, and Qi Zhao. Foveation-based mechanisms alleviate adversarial examples. *CoRR*, abs/1511.06292, 2015.

[89] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *CoRR*, abs/1706.06083, 2017.

[90] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *CoRR*, abs/1706.06083, 2017.

[91] Marco Melis, Ambra Demontis, Maura Pintor, Angelo Sotgiu, and Battista Biggio. secml: A python library for secure and explainable machine learning. *arXiv preprint arXiv:1912.10013*, 2019.

[92] Smitha Milli, Ludwig Schmidt, Anca D. Dragan, and Moritz Hardt. Model reconstruction from model explanations. In *Proceedings of the Conference on Fairness, Accountability, and Transparency, FAT* 2019, Atlanta, GA, USA, January 29-31, 2019*, pages 1–9. ACM, 2019.

[93] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Knockoff nets: Stealing functionality of black-box models, 2018.

[94] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. pages 582–597, 05 2016.

[95] Nicolas Papernot, Patrick D. McDaniel, and Ian J. Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *CoRR*, abs/1605.07277, 2016.

[96] Nicolas Papernot, Patrick D. McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical Black-Box Attacks against Deep Learning Systems using Adversarial Examples. *CoRR*, abs/1602.02697, 2016.

[97] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. Intriguing properties of adversarial ML attacks in the problem space. *CoRR*, abs/1911.02142, 2019.

[98] Yaguan Qian, Qiqi Shao, Jiamin Wang, Xiang Lin, Yankai Guo, Zhaoquan Gu, Bin Wang, and Chunming Wu. EI-MTD: moving target defense for edge intelligence against adversarial attacks. *CoRR*, abs/2009.10537, 2020.

[99] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K. Nicholas. Malware detection by eating a whole EXE. In *The Workshops of the The Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*, pages 268–276, 2018.

[100] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples, 2020.

[101] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. ”why should I trust you?”: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144, 2016.

[102] Maria Rigaki and Sebastian Garcia. Bringing a GAN to a knife-fight: Adapting malware communication to avoid detection. In *2018 IEEE Security and Privacy Workshops, SP Workshops 2018, San Francisco, CA, USA, May 24, 2018*, pages 70–75, 2018.

[103] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[104] Ishai Rosenberg, Asaf Shabtai, Lior Rokach, and Yuval Elovici. Generic black-box end-to-end attack against state of the art API call based malware classifiers. In *Research in Attacks, Intrusions, and Defenses - 21st International Symposium, RAID 2018, Heraklion, Crete, Greece, September 10-12, 2018, Proceedings*, pages 490–510, 2018.

[105] Ahmad El Sallab, Mohammed Abdou, Etienne Perot, and Senthil Kumar Yogamani. Deep reinforcement learning framework for autonomous driving. *CoRR*, abs/1704.02532, 2017.

[106] Hadi Salman, Andrew Ilyas, Logan Engstrom, Ashish Kapoor, and Aleksander Madry. Do adversarially robust imagenet models transfer better?, 2020.

[107] Krzystof Podgorski Samuel Kotz, Tomasz Kozubowski. The laplace distribution and generalizations: A revisit with applications to communications, economics, engineering, and finance, 2012.

[108] Patsorn Sangkloy, Jingwan Lu, Chen Fang, Fisher Yu, and James Hays. Scribbler: Controlling deep image synthesis with sketch and color, 2016.

[109] Vikash Sehwag, Mung Chiang, and Prateek Mittal. Ssd: A unified framework for self-supervised outlier detection. In *International Conference on Learning Representations*, 2021.

[110] Sailik Sengupta, Tathagata Chakraborti, and Subbarao Kambhampati. Mtdeep: Boosting the security of deep neural nets against adversarial attacks with moving target defense. In *Decision and Game Theory for Security - 10th International Conference, GameSec 2019*, volume 11836 of *Lecture Notes in Computer Science*, pages 479–491, 2019.

[111] Giorgio Severi, Jim Meyer, Scott Coull, and Alina Oprea. Explanation-guided backdoor poisoning attacks against malware classifiers. In *30th USENIX Security Symposium, USENIX Security 2021*. USENIX Association, 2021.

[112] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1528–1540. ACM, 2016.

[113] Reza Shokri, Martin Strobel, and Yair Zick. On the privacy risks of model explanations, 2021.

[114] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 3145–3153, 2017.

[115] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR Workshop Track Proceedings*, 2014.

[116] Charles Smutz and Angelos Stavrou. When a tree falls: Using diversity in ensemble classifiers to identify evasion in malware detectors. 01 2016.

[117] Joel Sommers, Hyungsuk Kim, and Paul Barford. Harpoon: A flow-level traffic generator for router and network tests. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '04/Performance '04, page 392, New York, NY, USA, 2004. Association for Computing Machinery.

[118] Qun Song, Zhenyu Yan, and Rui Tan. Moving target defense for embedded deep visual sensing against adversarial examples. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems, SenSys 2019*, pages 124–137, 2019.

[119] Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. In *6th International Conference on Learning Representations, ICLR 2018*, 2018.

[120] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. Striving for simplicity: The all convolutional net. In *3rd International Conference on Learning Representations, ICLR Workshop Track Proceedings*, 2015.

[121] Nedim Srndic and Pavel Laskov. Practical Evasion of a Learning-Based Classifier: A Case Study. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 197–211. IEEE Computer Society, 2014.

[122] Octavian Suciu, Scott E. Coull, and Jeffrey Johns. Exploring adversarial examples in malware detection. In *2019 IEEE Security and Privacy Workshops, SP Workshops 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 8–14. IEEE, 2019.

[123] Octavian Suciu, Radu Marginean, Yigitcan Kaya, Hal Daumé III, and Tudor Dumitras. When does machine learning fail? generalized transferability for evasion and poisoning attacks. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 1299–1316. USENIX Association, 2018.

[124] TESLA. TESLA AI Day. `https://www.youtube.com/watch?v=j0z4FweCy4M`, 2021.

[125] Shixin Tian, Guolei Yang, and Y. Cai. Detecting adversarial examples through image transformation. In *AAAI*, 2018.

[126] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian J. Goodfellow, Dan Boneh, and Patrick D. McDaniel. Ensemble adversarial training: Attacks and defenses. In *6th International Conference on Learning Representations*, 2018.

[127] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 601–618, Austin, TX, 2016.

[128] Jonathan Uesato, Brendan O'Donoghue, Aaron van den Oord, and Pushmeet Kohli. Adversarial risk and the dangers of evaluating against weak attacks, 2018.

[129] Jonathan Uesato, Brendan O'Donoghue, Aaron van den Oord, and Pushmeet Kohli. Adversarial risk and the dangers of evaluating against weak attacks, 2018.

[130] Qinglong Wang, Wenbo Guo, Kaixuan Zhang, Alexander Ororbia, Xinyu Xing, Xue Liu, and C. Giles. Adversary resistant deep neural networks with an application to malware detection. pages 1145–1153, 08 2017.

[131] Zhongjie Wang, Yue Cao, Zhiyun Qian, Chengyu Song, and Srikanth V. Krishnamurthy. Your state is not mine: A closer look at evading stateful internet censorship. In *Proceedings of the 2017 Internet Measurement Conference*, IMC '17, page 114–127, New York, NY, USA, 2017. Association for Computing Machinery.

[132] A. Warnecke, D. Arp, C. Wressnegger, and K. Rieck. Evaluating explanation methods for deep learning in security. In *2020 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 158–174, 2020.

[133] Philipp Winter and Stefan Lindskog. How the Great Firewall of China is blocking Tor. In *Free and Open Communications on the Internet*. USENIX, 2012.

[134] Eric Wong and J. Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope, 2018.

[135] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Yuyin Zhou, Lingxi Xie, and Alan L. Yuille. Adversarial examples for semantic segmentation and object detection. In *IEEE International Conference on Computer Vision, ICCV 2017*, pages 1378–1387, 2017.

[136] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5987–5995, 2017.

[137] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *Proceedings 2018 Network and Distributed System Security Symposium*, 2018.

[138] Weilin Xu, Yanjun Qi, and David Evans. Automatically evading classifiers: A case study on PDF malware classifiers. In *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*, 2016.

[139] Wei Yang, Deguang Kong, Tao Xie, and Carl A. Gunter. Malware detection in adversarial settings: Exploiting feature evolutions and confusions in android apps. In *Proceedings of the 33rd Annual Computer Security Applications Conference, Orlando, FL, USA, December 4-8, 2017*, pages 288–302, 2017.

[140] Liu X. Li C. J Ye, H. Dscae: a denoising sparse convolutional autoencoder defense against adversarial examples. *Ambient Intell Human Comput 13, 1419–1429 (2022).*, 2022.

[141] Valentina Zantedeschi, Maria-Irina Nicolae, and Ambrish Rawat. Efficient defenses against adversarial attacks. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 39–49, 2017.

[142] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.