ABSTRACT

Title of dissertation: THE LIMITATIONS OF DEEP LEARNING METHODS IN REALISTIC ADVERSARIAL SETTINGS

Yigitcan Kaya
Doctor of Philosophy, 2023

Dissertation directed by: Professor Tudor Dumitraș
Department of Electrical and Computer Engineering

The study of adversarial examples has evolved from a niche phenomenon to a well-established branch of machine learning (ML). In the conventional view of an adversarial attack, the adversary takes an input sample, e.g., an image of a dog, and applies a deliberate transformation to this input, e.g., a rotation. This then causes the victim model to abruptly change its prediction, e.g., the rotated image is classified as a cat. Most prior work has adapted this view across different applications and provided powerful attack algorithms as well as defensive strategies to improve robustness.

The progress in this domain has been influential for both research and practice and it has produced a perception of better security. Yet, security literature tells us that adversaries often do not follow a specific threat model and adversarial pressure can exist in unprecedented ways. In this dissertation, I will start from the threats studied in security literature to highlight the limitations of the conventional view and extend it to capture realistic adversarial scenarios.

First (Chapter 2), I will discuss how adversaries can pursue goals other than hurting the predictive performance of the victim. In particular, an adversary can wield adversarial examples to perform denial-of-service against emerging ML systems that rely on input-adaptiveness for efficient predictions. Our attack algorithm, DeepSloth, can transform the inputs to offset the computational benefits of these systems. Moreover, an existing conventional defense is ineffective against DeepSloth and poses a trade-off between efficiency and security.

Second (Chapter 3), I will show how the conventional view leads to a false sense of security for anomalous input detection methods. These methods build modern statistical tools around deep neural networks and have shown to be successful in detecting conventional adversarial examples. As a general-purpose analogue of blending attacks in security literature, we introduce the Statistical Indistinguishability Attack (SIA). SIA bypasses a range of published detection methods by producing anomalous samples that are statistically similar to normal samples.

Third, and finally (Chapter 4), I will focus on malware detection with ML, a domain where adversaries gain leverage over ML naturally without deliberately perturbing inputs like in the conventional view. Security vendors often rely on ML for automating malware detection due to the large volume of new malware. A standard approach for detection is collecting runtime behaviors of programs in controlled environments (sandboxes) and feeding them to an ML model. I have first observed that a model trained using this approach performs poorly when it is deployed on program behaviors from realistic, uncontrolled environments, which gives malware authors an advantage in causing harm. We attribute this deterioration to distribution shift and investigate possible improvements by adapting modern ML techniques, such as distributionally robust optimization.

Overall, my work has reinforced the importance of comprehensive threat models and applications with well-documented adversaries for properly assessing the security risks of ML.

THE LIMITATIONS OF DEEP LEARNING METHODS IN REALISTIC ADVERSARIAL SETTINGS

by

Yigitcan Kaya

Submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2023

Advisory Committee:
Professor Tudor Dumitraș, Chair/Advisor
Professor Dana Dachman-Soled, Dean's Representative
Professor Tianyi Zhou
Professor Leonidas Lampropoulos
Professor Furong Huang
Professor David Wagner
Doctor Krishnaram Kenthapadi

To my partner Mae who has always kept my body and my soul well nourished.

Acknowledgments

Before starting to write my own, I took a sneaky peek at the acknowledgments my dear friends Octavian and Sanghyun wrote for their dissertations. I thought to myself, wow, they have a lot of people listed here, I don't think I will have that many. After all, they are more active, well-connected, and social than I am. Then I started making a list of everyone I felt grateful to have met and interacted with along this journey that is PhD. To my surprise, the list just kept growing and growing and growing. Making this list, looking back on my last seven years, reminiscing, and scrolling through the old photos at the forgotten corners of my phone was bittersweet. I feel great joy that I am so close to earning the degree I came to this country for and excited about what the future holds in store for me, but I also can't help but feel some confusing grief that this fantastic page of my life is coming to an end. To quote Winnie The Pooh, *How lucky am I to have something that makes saying goodbye so hard*. While I am bound to forget some names below, I will do my best to acknowledge the people who made my journey feel like a breeze.

First of all, I am deeply grateful to my parents, Sevgi and Ejder, I am fortunate to be their son. Their sacrifices, love, and support brought me to where I am right now. From a very young age, they have allowed me to foster and satisfy my curiosity about computers. More importantly, my parents raised me to be honest, compassionate, kind, and hard-working, I hope I've lived up to that so far. I also owe my older brother, Oguzhan, a great deal for putting up with my little brother shenanigans for so long and always sharing his pearls of wisdom when needed.

My life profoundly changed when my advisor, Tudor Dumitras, offered me a research internship with him in 2016 when I was a junior undergraduate student at Bilkent. I knew precisely nothing about real research till then. Tudor helped me realize that I want to pursue a PhD and, more importantly, that I can do it. Tudor has been shaping me as a researcher ever since by always challenging me, guiding me to be a great judge of ideas, helping me acquire many skills most PhD students never do, and fostering an environment that created close friendships in our group. I would like to thank him for all his efforts and patience with me and for giving me an opportunity to spend seven wonderful years in his group doing impactful research. I also have to acknowledge my fellow interns in 2016, Tibi, Alina, and Razvan. We had a fun summer with lots of adventures, I hope our paths cross once again in the future.

My favorite part of the PhD was always the social aspects and the camaraderie in our group. In no particular order, I would like to thank my fellow students for their friendship, BumJun, Ziyun, Doowon, Sanghyun, Octavian, Erin, Michi, Shoumik, and Kamala. Erin, I will always remember our late-night lab crunches, sorry I let you down by abandoning the religion of Iron as of late. Michi, I hope one day when we are old we can get Mongolian shrimp again at the dining hall and waste hours talking. Sanghyun, I truly loved our whiskey-fueled research (and life) talks, I feel lucky to be your friend, oh the ideas we came up with but never worked on. And lastly Octavian, I always believed if it wasn't for him, I wouldn't have come back after my internship for a PhD. Octavian always looked out for me as an older brother would, helped me with all areas of life, held my hand when I needed it the most and threw mad BBQ parties to feed the hungry masses, he is an amazing friend, and a kind, generous, and sometimes fashionably grumpy soul.

My work was also made possible by the UMD staff, in particular, Dana Purcell and Tom Hurst, who took away many of the administrative burdens and answered many of my mundane

questions. Sarah and Eric, my sponsors at the DoD, funded most of my PhD and listened to me ramble for hours on end, thank you. I had two amazing internships at AWS and learned a lot about how things work in the industry. I would like to acknowledge and thank all my mentors and managers at AWS that made this possible: Homer, Ali, Krishnaram, Bilal, Nathalie, and Sergul.

I had the opportunity to work with and get advice from many accomplished professors: Tom Goldstein, Dana Dachman-Soled, Hal Daume III, Cristiano Giuffrida, Nicolas Papernot, Fabio Pierazzi, and Yizheng Chen. I'm particularly grateful that I got to work with David Wagner and Lorenzo Cavallaro, who have quickly become my academic role models with their humility, inquisitive minds, accomplishments, and kindness to everyone around them. A special shout-out to Feargus who introduced me to Lorenzo, I loved our long (and hectic) brainstorming sessions while working on TrojAI. Dave Levin gave me a chance to teach in his class and helped me build confidence in my teaching skills (that I thought didn't exist). There are also the members of my dissertation committee, Leonidas Lampropoulos, Tianyi Zhou, and Furong Huang who prepared me for this last step with their feedback and guidance.

Tudor's lab was unique as we had the chance to work closely with interns from all around the world every summer. Our interns brought so much energy and joy to the lab, which made the summer my favorite season (despite hating Maryland summers). I loved working with them and thanks to them I became more confident that I can be a halfway decent professor; they helped me grow tremendously as a mentor, advisor, and friend. Some of these interns are, Ionut, Simge, Georgios, Omer, Yavuz, David, Stefan, Cristi, Goktug, Radu, Maria, Sriram, Tasos, and many others, I'm proud to have known them all.

I am grateful to my friends in the US, who have become my family away from home and never made me feel alone. Eden and Lorraine (go River Rats!), Taiwo, Kenny, Vishal, Nohelia,

my roommate Yekanth, former roommate Kayla, Cristina and Sorin (thanks for hosting me twice in California!)...love y'all. My longtime friends from Turkey, Irem and Omer, Ahmet (♡), Ali and Pinar, Alp, Ilteris, Suleyman, Fatih... it gives me great joy that despite the distances we were able to keep up. Thank you all for going through this journey with me.

As I am closing this chapter of my life, I am thrilled to start the next one as a postdoc at UC Santa Barbara. Giovanni Vigna, Chris Kruegel, and all members of SecLab were incredibly warm, supportive, and welcoming, I cannot wait to learn from you and do great things with your guidance. Another shout-out to Aravind Machiry, my summer roommate in 2019, he very kindly connected me with Giovanni and Chris.

Finally, I thank my partner Mae with all my heart, this dissertation is dedicated to you. Mae helped me to find joys in life that I had been too blind to see, her ability to find beauty in little things has been (and will be) a great source of inspiration on this journey, I'm looking forward to the adventures we will have together, starting in Santa Barbara.

Table of Contents

List of Tables

List of Figures

Publications

The content of this dissertation includes ideas and findings previously presented in the following publications. An asterisk (*) denotes joint lead authorship.

1. Kaya Y., Hong S., Dumitras T. Shallow-Deep Networks: Understanding and Mitigating Network Overthinking. In *Proc. of International Conference on Machine Learning (ICML) 2019*

2. Hong S.*, Kaya Y.*, Modoranu IV, Dumitras T. A Panda? No, It's a Sloth: Slowdown Attacks on Adaptive Multi-Exit Neural Network Inference. In *Proc. of International Conference on Learning Representations (ICLR) 2021*

3. Kaya Y., Zafar MB., Aydore S., Rauschmayr N., Kenthapadi K. Generating distributional adversarial examples to evade statistical detectors. In *Proc. of International Conference on Machine Learning (ICML) 2022*

4. Kaya Y., Chen Y., Saha S. Pierazzi F., Cavallaro L., Wagner D., Dumitras T. Are Sandboxes Enough for Malware Detection in the Wild?. *Preprint*

Additionally, the author has contributed to the following publications which further discuss some of the concepts presented in this dissertation.

5. Suciu O., Marginean R., Kaya Y., Daume HDIII, Dumitras T. When does machine learning FAIL? generalized transferability for evasion and poisoning attacks. In *Proc. of USENIX Security Symposium 2018*

6. Hong S., Chandrasekaran V., Kaya Y., Dumitras T., Papernot N. On the effectiveness of mitigating data poisoning attacks with gradient shaping. Preprint

7. Hong S., Frigo P., Kaya Y., Giuffrida C., Dumitras Terminal Brain Damage: Exposing the Graceless Degradation in Deep Neural Networks Under Hardware Fault Attacks. In *Proc. of USENIX Security Symposium 2019*

Research for publications #1–2 and #4–7 carried out while the author was pursuing a PhD at the University of Maryland. Research for publication #3 was carried out while the author was an applied science intern at Amazon Web Services.

Chapter 1: Introduction

Machine learning is rapidly changing the world and every aspect of our day-to-day lives. Availability of computational resources, large-scale data repositories, and algorithmic innovations have paved the way for a constant stream of breakthroughs especially in areas such as computer vision or natural language processing. These advances are fueling a new era in technology that was once imagined in science fiction works, including autonomous vehicles, simultaneous translators, and human-like chatbots with enormous capabilities.

We already have started seeing the profound societal effects of this technology. Learning-based technologies increase efficiency, scalability, and automation in many critical industries, including healthcare, finance, and cybersecurity. Although such advances might improve working conditions and human well-being, they can also open up new doors for mass surveillance, censorship, disinformation campaigns, cyber-attacks or replace workers or artists in the name of maximizing profits.

Despite their clear power and potential, machine learning methods are prone to catastrophic failures due to their enigmatic nature and brittleness. A recent example of such failure is the case of Microsoft's chatbot Tay, briefly released in 2016. Tay was designed to learn from user interactions on Twitter and provide personalized responses. However, shortly after its release, Tay started posting offensive and inflammatory messages on Twitter. Tay, being intentionally fed racist

and hateful comments by users, changed its behavior as it was not designed to operate in such an adversarial environment. Microsoft had to shut down Tay within 24 hours and apologized for its unintended behavior. This incident highlights how well-performing machine learning methods malfunction in unexpected ways when they are deployed in adversarial settings.

Such real-world failures and rapidly growing applications of machine learning in security-critical settings, such as malware detection [1], have given rise to studying machine learning from an adversarial lens. Through designing realistic threat models that characterize the potential risks an application, such as Tay, might encounter, adversarial machine learning research aims to understand the properties of machine learning models that lead to undesirable, worst-case outcomes. In the case of Tay, the chatbot was the victim of a *poisoning* attack where the adversaries taint the training data of a model with carefully generated malicious data that aims to teach the model an undesirable or incorrect behavior, such as producing racist utterances.

Another heavily-studied threat model, called *adversarial examples*, exploits the fact that modern deep neural network-based machine learning models are sensitive to small perturbations to their run-time inputs. This sensitivity enables an adversary to add imperceptible changes to the input, e.g., changing a few pixels in an image of a lemon, that misleads the model into making an incorrect decision, e.g., the lemon is classified as an orange. This threat model is relevant when the adversary is unable to manipulate the model's training data but has some limited ability to modify its run-time inputs.

The early works in adversarial examples have established the threat model that has become conventional [2]. In this threat model, the adversary starts with a *clean* input that the victim machine learning model classifies correctly, e.g., an image of a lemon. The **goal** of the adversary is causing the model to misclassify this input, e.g., the lemon is classified as an orange. The **leverage**

of the adversary is being able to apply a *transformation* to the input, e.g., Gaussian blur. While applying this transformation, the adversary has specific **constraints** that depend on the application and the adversary's capabilities, e.g., the lemon should not be too blurry.

This threat model defines most works in adversarial examples that study these dimensions, i.e., goals, leverage, and constraints, individually or together. The defenses that are designed to prevent attacks in this threat model, such as adversarial training [3], have been influential and seen commercial applications in practice [1] Overall, works following this threat model are common in top-tier machine learning and security conferences and have been impactful in shaping the security perceptions of the community.

In this dissertation, along its three dimensions, we investigate the limitations of this conventional threat model in representing the security risks machine learning faces in realistic adversarial settings. The main theme of our work is bridging the classical threat models in cyber security literature and recent developments in machine learning. In particular, we study how **old becomes new**, i.e., how old threat models in security re-emerge in different forms to pose risks in machine learning.

Motivation: How Old Becomes New

In security literature, *what is old is new again* is an age-old adage [2]. Real-world adversaries do not wish to reinvent the wheel when there is a new technological advance and opt for leveraging existing infrastructure and older threats to maximize their opportunities. A threat model keeps reappearing in different forms and in various applications as long as practitioners make the same

[1] Robust Intelligence

[2] SecurityWeek – Malware Trends: What's Old Is Still New

mistakes that have given rise to it initially.

Mirai botnet [3] is a recent example that perfectly demonstrates this adage. In 2016, Mirai infected over 200 thousand devices (over 60 thousand on its first day), which has been used to launch some of the largest and most disruptive DDoS attacks, for example, against Rutgers University. With the advent of Internet-of-Things (IoT), there has been a significant increase in internet-connected devices. Mirai, by primarily infecting IoT devices, was able to create an enormous botnet. Mirai relied on a very simple old threat model to infect these new and untested IoT devices: password guessing. Mirai first scanned the Internet for IoT devices that run on the ARC processor, which runs a stripped-down version of the Linux operating system. Then Mirai simply guessed the default username-and-password combos to log in to an IoT device, which often was successful as the device owners did not change them. Password security is an active area of research since the 1990s [4] and the community has made tremendous progress in developing secure password schemes that are also user-friendly. These schemes have been developed through a series of incidents, each has provided new insights and better practices. However, IoT, as a new and exciting technology, has ignored these practices (by keeping default passwords for accessing the admin interface of a device). Mirai has exploited this old and classical threat model in a new application despite the experiences of the security community, i.e., what is old is new again.

Throughout our dissertation, we will explore similar settings where emerging machine learning, in particular, deep learning, methods have reinvigorated old and classical threat models in cyber security. More importantly, these threat models cannot be captured by the conventional adversarial example threat model that is predominantly studied in the machine learning literature. These shortcomings limit the community's ability to make progress in developing more secure

[3] Cloudflare – What is the Mirai Botnet?

and reliable machine learning models. This motivates us to highlight possible extensions to the conventional threat models to be more inclusive, realistic, and representative of the tendencies of real-world adversaries.

Dissertation Structure

This dissertation is structured in five chapters. We begin here, in Chapter 1, providing an introduction, an outline of the problem, and the motivation behind our work.

In Chapter 2, we will discuss an **emerging goal** an adversary can pursue wielding adversarial examples: performing denial-of-service against input-adaptive machine learning systems that enable efficient inferences. We implement DeepSloth as an attack algorithm that can transform the inputs to offset the computational benefits of these systems. Denial-of-service attacks have remained a disruptive tool in the arsenal of realistic adversaries for decades in the cyber security literature. However, the conventional adversarial examples threat model, by only considering hurting predictive performance as a goal, cannot represent this threat.

Attributions:

The findings in Chapter 2 have been previously published in:

1. Kaya Y., Hong S., Dumitras T. Shallow-Deep Networks: Understanding and Mitigating Network Overthinking. In *Proc. of International Conference on Machine Learning (ICML) 2019*

2. Hong S.*, Kaya Y.*, Modoranu IV, Dumitras T. A Panda? No, It's a Sloth: Slowdown Attacks on Adaptive Multi-Exit Neural Network Inference. In *Proc. of International*

Conference on Learning Representations (ICLR) 2021

The first publication describes the design and benefits of a prototypical input-adaptive neural network architecture. In this work, Kaya designed the architecture, defined the concept of overthinking, and performed all evaluations. Hong conducted experiments to understand the connections between input-adaptiveness and backdoor attacks against neural networks. Dumitras provided overall supervision throughout the project. The second publication designs and evaluates our DeepSloth attack against a range of input-adaptive architectures and explores defenses to this threat. In this work, Kaya designed the final version of the DeepSloth attack algorithm, including its loss function, and performed its evaluation in white-box and black-box settings against various input-adaptive architectures. Hong designed the initial idea of DeepSloth, provided supervision, evaluated the effectiveness of DeepSloth against the adversarial training defense, and finally conducted the denial-of-service experiments in the partitioned model scenario. Modoranu implemented the initial version of DeepSloth and performed proof-of-concept experiments to show its effectiveness. Dumitras provided overall supervision throughout the project.

In Chapter 3, we will describe a **realistic constraint** for improving the conventional adversarial examples threat model. Our constraint, implemented by Statistical Indistinguishability Attack (SIA), is inspired by traditional blending attacks in security literature [5, 6]. Simply put, SIA is engineered to produce adversarial examples that are statistically similar to normal samples. This allows SIA to bypass a range of published statistical detection methods that have been produced as a defense.

Attributions:

The findings in Chapter 3 have been previously published in:

1. Kaya Y., Zafar MB., Aydore S., Rauschmayr N., Kenthapadi K. Generating distributional adversarial examples to evade statistical detectors. In *Proc. of International Conference on Machine Learning (ICML) 2022*

In this work, Kaya proposed the idea, designed and implemented SIA, performed a systematical evaluation, and wrote the paper. Zafar provided close supervision through weekly meetings and helped Kaya to refine the idea. Aydore, Rauschmayr, and Kenthapadi acted as mentors to Kaya during his internship at Amazon Web Services and provided overall supervision and feedback to the project.

In Chapter 4, we will investigate malware detection with program behaviors: a popular application for machine learning where adversaries can gain **natural leverage** over a model. In the conventional threat model, adversaries *perturb* an input to a model deliberately and carefully to pursue their goals, e.g., the input being misclassified. A typical malware detector is trained on program behaviors collected from a sandbox, i.e., a controlled execution environment. However, a program's behaviors can vary across different environments. We find that this variability, especially between a sandbox and a real-world environment, places an adversarial pressure on detectors that are often not robust to the variability. This pressure is an inevitable and *natural* outcome of the domain rather than a deliberate effort of the adversary as it is in the conventional threat model.

Attributions:

The findings in Chapter 4 are presented in the following preprint manuscript that is under submission at the time this dissertation is submitted.

1. Kaya Y., Chen Y., Saha S. Pierazzi F., Cavallaro L., Wagner D., Dumitras T. Are Sandboxes

Enough for Malware Detection in the Wild?. *Preprint*

In this work, Kaya has collected and processed the datasets, identified the project's contributions, performed experiments and evaluations and implemented machine learning methods to improve malware detection. Saha performed some experiments and helped with generating the plots and figures to present the results. Chen, Wagner, Pierazzi, and Cavallaro acted as mentors to Kaya and provided feedback on the results, research direction, contributions, and manuscript. Dumitras provided overall supervision throughout the project and enabled Kaya to access a dataset on real-world program behaviors.

Finally, in Chapter 5, we provide a conclusion to this dissertation.

Chapter 2: Emerging Adversarial Goals Against Machine Learning

Recent increases in the computational demands of deep neural networks (DNNs), combined with the observation that most input samples require only simple models, have sparked interest in *input-adaptive* multi-exit architectures, such as MSDNets or Shallow-Deep Networks. These architectures enable faster inferences and could bring DNNs to low-power devices, *e.g.*, in the Internet of Things (IoT).

In this chapter, we first present a prototypical multi-exit architecture: shallow-deep networks (SDNs). SDNs allow us to define and measure a pathology of of neural networks we call *overthinking*. We discuss the benefits of SDNs for both model understanding by analyzing overthinking across different computer vision architectures and tasks and for inference efficiency by utilizing input adaptiveness.

This new ability and the resulting computational paradigms, such as partitioned deep learning models at IoT devices, make it critical to ask: are the computational savings provided by such approaches robust against adversarial pressure? In particular, an adversary may aim to slow down adaptive DNNs by increasing their average inference time—a threat analogous to the *denial-of-service* attacks from the Internet. This threat of slowdown provides an emerging goal for the adversary that has not been captured by the conventional adversarial examples threat model.

In the second part of this chapter, we conduct a systematic evaluation of this threat by

experimenting with generic SDN-based multi-exit DNNs (based on VGG16, MobileNet, and ResNet56) and a custom multi-exit architecture, on two popular image classification benchmarks (CIFAR-10 and Tiny ImageNet). This evaluation reveals that adversarial example-crafting techniques can be modified to cause slowdowns, and we propose a metric for comparing their impact on different architectures. We show that a slowdown attack reduces the efficacy of multi-exit DNNs by 90–100%, and it amplifies the latency by 1.5–5× in a typical IoT deployment. We also show that it is possible to craft universal, reusable perturbations and that the attack can be effective in realistic black-box scenarios, where the attacker has limited knowledge about the victim. Finally, we show that adversarial training provides limited protection against slowdowns. Our results in this chapter suggest that further research is needed for defending multi-exit architectures against this emerging threat.

2.1 A Promising Machine Learning Paradigm: Input-Adaptive Neural Networks

The inference-time computational demands of deep neural networks (DNNs) are increasing, owing to the "going deeper" [7] strategy for improving accuracy: as a DNN gets deeper, it progressively gains the ability to learn higher-level, complex representations. This strategy has enabled breakthroughs in many tasks, such as image classification [8] or speech recognition [9], at the price of costly inferences. For instance, with 4× more inference cost, a 56-layer ResNet [10] improved the Top-1 accuracy on ImageNet by 19% over the 8-layer AlexNet. This trend continued with the 57-layer state-of-the-art EfficientNet [11]: it improved the accuracy by 10% over ResNet, with 9× costlier inferences.

The accuracy improvements stem from the fact that the deeper networks *fix* the mistakes

Figure 2.1: **Simple to complex inputs.** Some Tiny ImageNet images a VGG-16 model can correctly classify if computation stops at the 1^{st}, 5^{th}, and 14^{th} layers.

of the shallow ones [12]. This implies that some samples, which are already correctly classified by shallow networks, do not necessitate the extra complexity. This observation has motivated research on *input-adaptive* mechanisms, in particular, multi-exit architectures [13, 12, 14, 15]. Multi-exit architectures save computation by making input-specific decisions about bypassing the remaining layers, once the model becomes confident, and are orthogonal to techniques that achieve savings by permanently modifying the model [16, 17, 18, 19]. Figure 2.3 illustrates how a multi-exit model [14], based on a standard VGG-16 architecture, correctly classifies a selection of test images from 'Tiny ImageNet' before the final layer. We see that more typical samples, which have more supporting examples in the training set, require less depth and, therefore, less computation.

2.1.1 A Prototypical Input-Adaptive Multi-Exit Architecture

As Figure 2.3 shows, the internal layers of a neural network contain a wealth of information regarding the model's decision process. Here we present the Shallow-Deep Network (SDN) as a generic modification to off-the-shelf DNNs for introducing *internal classifiers* (ICs) that allow us to explicitly monitor this information. Our modification attaches ICs to various stages of the forward pass of a DNN, as Figure 2.2 illustrates, and effectively combines shallower and deeper networks into one. The feature reduction in an IC acts as a regularizer and ensures that our method can scale up to large DNNs. We can apply the modification to both pre-trained and untrained DNNs. The conversion from a pre-trained DNN is efficient as we only train the parameters in the ICs. Moreover, by using a weighted loss function, we can also train the original network from scratch jointly with the ICs. This mode of training, the *SDN training*, often improves the original accuracy and yields more accurate ICs. Our modification is practical for a range of existing architectures and allows us to explore the overthinking problem. We refer the readers to our work published at ICML 2019 [14] for further technical details on this modification, including how the ICs are designed and trained.

2.1.2 Technical Setup

Setting. We consider the supervised classification setting with standard feedforward DNN architectures. A DNN model consists of N blocks, or layers, that process the input sample, $x \in \mathbb{R}^d$, from beginning to end and produce a classification. A classification, $F(x, \theta) \in \mathbb{R}^{\mathcal{K}}$, is the predicted probability distribution of x belonging to each label $\{1, 2, ..., \mathcal{K}\}$. Here, θ denotes the tunable parameters, or the weights, of the model, which we generally omit if it is clear from

the context. $F^j(x)$ denotes the predicted probability of x belonging to class j. The parameters are learned on a training set $\mathcal{D}$ that contains multiple (x_i, y_i) pairs; where y_i is the ground-truth label of the training sample x_i. We use θ_i to denote the parameters at and before the i^{th} block; *i.e.*, $\theta_i \subset \theta_{i+1}$ and $\theta_N = \theta$. Once a model is trained, its performance is then tested on a set of unseen samples, $\mathcal{S}$. A multi-exit model, such as an SDN, contains M *exit points*—internal classifiers—attached to a model's hidden blocks. We use F_i to denote the i^{th} exit point, which is attached to the j^{th} block. Using the output of the j^{th} ($j < N$) block on x, F_i produces an internal classification, i.e., $F_i(x, \theta_j)$, which we simply denote as $F_i(x)$.

Datasets. In our experiments, we use two computer vision datasets for benchmarking: CIFAR-10 and Tiny ImageNet. The CIFAR dataset consist of 32x32 pixels, colored natural images. CIFAR-10 images are drawn from 10 classes; containing 50,000 training and 10,000 validation images. We use a standard data augmentation scheme: padding, random cropping, and random horizontal mirroring. The Tiny ImageNet dataset consists of a subset of ImageNet images [20], resized at 64x64 pixels. There are 200 classes, each of which has 500 training and 50 validation images. We augment the training data with random crops, horizontal mirroring, and RGB intensity scaling.

Architectures and Hyper-Parameters. We experiment with three off-the-shelf CNNs: VGG [21], ResNet [22] and MobileNet [23]. Specifically, we use VGG-16 and ResNet-56 configurations of these architectures. We train the CNNs for 100 epochs, using the hyper-parameters the original studies describe. In our experiments, we use these CNNs as our pre-trained, off-the-shelf networks. We denote the network after our SDN modification by appending *'SDN'* to its original name; e.g. *VGG-16* becomes *VGG-16-SDN*. To *convert* a pre-trained network to an SDN, we freeze original weights and train only the weights in the attached ICs for 25 epochs, using the Adam

optimizer [24]. Freezing prevents any change to the off-the-shelf network, and, as a result, keeps its predictions fully intact.

Metrics. To quantify the test-time inference cost of a network, we measure the average number of floating point operations (*FLOPs*) a network performs to classify an input. As for the classification performance, we simply report the Top-1 accuracy on the test data (as a percentage).

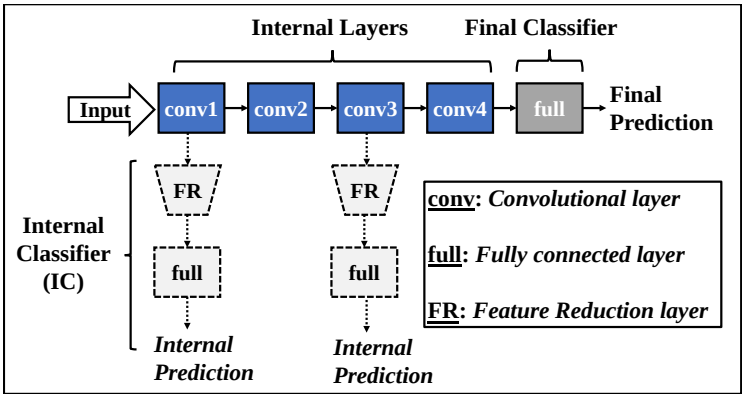

Figure 2.2: Modifying a standard convolutional neural network as a Shallow-Deep Network. Our modification introduces the dotted components—two ICs. The resulting network produces three predictions: two internal and one final.

2.1.3 The Overthinking Problem

Shallow-Deep Networks, by providing explicit internal predictions, allow us to study the overthinking problem in CNNs. In the context of an SDN, we define overthinking on an input sample as the network's ability to reach a correct internal prediction. Formally, the network overthinks on the input sample (x, y) if $\text{argmax}_{j \in \{1,\dots,K\}} F_i^j(x) = y$, i.e. the i^{th} internal classification of the model—which we will denote as $\hat{y}_i(x)$—is a correct one.

Overthinking is problematic as the rest of the forward pass ends up being invoked for no clear

benefit. The computational waste this leads to is *the wasteful effect* of overthinking (Section 2.1.4). Moreover, in some cases, excessive processing ultimately leads to a misclassification, i.e. $\hat{y}_i(x) = y \neq \hat{y}_{final}(x)$, where $\hat{y}_{final}(x)$ is the model's prediction on the input at its final classifier. We categorize this as *the destructive effect* of overthinking (Section 2.1.5). Note that, as opposed to *overfitting*, overthinking leads to both wasted computation and misclassifications. Further, whereas regularization, such as Dropout [25], can mitigate overfitting; there is no general remedy for overthinking.

To illustrate the overthinking problem in Sections 2.1.4 and 2.1.5, we focus on a case study: *VGG-16* network pre-trained on Tiny Imagenet task (59% test accuracy). We convert this network to an SDN—to *VGG-16-SDN*—by freezing the original weights and training only the ICs. The conversion allows us to monitor how the original network's predictions evolve, without altering them at all. We select a subset of the internal layers of VGG-16 (or another DNN we are modifying) to attach the ICs. Our selection criterion is simple: we pick the internal layers closest to the 15%, 30%, 45%, 60%, 75% and 90% of the full network's inference cost—the number of *FLOPs*. Given an input, our SDNs produce a total of 6 internal predictions and a final prediction. For each IC, we denote the *relative inference cost to the whole network* as $\mathcal{C}_i : 1 \leq i \leq M$. In particular, for our SDNs, $M = 6$ and $\mathcal{C}_1 \approx 0.15 ... \mathcal{C}_6 \approx 0.9$. For the final classifier: $\mathcal{C}_{final} = 1$. Overall, this simplifies our analysis of overthinking and eliminates the need for tuning.

2.1.4 The Wasteful Effect of Overthinking

Quantifying the Effect. For *VGG-16-SDN*, assuming that a sample exits at an IC right after a correct internal prediction; 24%, 15%, 14%, 6%, 8% and 4% of the test samples would exit at

each IC. The remaining 29% of the samples exit at the final classifier—4% correctly and 25% wrongly classified. Based on these *ideal* exit rates and $\mathcal{C}_i$'s (the IC's relative inference costs); the average inference cost would decrease by 43% as a result of eliminating overthinking. Further, on CIFAR-10, 95% of the samples do not require the full depth of a CNN —81% on CIFAR-100 and 69% on Tiny ImageNet—; whereas the rest exit at the final classifier. In consequence, eliminating overthinking would cut the average inference costs by 73% on CIFAR-10, by 55% on CIFAR-100, and by 40% on Tiny ImageNet. Evidently, overthinking leads to more severe waste of computation as the task gets less complex. Note that our assumption here is impractical as it requires a perfect way to discern a wrong classification from a correct one—Section 2.1.6 presents a realistic way.

Explaining the Behavior. Considering the perfect exit rates, 29% of the samples still need to be forwarded until the final classifier; whereas 24% can exit at the first IC. Figure 2.3 shows randomly sampled images—the input samples—that exit at each IC in *VGG-16-SDN*. The first column presents the samples that are correctly classified at the first IC (IC_1), and so on. In subsequent columns of the figure, notice how the input *complexity* progressively increases. These images suggest that the earlier ICs learn simple representations that allow them to recognize typical samples from a class, but that fail on atypical samples, e.g. where the subject is occluded or zoomed out. Overthinking leads to wasted computation as the network applies unnecessarily complex representations to classify the input. On the other hand, not having enough representational complexity causes wrong predictions, e.g. the IC_1 misclassifies the samples in the remaining columns of the figure.

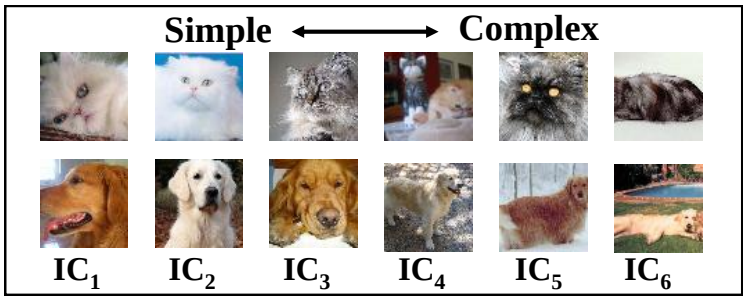

Figure 2.3: Sample images from the Tiny ImageNet classes *Persian Cat* (top row) and *Golden Retriever* (bottom row). Each column presents the samples that are correctly classified first at the given IC. We can see how the samples get progressively complex over ICs. The network is *VGG-16-SDN*, trained using the IC-only training strategy.

2.1.5 The Destructive Effect of Overthinking

Quantifying the Effect. Out of 41% of the samples *VGG-16* misclassifies; 3% are actually correctly classified first at IC_1 of *VGG-16-SDN*—4% at IC_2, 3% at IC_3, 2% at IC_4, 3% at IC_5, and 1% at IC_6. As a result, the *cumulative accuracy* of *VGG-16-SDN* is 75%—16% higher than the original accuracy. In cumulative accuracy, we consider a sample to be correctly classified if there is *at least one* correct internal prediction—or the final prediction—of the SDN. This metric measures the ideal accuracy a network can reach after eliminating overthinking. The difference between the cumulative and the original accuracies quantifies the destructive effect: 4% on CIFAR-10, 13% on CIFAR-100, and 14% on Tiny ImageNet task. This effect also explains the reason why an IC sometimes outperforms the final classifier: at a certain IC, the destructive effect starts to outweigh the accuracy gain from increased feature complexity; after which the accuracy starts decreasing. Overall, the destructive effect can be seen in up to 50% of all misclassifications a CNN makes.

Figure 2.4: A set of Tiny ImageNet samples only the first internal classifier of *VGG-16-SDN* can correctly classify. For each image, two labels are the correct label and the wrong prediction at the final classifier, respectively.

Explaining the Behavior. Figure 2.4 presents a selection of images that *only* the first IC can correctly classify—we identified 46 of such cases. *VGG-16* and all other ICs in *VGG-16-SDN* misclassify these samples. We hypothesize that these images consist of *confusing* elements, sometimes belonging to more than one class; while the first IC recognizes the objects displayed prominently, subsequent layers discover finer details that are irrelevant. These results suggest that correct classifications require the *appropriate* level of representational complexity for each sample, which further illustrates the utility of the ICs in an SDN.

2.1.6 Mitigating the Wasteful Effect of Overthinking With Confidence-Based Early Exits

After understanding the overthinking problem, in this section, we propose new remedies to mitigate its wasteful effect. For this purpose, we further utilize Shallow-Deep Networks as a vehicle for designing an *early exit* technique. An early exit mechanism allows a multi-exit DNN, such as an SDN, to be input adaptive. By adjusting the exit taken for the complexity of the given input, e.g., a simple input exits at the first IC, it is possible to reduce the inference cost of a DNN for an

average input.

In Section 2.1.4, we show that an ideal, but *impractical*, early exit mechanism could eliminate overthinking entirely. Here, as a *practical* mechanism, we evaluate using the *confidence* of an internal classifier on an input for determining when the network should stop thinking. If none of the internal predictions—or the final prediction—are confident enough to for an exit; the mechanism we have designed outputs the most confident among them. We opt for a simple confidence mechanism for highlighting that we mitigate overthinking regardless of the mechanism, which could be improved, for example, using prediction entropy [12]. To quantify confidence, we use the estimated probability of the sample x belonging to the predicted class, i.e. $\max_j F_i^j(x)$. We deem a prediction confident if this probability exceeds a threshold parameter T_i, that is specific to each IC. These thresholds determine the model should take the i^{th} exit for an input sample. A conservative T_i leads to fewer early exits (and increases the average inference costs) and the opposite hurts the accuracy as the estimate of whether $\hat{y}_i = y$ becomes unreliable. The thresholds facilitate on-the-fly adjustment of the input-adaptiveness on the resource availability and the performance requirements. As utility is a major practical concern, we set T_i's for balancing between efficiency and accuracy. We have observed that setting all thresholds to the same works acceptably. In consequence, in the following experiments, we set all thresholds to the same value for simplicity, i.e., $T_i = q$. We search for a q value ($0 \leq q \leq 1$) that satisfies our average inference cost constraints (in terms of FLOPs) on a small unlabeled holdout set. Here, in addition to applying the SDN modification on a pre-trained DNN (i.e., IC-only training), we also evaluate training the ICs and the weights of the original DNN together (i.e., SDN training). The loss function for SDN training is described in more detail in our publication [14].

DNNs	≤25%	≤50%	≤75%	MAX
		CIFAR-10		
VGG *(93.1)*	84.0 | 85.4	92.8 | **93.2**	**93.2** | **93.4**	**93.2** | **93.4**
RES *(91.9)*	57.6 | 76.2	84.5 | 91.4	91.4 | **92.1**	**91.9** | **92.1**
MOB *(90.8)*	87.9 | 88.2	**91.1** | **91.5**	**91.3** | **91.5**	**91.3** | **91.5**
		TINY IMAGENET		
VGG *(58.6)*	34.4 | 36.8	44.2 | 56.2	**58.5** | **63.1**	**60.4** | **63.4**
RES *(53.9)*	23.9 | 39.0	37.6 | 39.1	49.2 | 52.8	**54.0** | **55.1**
MOB *(59.3)*	36.0 | 45.3	55.5 | 57.3	59.1 | **61.5**	**60.3** | **61.8**

Table 2.1: Comparing the inference costs of the CNNs and SDNs with early exits. **DNNs** column lists the original CNNs and their accuracies. **≤25%**, **≤50%**, and **≤75%** report the early exit accuracy when we limit the average inference cost to at most 25%, 50% and 75% that of the original CNN's. **Max** column reports the highest accuracy early exits can achieve. We highlight the cases where an SDN outperforms the original CNN. In each cell, the left and right accuracies are from the IC-only and the SDN training strategies.

Early Exits Mitigate the Wasteful Effect. Table 2.1 presents the early exit accuracy of our SDNs with a limited computational budget, in two training strategies. We calculate the computational cost when the input samples are evaluated individually; similar to previous studies [26, 27]. We measure the accuracy under three budget settings: not exceeding 25%, 50% and 75% of the original CNN's inference cost. We control an SDN's average inference cost by adjusting the threshold parameter q. We also report the highest accuracy early exits can achieve, without any specific budget. First, the SDN training improves the early exit accuracy significantly; exceeding the

original accuracy while reducing the inference costs more than 50% on CIFAR-10 and CIFAR-100 tasks. Even with the IC-only training strategy, early exits are still effective; allowing more than 25% reduced inference cost. In our most complex task, Tiny ImageNet, early exits can reduce the cost by more than 25%, usually without any accuracy loss. Overall, an SDN's early exits can turn a standard DNN into an input-adaptive one, mitigate the wasteful effect of overthinking and provide significant improvements in inference efficiency.

2.2 Adversarial Slowdown Attacks on Input-Adaptive Inference

In Section 2.1, we have evaluated a prototypical multi-exit architecture and demonstrated the benefits of input-adaptive neural networks, both in terms of insights and computational savings. However, it is unknown if the computational savings provided by this promising capability are robust against adversarial pressure. As we discuss in Section 1, DNNs are vulnerable to a wide range of attacks, including adversarial examples that involve imperceptible input perturbations [28, 29, 30, 15]. Considering that a multi-exit model, on the worst-case input (i.e., a complex input), does not provide any computational savings, in this section, we ask: *Can the savings from multi-exit models be maliciously negated by input perturbations?* As some natural inputs do require the full depth of the model, it may be possible to craft adversarial examples that delay the correct decision; it is unclear, however, how many inputs can be delayed with imperceptible perturbations. Furthermore, it is unknown if universal versions of these adversarial examples exist, if the examples transfer across multi-exit architectures and datasets, or if existing defenses (*e.g.* adversarial training) are effective against slowdown attacks.

In this section, we consider an extension to the conventional adversarial example threat

model by defining an **emerging goal**: slowing down the inference of input-adaptive DNNs. This new threat is analogous to the *denial-of-service (DoS) attacks* that have been plaguing the Internet for decades. By imperceptibly perturbing the input to trigger this worst-case, the adversary aims to slow down the inferences and increase the cost of using the DNN. This is an important threat for many practical applications, which impose strict limits on the responsiveness and resource usage of DNN models (*e.g.* in the Internet-of-Things [19]), because the adversary could push the victim outside these limits. For example, against a commercial image classification system, such as Clarifai.com, a slowdown attack might waste valuable computational resources. Against a model partitioning scheme, such as Big-Little [31], it might introduce network latency by forcing excessive transmissions between local and remote models. A *slowdown attack* aims to force the victim to do more work than the adversary, e.g. by amplifying the latency needed to process the sample or by crafting reusable perturbations. The adversary may have to achieve this with incomplete information about the multi-exit architecture targeted, the training data used by the victim or the classification task.

2.2.1 Background

Adversarial Examples and Defenses. Prior work on adversarial examples has shown that DNNs are vulnerable to test-time input perturbations [28, 29, 32, 33, 34]. An adversary who wants to maximize a model's error on specific test-time samples can introduce human-imperceptible perturbations to these samples. Moreover, an adversary can also exploit a *surrogate* model for launching the attack and still hurt an unknown victim [35, 36, 37]. This *transferability* leads to adversarial examples in more practical black-box scenarios. Although many defenses [38, 39,

40, 41, 42] have been proposed against this threat, *adversarial training* (AT) has become the frontrunner [34]. In the following sections, we evaluate the vulnerability of multi-exit DNNs to adversarial slowdowns in white-box (Section 2.2.6) and black-box scenarios (Section 2.2.7). In Section 2.2.8, we show that standard AT and its simple adaptation to our perturbations are not sufficient for preventing slowdown attacks.

Efficient Input-Adaptive Inference. Recent input-adaptive DNN architectures have brought two seemingly distant goals closer: achieving both high predictive quality and computational efficiency. There are two types of input-adaptive DNNs: adaptive neural networks (AdNNs) and multi-exit architectures. During the inference, AdNNs [43, 44] dynamically skip a certain part of the model to reduce the number of computations. This mechanism can be used only for ResNet-based architectures as they facilitate skipping within a network. On the other hand, multi-exit architectures [13, 12, 14] introduce multiple side branches—or early-exits—to a model. During the inference on an input sample, these models can preemptively stop the computation altogether once the stopping criteria are met at one of the branches. Kaya et al. [14] have also identified that standard, non-adaptive DNNs are susceptible to *overthinking*, *i.e.*, their inability to stop computation leads to inefficient inferences on many inputs.

Hague et al. [45] presented attacks specifically designed for reducing the energy-efficiency of AdNNs by using adversarial input perturbations. However, our work studies a new threat model that an adversary causes slowdowns on multi-exit architectures. By imperceptibly perturbing the inputs, our attacker can (i) introduce network latency to an infrastructure that utilizes multi-exit architectures and (ii) waste the victim's computational resources. To quantify this vulnerability, we will rely on the efficacy metric based on the early-exit capability (EEC) curves (defined in the

previous section).

Model Partitioning. Model partitioning has been proposed to bring DNNs to resource-constrained devices [31, 19]. These schemes split a multi-exit model into sequential components and deploy them in separate endpoints, e.g., a small, local on-device part and a large, cloud-based part. For bringing DNNs to the Internet of Things (IoT), partitioning is instrumental as it reduces the transmissions between endpoints, a major bottleneck. In Section 2.2.6, on a partitioning scenario, we show that our attack can force excessive transmissions.

2.2.2 Attacking Multi-Exit Architectures

Technical Details. We follow the same setting and notation as in Section 2.1.2. Here, in addition to a confidence-based early-exit mechanism as the stopping criterion (Section 2.1.6), there are also architectures, such as MSDNets [12], that use a prediction entropy-based criterion. Our attack (see Section 2.2.5) leverages the fact that a uniform $F_i(x)$ has both the highest entropy and the lowest confidence. We experiment with both confidence-based—SDNs—and entropy-based—MSDNets—strategies for generality. These strategies follow the same thresholding method (described in Section 2.1.6) for determining whether the model should take an exit for an input sample. For the confidence-based strategy, a prediction at the i^{th} exit is confident if the prediction probability is higher than T_i. For the entropy-based strategy, a prediction at the i^{th} exit is confident if the prediction entropy is lower than T_i. On a holdout set, we tune the thresholds to maximize a model's efficacy (i.e., average inference cost) while keeping its relative accuracy drop (RAD) over its maximum accuracy within 5% and 15%. We refer to these two settings as RAD$<$5% and RAD$<$15%. Table 2.3 (*first* segment) shows how accuracy and efficacy change in each setting.

The SDNs and MSDNets were designed for different purposes: SDNs are generic and can convert any DNN into a multi-exit model, and MSDNets are custom designed for efficiency. In our experiments, we set $M = N$ for SDNs, *i.e.*, one internal classifier at each block and $M = 6$ for MSDNets. Overall, we evaluate an MSDNet architecture (6 exits) and three SDN architectures, based on VGG-16 [46] (14 exits) and MobileNet [47] (14 exits).

Inference Cost Efficiency Metric. We define the *early-exit capability* (EEC) curve of a multi-exit model to indicate the fraction of the test samples that exit early at a specific fraction of the model's full inference cost. Figure 2.5 shows the EEC curves of our SDNs on Tiny ImageNet, assuming that the computation stops when there is a correct classification at an exit point. For example, VGG-16-based SDN model can correctly classify $\sim$50% of the samples using $\sim$50% of its full cost. Note that this stopping criterion is impractical; in Sec 2.2.2, we will discuss the practical ones.

We define the early-exit efficacy, or *efficacy* in short, to quantify a model's ability of utilizing its exit points. The efficacy of a multi-exit model is the area under its EEC curve, estimated via the trapezoidal rule. An ideal efficacy for a model is close to 1, when most of the input samples the computation stops very early; models that do not use their early exits have 0 efficacy. A model with low efficacy generally exhibits a higher *latency*; in a partitioned model, the low efficacy will cause more input transmissions to the cloud, and the latency is further *amplified* by the network round trips. A multi-exit model's efficacy and accuracy are dictated by its stopping criteria (i.e., how conservative the thresholds T_i are), which we have discussed in the previous section.

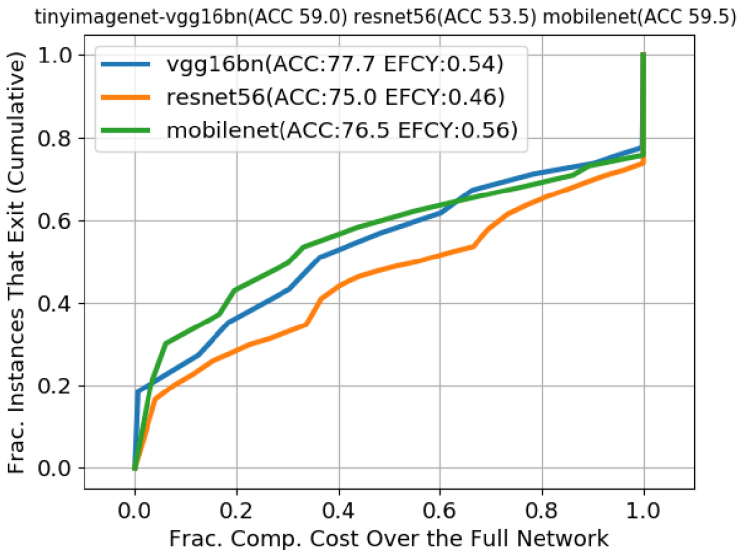

Figure 2.5: The early-exit capability curves to quantify inference efficiency. Each curve shows the fraction of test samples a model classifies using a certain fraction of its full inference cost. 'EFCY' is short for the model's efficacy.

2.2.3 Threat Model

We consider an adversary whose goal is to decrease the early-exit efficacy of a *victim* model. The attacker crafts an *imperceptible* adversarial perturbation, $v \in \mathbb{R}^d$ that, when added to a test-time sample $x \in \mathcal{S}$, prevents the model from taking early-exits.

Adversary's Capabilities. The attacker is able to craft adversarial examples: i.e., modify the victim's test-time samples to apply the perturbations, *e.g.*, by compromising a camera that collects the data for inference. To ensure the imperceptibility, we focus on ℓ_∞ norm bounded perturbations as they (i) are well are studied; (ii) have successful defenses [34]; (iii) have prior extension to multi-exit models [15]; and (iv) are usually the most efficient to craft. In line with the prior work, we bound the perturbations as follows: for CIFAR-10, $||v||_\infty \leq \epsilon = 0.03$ [48], $||v||_1 \leq 8$ [49] and

$||v||_2 \leq 0.35$ [50]; for Tiny ImageNet, $||v||_\infty \leq \epsilon = 0.03$ [51], $||v||_1 \leq 16$ and $||v||_2 \leq 0.6$.

Adversary's Knowledge. To assess the security vulnerability of multi-exit architectures, we study *white-box scenarios*, *i.e.*, the attacker knows all the details of the victim model, including its $\mathcal{D}$ (the training set) and θ (the model parameters). Further, in Section 2.2.7, we study more practical *black-box scenarios*, *i.e.*, the attacker crafts v on a *surrogate* model and applies it to an unknown victim model.

Adversary's Goals. We consider three DeepSloth variants, (i) the *standard*, (ii) the *universal* and (iii) the *class-universal*. The adversary, in (i) crafts a different v for each $x \in \mathcal{S}$; in (ii) crafts a single v for all $x \in \mathcal{S}$; in (iii) crafts a single v for a target class $i \in M$. Further, although the adversary does not explicitly target it; we observe that DeepSloth usually hurts the accuracy. By modifying the objective function we describe in Sec 2.2.5, we also experiment with DeepSloth variants that can explicitly preserve or hurt the accuracy, in addition to causing slowdowns.

2.2.4 Standard Adversarial Attacks Do Not Cause Delays

To motivate DeepSloth, we first evaluate whether previous adversarial attacks have any effect on the efficacy of multi-exit models. These attacks add imperceptible perturbations to a victim's test-time samples to force misclassifications. We experiment with the standard PGD attack [48]; PGD-avg and PGD-max variants against multi-exit models [15] and the Universal Adversarial Perturbation (UAP) attack that crafts a single perturbation for all test samples [52]. Table 2.2 summarizes our findings that these attacks, although they hurt the accuracy, fail to cause any meaningful decrease in efficacy. In many cases, we observe that the attacks actually increase the efficacy. These experiments help us to identify the critical elements of the objective function of an

attack that decreases the efficacy.

NETWORK	NO ATTACK	PGD-20	PGD-20 (AVG.)	PGD-20 (MAX.)	UAP
VGG-16	0.77 / 89%	0.79 / 29%	0.85 / 10%	0.81 / 27%	0.71 / 68%
RESNET-56	0.52 / 87%	0.55 / 12%	0.82 / 1%	0.70 / 6%	0.55 / 44%
MOBILENET	0.83 / 87%	0.85 / 14%	0.93 / 3%	0.89 / 12%	0.77 / 60%

Table 2.2: Impact of existing evasion attacks on efficacy. Each entry shows a model's efficacy (*left*) and accuracy (*right*) when subjected to the respective attack. The multi-exit models are trained on CIFAR-10 and use RAD<5% as their early-exit strategy.

2.2.5 The DeepSloth Attack

As we have seen in the previous section, adversarial examples crafted by prior attacks, such as PGD [48], fail to bypass the victim model's early exits. These attacks cannot be used to study the robustness of multi-exit architectures against adversarial slowdowns and the threat model we aim to extend by considering this emerging goal. To conduct this study, we adapt prior attacks to the goal of slowdown by modifying their objective functions. We call the resulting attack *DeepSloth*, which we describe and evaluate in the following sections.

The Layer-Wise Objective Function. Figure 2.6 shows that the attacks that only optimize for the final output, *e.g.*, PGD or UAP, do not perturb the model's earlier layer representations. This does not bypass the early-exits, which makes these attacks ineffective for decreasing the efficacy. Therefore, we modify the objective functions of adversarial example-crafting algorithms to incorporate the outputs of all $F_i | i < M$. For crafting ℓ_∞, ℓ_2 and ℓ_1-bounded perturbations, we

adapt the PGD [48], the DDN [53] and the SLIDE algorithms [49], respectively. Next, we describe how we modify the PGD algorithm—we modify the others similarly:

$$v^{t+1} = \prod_{||v||_\infty < \epsilon} \left(v^t + \alpha \text{ sgn} \left(\nabla_v \sum_{x \in D'} \sum_{0<i<M} \mathcal{L} \left(F_i \left(x + v \right), \bar{y} \right) \right) \right)$$

Here, t is the current attack iteration; α is the step size; Π is the projection operator that enforces $||v||_\infty < \epsilon$ and $\mathcal{L}$ is the cross-entropy loss function. The selection of $\mathcal{D}'$ determines the type of the attack. For the standard variant: $\mathcal{D}' = \{x\}$, *i.e.*, a single test-time sample. For the universal variant: $\mathcal{D}' = \mathcal{D}$, *i.e.*, the whole training set. For the class-universal variant against the target class $i \in \{1, ..., \mathcal{K}\}$: $\mathcal{D}' = \{(x, y) \in \mathcal{D} | y = i\}$, *i.e.*, the training set samples from the i^{th} class. Finally, $\bar{y}$ is the target label distribution our objective pushes $F_i(x)$ towards. Next, we explain how we select $\bar{y}$.

Pushing $F_i(x)$ Towards a Uniform Distribution. Despite including all F_i, attacks such as PGD-avg and PGD-max [15] still fail to decrease efficacy. How these attacks select $\bar{y}$ reflects their goal of causing misclassifications and, therefore, they trigger errors in early-exits, *i.e.*, $\text{argmax}_j \, F_i^j(x) = \bar{y} \neq y$. However, as the early-exits still have high confidence, or low entropy, the model still stops its computation early. We select $\bar{y}$ as a *uniform distribution* over the class labels, *i.e.*, $\bar{y}^{(i)} = 1/m$. This ensures that $(x + v)$ bypasses common stopping criteria as a uniform $F_i(x)$ has both the lowest confidence and the highest entropy.

2.2.6 Evaluating DeepSloth in the White-Box Setting

Here, we present the results for ℓ_∞ DeepSloth against three SDNs—VGG-16, ResNet-56 and MobileNet-based—and against the MSDNets. Overall, we observe that ℓ_∞-bounded perturbations

are more effective for slowdowns. The optimization challenges might explain this, as ℓ_1 and ℓ_2 attacks are usually harder to optimize [33, 49]. Unlike objectives for misclassifications, the objective for slowdowns involves multiple loss terms and optimizes over all the output logits.

Perturbations Eliminate Early-Exits. Table 2.3 (*second* segment) shows that the victim models have ~ 0 efficacy on the samples perturbed by DeepSloth. Across the board, the attack makes the early exits completely ineffective and forces the victim models to forward all input samples till the end. Further, DeepSloth also drops the victim's accuracy by 75–99%, comparable to the PGD attack. These results give an answer to our main research question: *the multi-exit mechanisms are vulnerable and their benefits can be maliciously offset by adversarial input perturbations*. In particular, as SDN modification mitigates overthinking in standard, non-adaptive DNNs [14], DeepSloth also leads SDN-based models to overthink on almost all samples by forcing extra computations.

Note that crafting a single perturbation requires multiple back-propagations through the model and more floating-point operations (*FLOPs*) than the forward pass. The high cost of crafting, relative to the computational damage to the victim, might make this vulnerability unattractive for the adversary. In the next sections, we highlight scenarios where this vulnerability might lead to practical exploitation. First, we show that in an IoT-like scenario, the input transmission is a major bottleneck and DeepSloth can exploit it. Second, we evaluate universal DeepSloth attacks that enable the adversary to craft the perturbation only once and reuse it on multiple inputs.

NETWORK	MSDNET		VGG16		MOBILENET	
SET.	RAD<5%	RAD<15%	RAD<5%	RAD<15%	RAD<5%	RAD<15%
			BASELINE (NO ATTACK)			
C10	0.89 / 85%	0.89 / 85%	0.77 / 88%	0.89 / 79%	0.83 / 87%	0.92 / 79%
TI	0.64 / 55%	0.83 / 50%	0.39 / 57%	0.51 / 52%	0.42 / 57%	0.59 / 51%
			DEEPSLOTH			
C10	0.06 / 17%	0.06 / 17%	0.01 / 13%	0.04 / 16%	0.01 / 12%	0.06 / 16%
TI	0.06 / 7%	0.06 / 7%	0.00 / 2%	0.01 / 2%	0.02 / 6%	0.04 / 6%
			UNIVERSAL DEEPSLOTH			
C10	0.85 / 65%	0.85 / 65%	0.62 / 65%	0.86 / 60%	0.73 / 61%	0.90 / 59%
TI	0.58 / 46%	0.81 / 41%	0.31 / 47%	0.44 / 44%	0.33 / 47%	0.51 / 43%
			CLASS-UNIVERSAL DEEPSLOTH			
C10	0.82 / 32%	0.82 / 32%	0.47 / 35%	0.78 / 33%	0.60 / 30%	0.85 / 27%
TI	0.41 / 21%	0.71 / 17%	0.20 / 28%	0.33 / 27%	0.21 / 27%	0.38 / 25%

Table 2.3: The effectiveness of ℓ_∞ DeepSloth in the white-box setting.'RAD<5,15%' columns list the results in each early-exit setting. Each entry includes the model's efficacy (*left*) and accuracy (*right*). The class-universal attack's results are an average of 10 classes. 'TI': Tiny ImageNet and 'C10': CIFAR-10.

Attacking an IoT Scenario. Many IoT scenarios, e.g., health monitoring for elderly [54], require collecting data from edge devices and making low-latency inferences on this data. However,

complex deep learning models are impractical for low-power edge devices, such as an Arduino, that are common in the IoT scenarios [55]. For example, on standard hardware, an average inference takes the MSDNet model on Tiny ImageNet 35M FLOPs and ∼10ms.

A potential solution is sending the inputs from the edge to a cloud model, which then returns the prediction. Even in our optimistic estimate with a nearby AWS EC2 instance, this back-and-forth introduces ∼11ms latency per inference. Model partitioning alleviates this bottleneck by splitting a multi-exit model into two; deploying the small first part at the edge and the large second part at the cloud [31]. The edge part sends an input only when its prediction does not meet the stopping criteria. For example, the first early-exit of MSDNets sends only 5% and 67% of all test samples, on CIFAR-10 and Tiny ImageNet, respectively. This leads to a lower average latency per inference, i.e., from 11ms down to 0.5ms and 7.4ms, respectively.

The adversary we study uses DeepSloth perturbations to force the edge part to send all the input samples to the cloud. For the victim, we deploy MSDNet models that we split into two parts at their first exit point. Targeting the first part with DeepSloth forces it to send 96% and 99.97% of all test samples to the second part. This increases average inference latency to ∼11ms and invalidates the benefits of model partitioning. In this scenario, perturbing each sample takes ∼2ms on a Tesla V-100 GPU, i.e., the time adversary spends is amplified by 1.5-5× as the victim's latency increase.

Reusable Universal Perturbations. The universal attacks, although limited, are practical as the adversary can reuse the same perturbation indefinitely to cause minor slowdowns. Table 2.3 (*third* segment) shows that they decrease the efficacy by 3–21% and the accuracy by 15–25%, over the baselines. Having a less conservative early-exit strategy, e.g., RAD$<$15%, increases the resilience

to the attack at the cost of accuracy. Further, MSDNets are fairly resilient with only 3–9% efficacy drop; whereas SDNs are more vulnerable with 12–21% drop. The attack is also slightly more effective on the more complex task, Tiny ImageNet, as the early-exits become easier to bypass. Using random noise as a baseline, i.e., $v \sim U^d(-\epsilon, \epsilon)$, we find that at most it decreases the efficacy by $\sim$3%.

In the universal attack, we observe a phenomenon: it pushes the samples towards a small subset of all classes. For example, $\sim$17% of the perturbed samples are classified into the 'bird' class of CIFAR-10; up from $\sim$10% for the clean samples. Considering certain classes are distant in the feature space, e.g., 'truck' and 'bird'; we expect the class-universal variant to be more effective. The results in Table 2.3 (*fourth* segment) confirm our intuition. We see that this attack decreases the baseline efficacy by 8–50% and the accuracy by 50–65%. We report the average results across multiple classes; however, we observe that certain classes are slightly more vulnerable to this attack.

Preserving or Hurting the Accuracy with DeepSloth. Here, we aim to answer whether DeepSloth can be applied when the adversary explicitly aims to cause or prevent misclassifications, while still causing slowdowns. Our main threat model has no explicit goal regarding misclassifications that hurt the user of the model, i.e., who consumes the output of the model. Whereas, slowdowns additionally hurt the executor or the owner of the model through the computations and latency increased at the cloud providers. In some ML-in-the-cloud scenarios, where these two are different actors, the adversary might aim to target only the executor or both the executor and the user. To this end, we modify our objective function to push $F_i(x)$ towards a slightly non-uniform distribution, favoring either the ground truth label for preventing misclassifications or a wrong

label for causing them. We test this idea on our VGG-16-based SDN model on CIFAR-10 in RAD<5% setting. We see that DeepSloth for preserving the accuracy leads to 81% accuracy with 0.02 efficacy and DeepSloth for hurting the accuracy leads to 4% accuracy with 0.01 efficacy—the original DeepSloth led to 13% accuracy with 0.01 efficacy. These results show the flexibility of DeepSloth and how it could be modified depending on the attacker's goals.

Feature Visualization of DeepSloth. In Figure 2.6, to shed light on how DeepSloth differs from prior attacks, e.g., PGD and PGD-avg, we visualize a model's hidden block (layer) features on the original and perturbed test-time samples. We observe that in an earlier block (*left* panel), DeepSloth seems to disrupt the original features slightly more than the PGD attacks. Leaving earlier representations intact prevents PGDs from bypassing the early-exits. The behaviors of the attacks diverge in the middle blocks (*middle* panel). Here, DeepSloth features remain closer to the original features than prior attacks. The significant disruption of prior attacks leads to high-confidence misclassifications and fails to bypass early-exits. In the later block (*right* panel), we see that the divergent behavior persists.

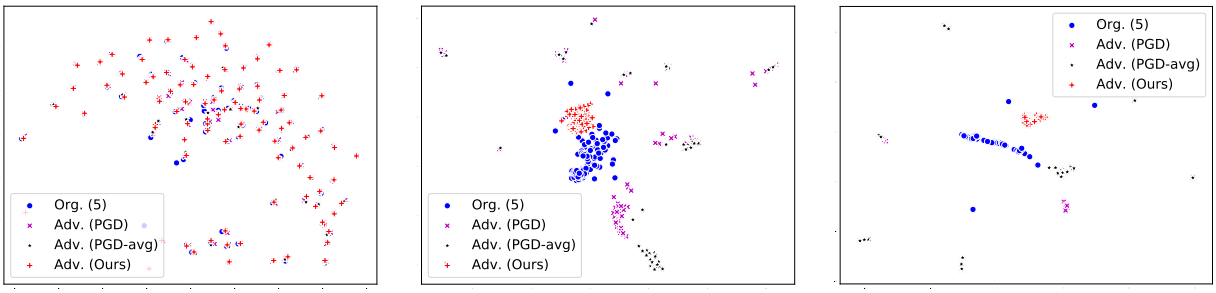

Figure 2.6: Visualising internal layer features against adversarial attacks using UMAP. VGG-16's 3rd (*left*), 8th (*middle*), and 14th (*right*) hidden block features on CIFAR-10's 'dog' class (Best viewed in color, zoomed in).

Hyperparameter Tuning. We find that ℓ_∞-based DeepSloth does *not* require careful tuning. For the standard attack, we set the total number of iterations to 30 and the step size to $\alpha = 0.002$. For the modified attacks for hurting or preserving the accuracy, we set the total number of iterations to 75 and the step size to $\alpha = 0.001$. We compute the standard perturbations using the entire 10k test-set samples in CIFAR-10 and Tiny Imagenet. For the universal variants, we set the total number of iterations to 12 and reduce the initial step size of $\alpha = 0.005$ by a factor of 10 every 4 iterations. To compute a universal perturbation, we use randomly chosen 250 (CIFAR-10) and 200 (Tiny Imagenet) training samples.

Computational Cost of Crafting DeepSloth Samples.

ATTACKS	TIME (SEC.)
PGD-20	38
PGD-20 (AVG.)	48
PGD-20 (MAX.)	475
DEEPSLOTH	44
UNIVERSAL DEEPSLOTH	2

Table 2.4: Time it takes to craft adversarial examples with DeepSloth.

In Table 2.4, we compare the cost of DeepSloth with other attack algorithms on a VGG16-based CIFAR-10 model—executed on a single Nvidia Tesla-V100 GPU. For the universal DeepSloth, we measure the execution time for crafting a perturbation using one batch (250 samples) of the training set. For the other attacks, we measure the time for perturbing the whole test set of

CIFAR-10. Our DeepSloth takes roughly the same time as the PGD and PGD-avg attacks and significantly less time than the PGD-max attack. Our universal DeepSloth takes only 2 seconds (10x faster than DeepSloth) as it only uses 250 samples.

2.2.7 Evaluating DeepSloth in the Black-Box Setting

Transferability of adversarial examples implies that they can still hurt a model that they were not crafted on [56, 57]. Even though white-box attacks are important to expose the vulnerability, black-box attacks, by requiring fewer assumptions, are more practical. Here, on four distinct scenarios, we investigate whether DeepSloth is transferable. Based on the scenario's constraints, we (i) train a *surrogate* model; (ii) craft the DeepSloth samples on it; and (iii) use these samples on the victim. We run these experiments on CIFAR-10 in the RAD$<$5%.

Cross-Architecture. First, we relax the assumption that the attacker knows the victim architecture. We evaluate the transferability between a VGG-16-based SDN and an MSDNet—all trained using the same $\mathcal{D}$. We find that the samples crafted on the MSDNet can slow down the SDN: reducing its efficacy to 0.63 (from 0.77) and accuracy to 78% (from 88%). Interestingly, the opposite seems not to be the case: on the samples crafted against the SDN, the MSDNet still has 0.87 efficacy (from 0.89) and 73% accuracy (from 85%). This hints that DeepSloth transfers if the adversary uses an effective multi-exit model as the surrogate.

Limited Training Set Knowledge. Second, we relax the assumption that the attacker knows the victim's training set, $\mathcal{D}$. Here, the attacker only knows a random portion of $\mathcal{D}$, i.e., 10%, 25%, and 50%. We use VGG-16 architecture for both the surrogate and victim models. In the 10%, 25%, and 50% settings, respectively, the attacks reduce the victim's efficacy to 0.66, 0.5, 0.45, and 0.43

(from 0.77); its accuracy to 81%, 73%, 72%, and 74% (from 88%). Overall, the more limited the adversary's $\mathcal{D}$ is, the less generalization ability the surrogate has and the less transferable the attacks are.

Cross-Domain. Third, we relax the assumption that the attacker exactly knows the victim's task. Here, the attacker uses $\mathcal{D}_f$ to train the surrogate, different from the victim's $\mathcal{D}$ altogether. We use a VGG-16 on CIFAR-100 as the surrogate and attack a VGG-16-based victim model on CIFAR-10. This transfer attack reduces the victim's efficacy to 0.63 (from 0.77) and its accuracy to 83% (from 88%). We see that the cross-domain attack might be more effective than the limited $\mathcal{D}$ scenarios. This makes DeepSloth particularly dangerous as the attacker, without knowing the victim's $\mathcal{D}$, can collect a similar dataset and still slow down the victim. We hypothesize the transferability of earlier layer features in CNNs [58] enables the perturbations attack to transfer from one domain to another, as long as they are similar enough.

Cross-Mechnanism. Finally, we test the scenario where the victim uses a completely different mechanism than a multi-exit architecture to implement input adaptiveness, i.e., SkipNet [43]. A SkipNet, a modified residual network, selectively skips convolutional blocks based on the activations of the previous layer and, therefore, does not include any internal classifiers. We use a pre-trained SkipNet on CIFAR-10 that reduces the average computation for each input sample by $\sim$50% over an equivalent ResNet and achieves $\sim$94% accuracy. We then feed DeepSloth samples crafted on an MSDNet to this SkipNet, which reduces its average computational saving to $\sim$32% (36% less effective) and its accuracy to 37%. This result suggests that the two different mechanisms have more in common than previously known and might share the vulnerability. We believe that understanding the underlying mechanisms through which adaptive models save

computation is an important research question for future work.

2.2.8 Evaluating the Adversarial Training Defense Against DeepSloth

In this section, we examine whether a defender can adapt a standard countermeasure against adversarial perturbations, adversarial training (AT) [34], to mitigate our attack. AT decreases a model's sensitivity to perturbations that significantly change the model's outputs. While this scheme is effective against adversarial examples that aim to trigger misclassifications; it is unclear whether using our DeepSloth samples for AT can also robustify a multi-exit model against slowdown attacks.

To evaluate, we train our multi-exit models as follows. We first take a *base* network—VGG-16—and train it on CIFAR-10 on PGD-10 adversarial examples. We then convert the resulting model into a multi-exit architecture, using the modification from [14]. During this conversion, we adversarially train individual exit points using PGD-10, PGD-10 (avg.), PGD-10 (max.), and DeepSloth; similar to [15]. Finally, we measure the efficacy and accuracy of the trained models against PGD-20, PGD-20 (avg.), PGD-20 (max.), and DeepSloth, on CIFAR-10's test-set.

ADV. TRAINING	NO ATTACK	PGD-20	PGD-20 (AVG.)	PGD-20 (MAX.)	DEEPSLOTH
UNDEFENDED	0.77 / 89%	0.79 / 29%	0.85 / 10%	0.81 / 27%	**0.01** / 13%
PGD-10	0.61 / 72%	0.55 / 38%	0.64 / 23%	0.58 / 29%	**0.33** / 70%
PGD-10 (AVG.)	0.53 / 72%	0.47 / 36%	0.47 / 35%	0.47 / 35%	**0.32** / 70%
PGD-10 (MAX.)	0.57 / 72%	0.51 / 37%	0.54 / 30%	0.52 / 34%	**0.32** / 70%
OURS	0.74 / 72%	0.71 / 38%	0.82 / 14%	0.77 / 21%	**0.44** / 67%
OURS + PGD-10	0.61 / 73%	0.55 / 38%	0.63 / 23%	0.58 / 28%	**0.33** / 70%

Table 2.5: Evaluating adversarial training against slowdown attacks. Each entry includes the model's efficacy score (*left*) and accuracy (*right*). Results are on CIFAR-10, in the RAD<5% setting.

Our results in Table 2.5 verify that AT provides resilience against all PGD attacks. Besides, AT provides some resilience to our attack: DeepSloth reduces the efficacy to ∼0.32 on robust models vs. 0.01 on the undefended one. However, we identify a trade-off between the robustness and efficiency of multi-exits. Compared to the undefended model, on clean samples, we see that robust models have lower efficacy—0.77 vs. 0.53∼0.61. We observe that the model trained only with our DeepSloth samples (Ours) can recover the efficacy on both the clean and our DeepSloth samples, but this model loses its robustness against PGD attacks. Moreover, when we train a model on both our DeepSloth samples and PGD-10 (Ours + PGD-10), the trained model suffers from low efficacy. Our results imply that a defender may require an out-of-the-box defense, such as flagging the users whose queries bypass the early-exits more often than clean samples for which the multi-exit network was calibrated.

2.2.9 Takeaways and Conclusions

In Section 2, we have discussed an emerging capability in machine learning, i.e., input-adaptive inference (Section 2.1), and a goal, i.e., slowdown attacks (Section 2.2), an adversary can pursue against this capability. An adversary can exploit the slowdown attacks in scenarios such as the following:

- **(Case 1) Attacks on cloud-based IoT applications.** In most cases, cloud-based IoT applications, such as Apple Siri, Google Now, or Microsoft Cortana, run their DNN inferences in the cloud. This cloud-only approach puts all the computational burden on cloud servers and increases the communications between the servers and IoT devices. In consequence, recent work [59, 60, 61] utilizes multi-exit architectures for bringing computationally expensive models, e.g., language models [62, 63], in the cloud to IoT (or mobile) devices. They split a multi-exit model into two partitions and deploy each of them to a server and IoT devices, respectively. Under this scheme, the cloud server only takes care of complex inputs that the shallow partition cannot correctly classify at the edge. As a result, one can reduce the computations in the cloud and decrease communications between the cloud and edge.

  On the other hand, our adversary, by applying human-imperceptible perturbations, can convert simple inputs into complex inputs. These adversarial inputs will bypass early-exits and, as a result, reduce (or even offset) the computational and communication savings provided by prior work.

  Here, a defender may deploy DoS defenses such as firewalls or rate-limiting. In this setting, the attacker may not cause DoS because defenses keep the communications between the server and

IoT devices under a certain-level. Nevertheless, the attacker still increases: (i) the computations at the edge (by making inputs skip early-exits) and (ii) the number of samples that cloud servers process. Recall that a VGG-16 SDN model classifies 90% of clean CIFAR-10 instances correctly at the first exit. If the adversarial examples crafted by the attacker bypass only the first exit, one can easily increase the computations on IoT devices and make them send requests to the cloud.

- **(Case 2) Attacks on real-time DNN inference for resource- and time-constrained scenarios.** Recent work on the real-time systems [64, 65] harnesses multi-exit architectures and model partitioning as a solution to optimize real-time DNN inference for resource- and time-constrained scenarios. [64] showed a real-world prototype of an optimal model partitioning, which is based on a self-driving car video dataset, can improve latency and throughput of partitioned models on the cloud and edge by 6.5–14×, respectively.

  However, the prior work does not consider the danger of slowdown attacks; our threat model has not been discussed before in the literature. Our results suggest that slowdown can be induced adversarially, potentially violating real-time guarantees. For example, our attacker can force partitioned models on the cloud and edge to use maximal computations for inference. Further, the same adversarial examples also require the inference results from the model running on the cloud, which potentially increases the response time of the edge devices by 1.5–5×. Our work showed that multi-exit architectures should be used with caution in real-time systems.

In conclusion, the work we have presented in this section exposes the vulnerability of input-adaptive inference mechanisms against adversarial slowdowns. This vulnerability steps outside the conventional adversarial examples threat model and highlights its limitations. As a vehicle

for exploring this vulnerability systematically, we propose DeepSloth, an attack that introduces imperceptible adversarial perturbations to test-time inputs for offsetting the computational benefits of multi-exit inference mechanisms. We show that a white-box attack, which perturbs each sample individually, eliminates any computational savings these mechanisms provide. We also show that it is possible to craft universal slowdown perturbations, which can be reused, and transferable samples, in a black-box setting. Moreover, adversarial training, a standard countermeasure for adversarial perturbations, is not effective against DeepSloth. Our analysis suggests that slowdown attacks are a realistic, yet under-appreciated, threat against input-adaptive models.

Chapter 3: Realistic Constraints of Adversaries

In the previous chapter, we have described a new adversarial attack and evaluated a standard defense, i.e., adversarial training, which aims to make a model robust to adversarial examples (AEs) altogether. However, another type of defense based on *detection* is also possible and has been explored heavily in prior work. These defenses aim to answer whether a given input is an AE or a natural (clean) example. Assumptions about the statistical differences between natural and AEs are commonplace in many detection techniques. As a best practice, AE detectors are evaluated against *adaptive* attackers who actively perturb their inputs to avoid detection [66]. These attackers have full knowledge of the detector and take active steps to craft AEs that avoid detection. As a result of this *arms race*, adaptive attacks have become an evaluation standard for detectors [67, 68].

Due to the difficulties in designing adaptive attacks, however, recent work suggests that most detectors have an incomplete evaluation and can still be defeated by more motivated adversaries [69]. Such adversaries identify and target important defensive assumptions instead of attacking many components a typical defense combines [69]. This methodology, by tailoring attacks to specific detectors, has been effective in exposing the vulnerabilities.

Our work presented in this chapter identifies that the standard way of constraining adversaries in the conventional threat models, i.e., imperceptible input perturbations for a human observer, fails

to craft AEs that can avoid such detectors and exacerbates the arms race. Inspired by the classical security literature, we have defined a more **realistic constraint** based on crafting statistically indistinguishable AEs. We realize this constraint via a generic adaptive attack against detectors: the *statistical indistinguishability attack* (SIA). SIA optimizes a novel objective to craft AEs that follow the same distribution as the natural inputs with respect to DNN representations. Our objective targets all DNN layers simultaneously as we show that AEs being indistinguishable at one layer might fail to be so at other layers. SIA is formulated around evading distributional detectors that inspect a set of AEs as a whole and is also effective against four individual AE detectors, two dataset shift detectors, and an out-of-distribution sample detector, curated from published works. This suggests that SIA can be a reliable tool for evaluating the security of a range of detectors.

Notation and Terminology. We follow a similar notation as in Chapter 2 with a few key differences. As the work in this chapter does not consider any multi-exit architecture, we instead use $F_i(x)$ to denote the intermediate output after the i^{th} hidden (internal) layer of the model. In other terms, $F_i(x)$ is the i^{th} layer *representations* (of F on the input x). As special cases, we refer to $F_{N-1}(x)$ as the *penultimate*-layer representations (**PLR**) and to $F_N(x)$ as the *logits*. The final output of the model—$F(x)$—is a $\mathcal{K}$-dimensional probability distribution over $\mathcal{K}$ classes, obtained by applying the softmax function over the logits. $F^j(x)$ refers to the predicted probability of x belonging to class j. The prediction of a model is $\text{argmax}_j\, F^j(x) = \hat{y}(x)$, i.e., the most probable class. F is trained on a training set, which consists of multiple input-label pairs drawn from an underlying *natural* data distribution, $\{x, y\} \sim \mathbb{N}$. Training uses back-propagation to adjust the DNN's parameters to minimize its *loss* on the training set using a loss function, e.g., cross-entropy.

Datasets. In this chapter, we conduct our experiments on the CIFAR-10 and Tiny-ImageNet datasets, similar to our works in the previous section. The details are shared in Section 2.1.2.

Architectures and Hyper-parameters. We run most of our experiments in this chapter on a standard ResNet-50 [70] architecture. The model has 16 convolutional blocks, a pooling layer and a fully connected layer that produces the logits. We also evaluate MobileNetV2 [71], with 19 blocks, and VGG-16 [72], with 14 blocks. We follow the standard training data augmentation steps described in Section 2.1.2. The models we experiment with are comparable to the models in the prior work, e.g., ResNet-50 models on CIFAR-10 and Tiny-ImageNet reach 94% and 65% accuracy, respectively.

3.1 Background

3.1.1 The Adversary's Constraints in the Adversarial Examples (AEs) Threat Model

The conventional threat model [2] defines an AE as an input sample x_a that is *very similar* to a natural example (**NE**) $x_n \sim \mathbb{N}$ but $\hat{y}(x_n) \neq \hat{y}(x_a)$. The adversary, while crafting the AE, is constrained to ensure that x_n and x_a are *similar* according to a distance metric $D(\cdot, \cdot)$, usually an ℓ_p-norm distance defined over the input space (e.g., pixel-wise distance for images). Generally, a crafting method, such as PGD [3], optimizes an objective function to iteratively *perturb* x_n into x_a to change the model's prediction while keeping $D(x_n, x_a)$ small.

Unlike the AE threat model, realistic adversaries in security research, who perform attacks such as malicious network traffic [73] or malware [74], have to defeat defenders who deploy

intrusion detection systems (IDS). Essentially, an IDS learns the normal system behaviors when there is no attack and then it recognizes possible attacks by looking for statistical abnormalities. Facing an IDS introduces an additional constraint for the adversaries as their attacks need to evade an IDS in addition to achieving the malicious outcome. To this end, the prior work [6, 5] has developed evasive methods that modify attacks to statistically match a system's normal profile. Although undetectability is a core constraint of a realistic adversary in security, powerful AE crafting methods in ML literature are often only designed for attack success [3, 75, 76]. The work in this chapter aims to fill this gap by developing a general-purpose AE crafting method as an analogue of traditional evasive attacks. Our method (introduced in Section 3.2) crafts AEs while balancing between two objectives: attack success and evading statistical anomaly detectors.

3.1.2 Detecting Adversarial Examples

Two defensive approaches have emerged against AEs: eliminating them by making DNNs robust [3]; or detecting them before they harm the rest of the system [77, 78]. Detection techniques have gained traction as both theory [79] and practice [80] suggest that achieving robustness is challenging. Typical detectors, which we refer to as *individual detectors* (**IDs**), operate on a single input sample to answer whether it is adversarial. Supervised IDs [81, 82] use a set of known AEs to train a binary classifier that learns to discriminate AEs from NEs. Unsupervised IDs [78, 83, 67, 84], rely only on the statistical properties of NEs and detect non-conforming samples as AEs. A common thread in most IDs is extracting informative test statistics derived from DNN layer representations on the input samples [68]. Further, researchers have also proposed *distributional detectors* (**DDs**) [85, 86] that inspect a set of samples as a whole. DDs assume that

a set contains either only AEs or only NEs and make a determination between the two options. Although this assumption allows DDs to extract stronger statistical signals compared to IDs, it also makes DDs less practical [66].

Distributional Detection Methodology. We formulate our realistic adversary's constraint around defeating more powerful DDs. This will allow us to develop an adaptive attack that would be effective against a broad class of detectors (both IDs and DDs). Prior DDs [85, 86] first form the set X_n by drawing natural samples from a holdout set. We consider DDs that construct X_n according to the model's predictions on X_a. This aims to prevent detection solely due to label shift, e.g., the holdout data is balanced and the non-adversarial inputs in X_a come from a single class. For each class y, this ensures $|\{x_i{\in}X_a|\hat{y}(x_i){=}y\}| \approx |\{x_j{\in}X_n|\hat{y}(x_j){=}y\}|$.

Next, a DD applies a two-sample test (**2ST**) between X_n and X_a. The null hypothesis of a 2ST ($\mathfrak{H}_0$) states that the two sets come from the same distribution. A common test statistic is the maximum mean discrepancy (**MMD**) [87]. MMD measures the *closeness* between two distributions in terms of a kernel k that gives point-level *similarities* of its inputs. The MMD between X_n and X_a is computed using the unbiased empirical estimator [87], which assumes $|X_n|{=}|X_a|{=}m$. A permutation test or the wild bootstrap [88] is used to obtain the p-value of the estimate. Ultimately, a DD rejects $\mathfrak{H}_0$ if the p-value is less than a selected significance level, $0{<}\alpha{<}1$. This implies detecting X_a as adversarial and α controls the false positive rate when X_a is natural.

Applying the MMD test in the input-space has been shown to perform poorly for distributions with complex structures [89]. Semantic-aware MMD (SAMMD), the state-of-the-art DD by Gao et al. [86], tackles this problem by applying the MMD test on the representations of a pre-trained

DNN. To tune its kernel, SAMMD splits the available data and learns the kernel parameters that maximize the power of the MMD test on the training split. Using these parameters, it then performs a 2ST on the test split.

As SAMMD specifically focuses on the PLR, we also consider more general *layerwise* DDs that can operate on any DNN layer. These DDs use the following kernel at i^{th} layer: $k_i(x, z) = exp(-\frac{1}{2\sigma_i^2}\|F_i(x) - F_i(z)\|^2)$, where x and z are the inputs. Here, σ_i is the *length-scale* of the Gaussian kernel, which, intuitively, controls how close the inputs of the kernel have to be to significantly influence its output. We also use the holdout data to standardize the representations and reduce their dimensionality with average pooling for CNNs. SAMMD uses a splitting strategy to tune its kernel, which reduces the test power as it leaves fewer samples for testing [90]. Therefore, layerwise DDs combine multiple kernels tuned using the median heuristic as a more efficient alternative [90].

Threat Model Against Distributional Detectors. We assume the role of an adversary who starts with X_0, an *initial set* of i.i.d. NEs drawn from $\mathbb{N}$ ($|X_0| = m$). With full access to the victim model, the adversary crafts an AE for each $x \in X_0$. The defender aggregates these AEs into the set X_a and deploys a DD to inspect it. Our goal is to ensure almost all AEs in X_a are successful while still bypassing the defender. We consider an AE as successful if it changes the model's output: when $\hat{y}(x_n) \neq \hat{y}(x_a)$ or when $\hat{y}(x_a) = t$ for a targeted AE with the target class t.

Attack and Detection Success Metrics. The attack success rate (**ASR**) measures the percentage of successful AEs among the crafted samples. The performance of a statistical test depends on the number of samples available to it, i.e., $|X_a| = |X_n| = m$. To this end, we repeat the test 100 times, each time by randomly selecting m samples from the crafted AEs and m samples from the

NEs in the holdout set. Denoted by $\mathbf{DR_m}$, we then report the detection rate as the percentage of tests with sample size m that resulted in a p-value less than $\alpha = 0.05$.

3.2 Statistical Indistinguishability Attack (SIA)

This section focuses on a case study to motivate our work and to formulate our attack methodology. We experiment with a ResNet-50 model on CIFAR-10 (the details are given at the beginning of this chapter), start from 2000 test samples as our initial set X_0, and craft targeted AEs with random targets. We use UMAP [91] to visually compare the representations of AEs and NEs in the holdout set. For simplicity, we visualize only the samples classified into the *horse* (randomly selected) class.

3.2.1 Standard Attacks Are Detectable

In this section, we show how conventional attacks such as PGD [3] lead to distinct representation distributions. We experiment with three conventional attack algorithms that constrain the perturbations with different input-space distance metrics: PGD [3] uses ℓ_∞, DDN [92] uses ℓ_2 and SLIDE [93] uses ℓ_1 distances. They all place an upper bound on the perturbations with a hyper-parameter, i.e., $D(x_n, x_a) < \varepsilon$. For each attack, we find the minimum ε value to reach $\sim$100% ASR.

In Figure 3.1, we visualize the model's penultimate-layer representations (PLR) on the AEs crafted by these attacks. We visually demonstrate that the distributions of these AEs are distinct from the NEs. Standard attacks optimize only for changing the model's prediction as a goal and disregard the natural distribution [69]. As a result, SAMMD obtains perfect detection rates (100%)

when it has access to 30, 40, and 50 AEs against PGD, DDN, and SLIDE, respectively.

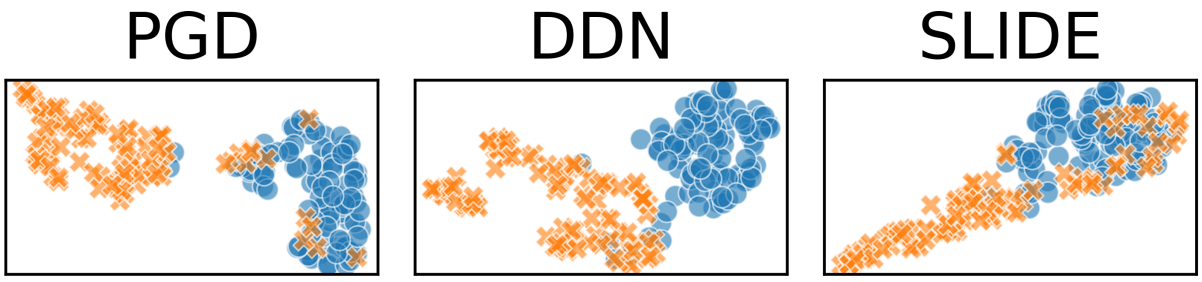

Figure 3.1: The penultimate-layer representations of the natural (✖) and the adversarial (●) examples crafted by three standard attacks.

3.2.2 Formulating SIA

We have shown that optimizing only for changing the model's predictions and constraining AEs only for input-space imperceptibility with distance functions such as ℓ_∞ lead to detectable attacks. Ideally, an attack against a DD crafts a set of AEs that have *statistically indistinguishable* representations with respect to NEs.

Towards this goal, we first note that the MMD between two sets of samples with respect to the i^{th} layer representations—$\text{MMD}_i(X_n, X_a)$—is end-to-end differentiable. Previously, this has allowed MMD to act as a loss function to train generative models by minimizing the distance between the generated and the natural distributions [94]. Inspired by this, we design the following objective function for SIA that finds the set of perturbations Δ, where $X_a = \{x_j + \Delta_j \mid x_j \in X_0,\ \ \Delta_j \in \Delta,\ \ j = 1 \cdots m\}$ which we shortly denote as $X_0 + \Delta$.

$$\min_{\Delta} \gamma \text{MMD}_i(X_g, X_0 + \Delta) + \sum_{j=1}^{|X_0|} \mathcal{L}(\hat{y}(x_j + \Delta_j), T_j)$$

The first term keeps the AE and NE distributions close with respect to the i^{th} layer repre-

sentations. The attacker samples a random set of NEs to form X_g and uses it during the MMD computations as a *guide set*. To prevent overfitting, we sample three guide sets and take the average MMD loss on them. The attack ultimately aims to defeat DDs by making X_a statistically equivalent to the guide sets. We sample the guide sets such that the predicted class labels of the samples in X_g match T, the target labels of the attack. We tune the kernel using the median heuristic, i.e., the median pairwise distance between the points in X_g and $X_0 + \Delta$.

The second loss term ensures that adding the corresponding perturbation, $\Delta_j \in \Delta$, to each initial sample, $x_j \in X_0$, changes the model's prediction into the target class, $T_j \in T$. We use cross-entropy as the loss function $\mathcal{L}$ in the second term. In this section, we perform targeted attacks by selecting $T_j \neq \hat{y}(x_j)$ randomly from the list of available classes.

The hyper-parameter γ balances between the two terms to reflect the attacker's priorities ($\gamma \geq 0$). As γ decreases the second term will start dominating at the risk of increasing the detectability; whereas as γ increases the ASR of the AEs might start dropping.

For its simplicity, we use the ℓ_∞-norm bounded PGD attack ($\|\Delta\|_\infty \leq \varepsilon$) to minimize our objective. Unless specified otherwise, we set $\varepsilon = 0.03$, following prior work [3]. We perform 200 PGD iterations, use a scheduler that periodically reduces the step size, and craft 250 AEs at once as a batch. Note that SIA objective can be integrated into any other distance metric and gradient-based attack method.

Applying SIA Against the PLR. We apply SIA against the PLR (17th layer) and visualize the resulting representations in Figure 3.2. We search $\gamma \in [0, 128]$ to find the value that minimizes SAMMD's DR while still achieving $\sim$100% ASR. Compared to Figure 3.1, we first observe the closeness between AE and NE distributions at the PLR. As a result, SAMMD fails to detect these

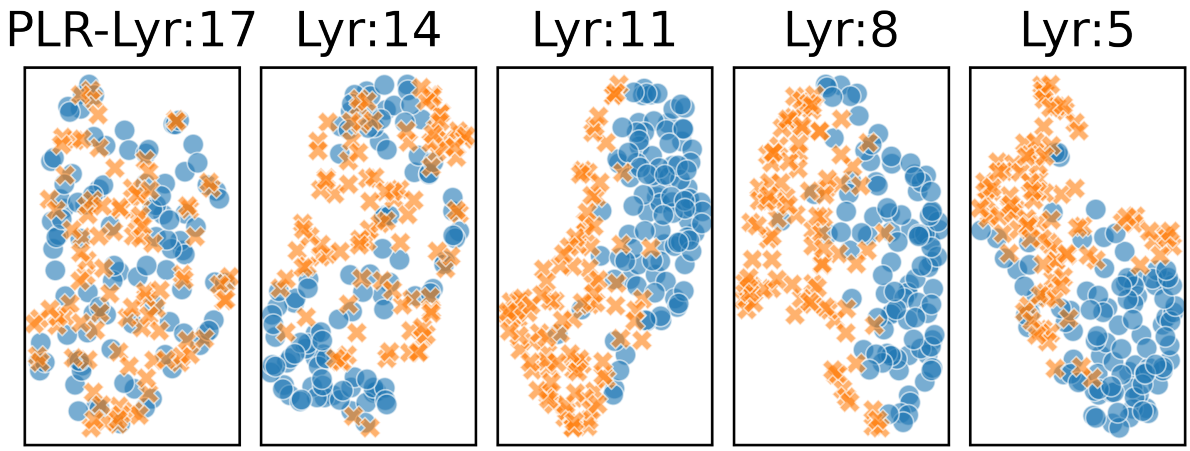

Figure 3.2: The representations of adversarial examples crafted by SIA against the PLR.

AEs with 100 samples, i.e., DR_{100} = 1%; whereas it could detect the standard attacks perfectly with only 50 samples. Note that SAMMD focuses only on the PLR and consumes a total of $2m$ AEs to obtain DR_m as it uses half of them for training its MMD kernel. Therefore, we also focus on the layerwise DDs (Section 3.1.2), which tune the kernels heuristically and can operate on any layer.

Turning our attention to the other layers reveals that the AEs are still detectable. Layerwise DDs obtain between 80% and 100% DR_{100} up until the 14th layer, after which the DR drops rapidly. Although applying SIA against the PLR breaks SAMMD (the most recent DD), it fails to craft statistically indistinguishable AEs at the remaining layers.

SIA Against One Layer Is Still Detectable at Others. Here, we perform SIA against the i^{th} layer and measure the DR of a layerwise DD applied at the j^{th} layer. Figure 3.3 presents the results for a uniformly spaced subset of layers. For example, attacking the 5th layer drops the DR at this layer to 3% whereas the same AEs are still detectable at the 14th layer with 100% DR. Note how attacking certain layers also defeats the detection at some other layers, e.g., 8th and 11th layers.

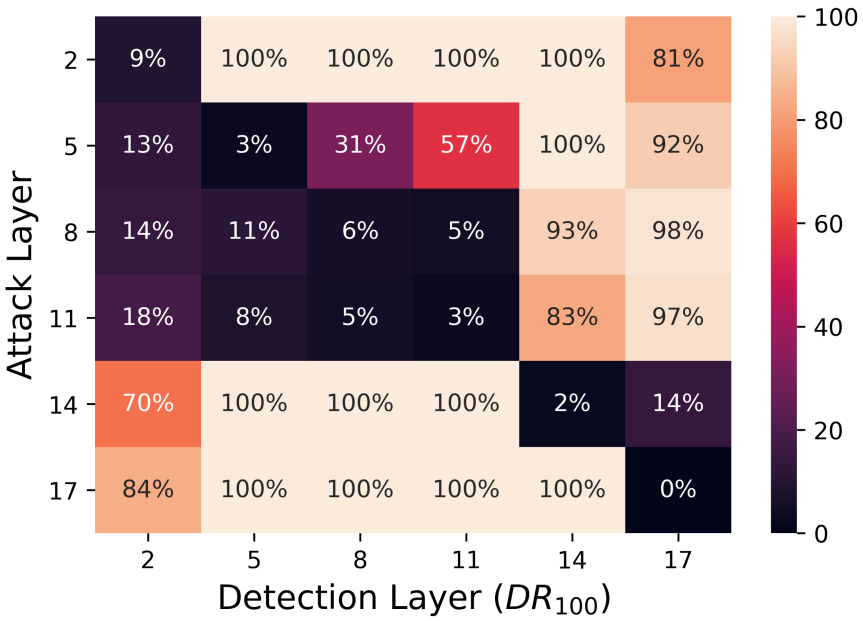

Figure 3.3: Crafting adversarial examples with SIA against one layer and measuring the detection rate at the other layers.

We hypothesize that this pattern arises from the architecture and the feature diversity among the layers [95]. The layers in the same group, e.g., 8th and 11th, have the same feature map scale and, therefore, extract similar features. Prior representation similarity metrics [96] also report similar findings. Ultimately, SIA perturbations against one layer create outlier representations at a *dissimilar* layer. Overall, these results lead us to conclude that we have to attack multiple layers jointly to craft indistinguishable AEs across the model.

Attacking All layers. Our previous formulation only attacks a single layer, which has failed to evade detection at others. To jointly attack all layers, we update the first term of SIA objective as $\sum_{i=1}^{N} \gamma_i \text{MMD}_i(X_g, X_0 + \Delta)$, where N is the number of layers. This minimizes the weighted sum of the MMD loss from all layers, with γ_i as the weight of the i^{th} layer. We opt for attacking all layers for simplicity as attacking only the dissimilar layers has not led to any improvements.

We experimentally find that assigning equal weights to each layer performs well, i.e., $\gamma_i = \gamma$ for $i \in \{1, \cdots, N\}$. To find the ideal γ, we search in the range $[0, 128]$, while keeping the ASR above 95%. This allows us to craft successful AEs that are also indistinguishable.

Figure 3.4 suggests that the resulting AEs with 97% ASR follow the same distribution as the NEs across the model. Layerwise DDs at each layer result in DR_{100} between 0% and 3%, i.e., the AEs are not detectable at any layer. SAMMD also fails to detect the AEs we craft as it obtains less than 1% DR_{100}, averaged over 10 runs.

Regarding the impact of γ, we see a trend that confirms our intuition. When we set $\gamma = 1$, the AEs we craft have 100% ASR, however, SAMMD achieves a non-trivial 20% DR_{300} against these AEs. As the sweet spot, we have presented the results for $\gamma = 32$, where DR_{300} is 6% and the ASR is 97%. On the other extreme, when $\gamma = 128$, the DR_{300} is only 0.3%, however, the ASR drops to 58%, as we expected.

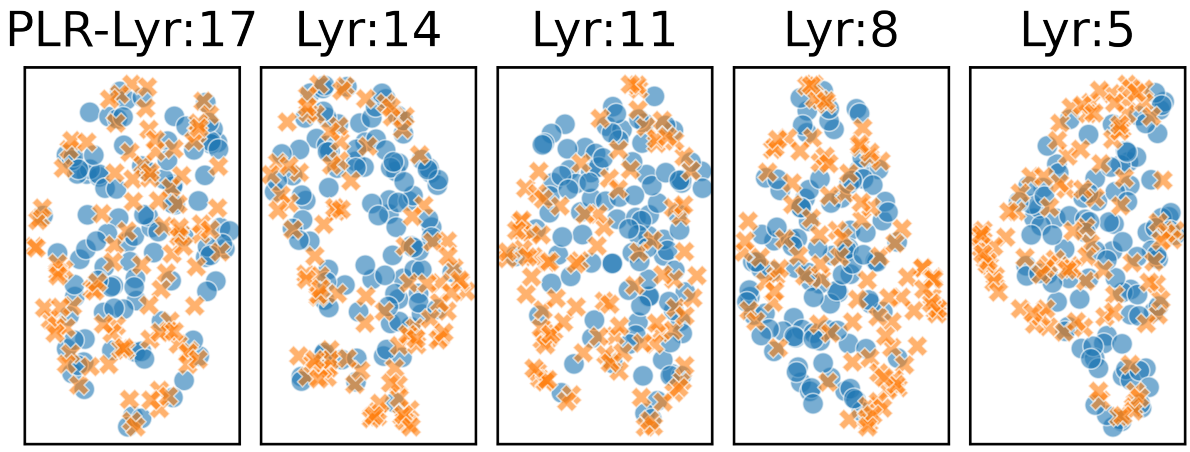

Figure 3.4: The representations of adversarial examples crafted by SIA against all layers.

Runtime Performance. The main overhead SIA has over PGD is computing MMD between the adversarial and guide sets with respect to the model's representations at each layer. Runtime

of MMD computation grows linearly, quadratically, and linearly with the number of dimensions in the representations, the number of samples in the sets and the number of layers in the model, respectively. This results in around 30–40% slower attack iterations compared to PGD. For example, against ResNet-50/CIFAR-10, VGG-16/CIFAR-10, and ResNet-50/TinyImageNet models, crafting 1000 samples on a commodity GPU takes SIA (*200 iterations*) 10, 4, and 30 minutes, respectively; whereas it takes PGD (*50 iterations*) 1.5, 0.5 and 5 minutes.

3.2.3 Comparison With the Feature-Level Attack (FLA)

Similar to our attack, the FLA [97] aims to craft AEs that match the representations of NEs at a layer. For an initial sample x_n, it randomly selects a natural guide sample x_g and crafts the perturbation δ that minimizes $\|F_i(x_n + \delta) - F_i(x_g)\|_2$. The FLA has been an effective attack against typical detectors that operate on individual samples [69]. To evaluate whether it can also defeat DDs, we apply the ℓ_∞-norm bounded FLA with random target labels against each layer. We present the detection results in Table 3.1 for attacking different layers with the FLA. We set the ℓ_∞-norm perturbation-bound to $\varepsilon = 0.03$, same as SIA. For an initial sample, x_n, we select the corresponding guide, x_g, randomly among the holdout samples such that $\hat{y}(x_g) = t$, where t is the attack's target class. We see how the FLA is successful against the deeper layers in evading the DDs; however, it fails in earlier layers. This aligns with Sabour et al.'s [97] findings that it is more challenging to apply the FLA against earlier layers as they are less sensitive to perturbations. Moreover, attacking all layers with the FLA also cannot evade the DDs. These results demonstrate that the FLA is ineffective against DDs regardless of the parameters and layerwise DDs against the FLA result in over 60% DR_{100} at most layers. Overall, although the AEs crafted by the FLA

evade individual detectors, collectively they are from a different distribution than the NEs.

ATTACKED LAYER	ASR	ATTACKED LAYER DR	AVG LYR	MAX LYR
2	98%	40%	89%	100%
5	98%	63%	88%	100%
8	100%	91%	93%	100%
11	100%	100%	94%	100%
14	100%	0%	75%	100%
17	100%	0%	79%	100%
ALL	99%	-	78%	100%

Table 3.1: The detection performance (DR_{100}) of DDs against the FLA. We attack individual layers with the FLA to craft targeted AEs with random targets. We apply the layerwise DDs and measure the detection rates at the attacked layers. We also report the average and the highest DRs among the layerwise DDs. In the last row (ALL), we attack all the layers in the DNN. We experiment with a ResNet-50 model on CIFAR-10.

.

Further, our next experiment suggests that attacking multiple layers simultaneously with the FLA might also be challenging. We first randomly sample NEs from the *ship* and *horse* classes and compute the pairwise ℓ_2 distances between the representations of these samples at different layers. For each ship sample, we then find the Spearman's rank correlation between its distances to the horse samples at layers i and j. Finally, for all ship samples, we take the average of this measurement which results in a metric we denote by $COR_{(i,j)}$. Having $COR_{(i,j)}$ close to one

implies that if two samples are close at layer i, they are close at layer j as well. However, we see that the distances are poorly correlated between certain layers. For example $COR_{(2,5)} = 0.77$, whereas $COR_{(2,14)} = 0.16$ and $COR_{(5,8)} = 0.81$ whereas $COR_{(5,14)} = 0.29$. This experiment produces consistent results for other classes as well. As the FLA selects only a single guide sample for each AE, an AE that is close to the guide at one layer might fail to be close to it at another.

Layerwise Analysis of SIA. Being able to select only a single guide sample for all layers makes it difficult for FLA to avoid detection at all layers. Here, we show that SIA crafts an AE by implicitly selecting different guides at different layers. In Figure 3.5, we present images from a case study we conduct to understand this tendency of SIA. We visualize two clean initial samples from X_0 and the corresponding AEs SIA crafts when it is applied against all layers of the DNN. We then find the holdout images *closest* to the initial clean, and adversarial, images with respect to the representations at different layers. This reveals two behaviors that make SIA perform better than the FLA in evading layerwise DDs.

First, for a clean image, the closest images at different layers are not classified into the same class as the clean image. For example, for the clean *sports car*, the closest image at the 2nd and 7th layers is an image of a *teapot*. These results align with our findings using the COR metric in Section 3.2.3. This behavior demonstrates why selecting a single guide per initial sample makes it challenging for the FLA to evade multiple layers.

Second, for an adversarial image, the closest images at different layers change from layer to layer. For example, for the adversarial *bow tie* (originally a *candle*), we see the closest images are *orange* and *lampshade* and *bow tie* at the 7th, 12th and the 17th layers, respectively. This shows that SIA implicitly selects different guide samples at different layers to craft AEs.

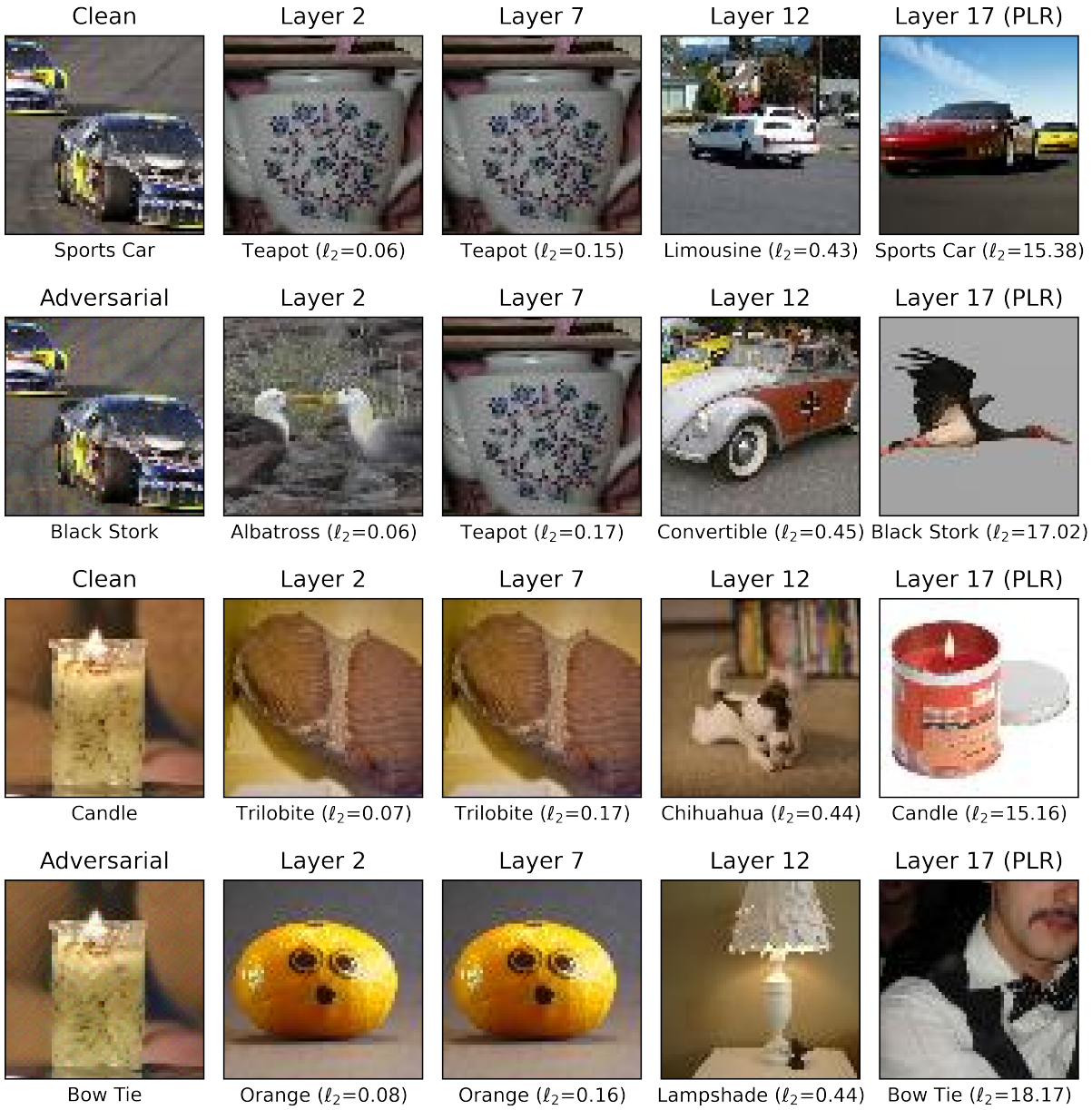

Figure 3.5: Understanding the layerwise behavior of SIA. Displaying the holdout images closest to the initial natural and the corresponding adversarial examples at different layers. We craft the AEs using SIA against all layers of a ResNet-50 model trained on Tiny-ImageNet. At the bottom of each image, we display the model's predicted class on that image. We find the closest images by measuring the ℓ_2 distance between two images with respect to the model's representations.

3.3 Empirical Evaluation of SIA Against Distributional Detectors

Attack Details. We craft 1000 targeted AEs, starting from a random subset of the test samples as our initial set X_0. We select the target labels T randomly from the list of available classes. This selection reflects the average-case difficulty in crafting targeted AEs [98].

Detector Details. Against our attack, we apply the layerwise DDs at each layer and report the average and the highest detection rates (DR) among them. We observe that combining the p-values from each layer using popular methods, such as Fisher's [99] or Brown's [100] does not improve the DR over the best layer. We also evaluate SAMMD [86] as it is the state-of-the-art DD. To understand how consuming more samples affects DDs, we report DR_{100}, DR_{200} and DR_{300}. These detectors have around 5% (false) DR when they operate on NEs instead of AEs.

3.3.1 White-Box Scenarios

In Table 3.2, we present the detection results against SIA in the white-box setting, i.e., the attacker has full access to the victim model. We see how SIA evades the DDs across the board, while still achieving a high ASR. The low DR of layerwise DDs shows that the AEs we craft are not detectable at any layer. The more complex the task gets (e.g., larger inputs or more classes), performing adversarial attacks becomes easier as the models become more sensitive to perturbations [101]. As a result, SIA against Tiny-ImageNet models is even more successful and SAMMD completely fails.

Comparisons With Baseline Attacks. Here, we compare SIA with two popular baseline attacks—PGD and AutoAttack [75]—in terms of detection rates. We have already presented the comparisons

A	ASR	AVG LYR	MAX LYR	SAMMD
		RANDOM CLASS TARGETED		
		CIFAR-10		
		$DR_{100\ /\ 200\ /\ 300}$	$DR_{100\ /\ 200\ /\ 300}$	$DR_{100\ /\ 200\ /\ 300}$
R	97%	1 / 2 / 3 %	3 / 5 / 7 %	0 / 2 / 6 %
V	94%	1 / 1 / 1 %	3 / 4 / 4 %	0 / 5 / 28 %
M	100%	3 / 2 / 3 %	7 / 5 / 9 %	0 / 0 / 1 %
		TINY-IMAGENET		
R	100%	3 / 4 / 4 %	9 / 8 / 9 %	0 / 0 / 0 %
V	100%	2 / 2 / 3 %	6 / 5 / 9 %	0 / 0 / 0 %
M	100%	2 / 4 / 2 %	7 / 8 / 6 %	0 / 0 / 2 %

Table 3.2: The performance of distributional detectors against SIA. Column [A] includes three architectures: **R**esNet-50, **V**GG-16 and **M**obileNet. [AVG LYR] presents the average DR across the layers of the model and [MAX LYR] presents the highest DR among all layers.

with another attack FLA [97] in Section 3.2.3.

We present the detection results against PGD in Figure 3.6 for increasing perturbation-bounds (ε) of PGD. We see how the ASR increases as we increase ε and how, as a result, the AEs become more detectable. Even with low ASR levels when $\varepsilon < 0.01$, the layerwise DDs still have moderate success against PGD.

We present the detection results against AutoAttack [75] in Table 3.3 for increasing perturbation-bounds (ε). We found that maximizing *difference of logits ratio* loss, instead of cross-entropy, allows AutoAttack to be less detectable than PGD in the final layers. However,

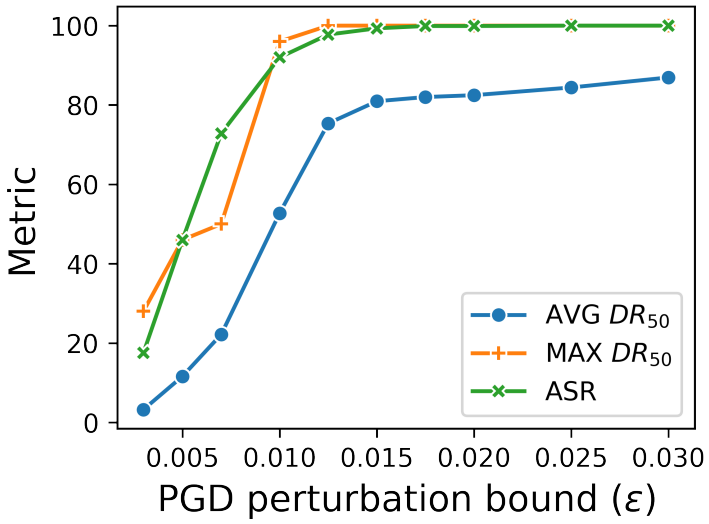

Figure 3.6: Detection performance (DR_{50}) of the layerwise detectors against PGD with increasing perturbation-bounds (ε). AVG and MAX report the average and the highest DR of the layerwise DDs. We experiment with a ResNet-50 model trained on CIFAR-10.

in the remaining layers, AutoAttack is still easily detected: e.g., 100% detection rate at the 11th layer (SIA has only a 2% detection rate). Overall, these results validate our formulation to evade detection at multiple layers.

Applying SIA Against an Adversarially Robust Model. Robustness aims to prevent AEs by making the model less sensitive to perturbations [2]. Although it has been studied separately from detection, recent work has formed a connection between the two approaches [102, 103]. Here, we apply SIA and standard PGD against a ℓ_∞-norm adversarially-trained robust model, provided by Madry et al. [3]. When $\varepsilon = 0.03$, SIA and PGD craft AEs with 42% ASR, on which SAMMD obtains 37% and 100% DR_{300}, respectively. A higher ε increases the ASR without making SIA fully detectable, e.g., when $\varepsilon = 0.07$, ASR is 82% and DR_{300} is 46%. This shows that PGD is still easily detectable even at a low ASR level, whereas SIA can still evade DDs that rely on

DETECTION LAYER					
2	**5**	**8**	**11**	**14**	**17**
$\epsilon = 0.005$, ASR=79%					
4%	10%	35%	36%	10%	4%
$\epsilon = 0.01$, ASR=99%					
5%	92%	100%	100%	63%	10%
$\epsilon = 0.02$, ASR=100%					
6%	100%	100%	100%	100%	64%

Table 3.3: The detection performance (DR_{100}) of distributional detectors against AutoAttack. We apply the attack with different perturbation bounds (ϵ) and measure the detection rate at different layers of the model. We experiment with a ResNet-50 model on CIFAR-10.

.

representations from robust models.

Giving More Samples to the Detectors. A DD performs better as the number of samples available to it increases. Here, we experiment with our ResNet-50 model on CIFAR-10 and quantify how many AEs a DD needs to be able to consistently detect SIA. We present the full detection trend in Figure 3.7. When SAMMD consumes 2000 AEs (1000 for kernel tuning and 1000 for testing), it achieves 78% DR_{1000}, with 4% false DR. The highest DR_{1000} among the layerwise DDs is 27%, with around 8% false detection rate. Note that these DDs have a perfect (100%) DR against the standard attacks with only 50 samples. As a realistic defender might only have a limited number

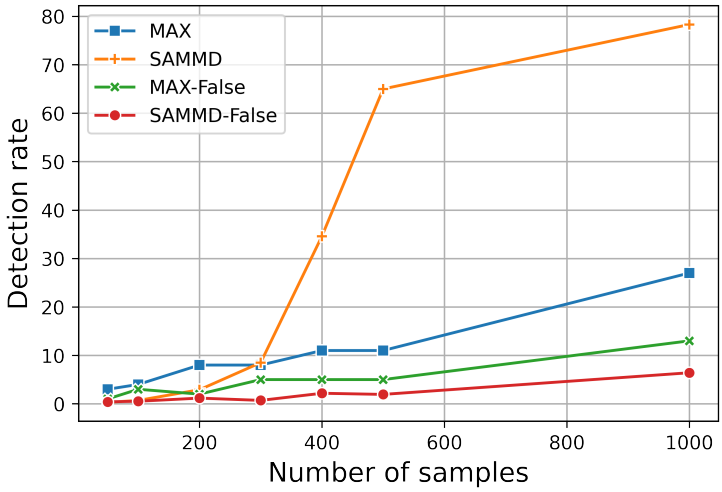

Figure 3.7: The detection performance of distributional detectors as a function of the number of available samples. We experiment with a ResNet-50 model trained on CIFAR-10. We measure the false detection rates by providing NEs to the detectors instead of AEs.

of AEs, this casts doubt on whether detecting SIA is practical.

3.3.2 Transferability of SIA

As we discussed in Section 2.2.7 in the context of DeepSloth, the transferability of AEs implies that they can still hurt a model that they were not crafted on [104, 105]. Even though white-box attacks are important to expose a vulnerability, transfer attacks, by requiring fewer assumptions, are more practical. Here, we investigate whether SIA can craft transferable AEs that also avoid detection at an unknown model. To this end, we follow the same methodology as in Section 2.2.7. We first craft the AEs against a *surrogate* model and transfer them to a *target* model. We then measure (i) the ASR of these AEs against the target model and (ii) the performance of the layerwise DDs that operate on the target model's representations. As a baseline, we also craft AEs via PGD and tune its perturbation-bound ε to achieve a similar ASR to SIA.

S-T	SIA			PGD		
	ASR	**MAX LYR**	**AVG LYR**	**ASR**	**MAX LYR**	**AVG LYR**
		$DR_{50\,/\,100}$	$DR_{50\,/\,100}$		$DR_{50\,/\,100}$	$DR_{50\,/\,100}$
R-R	46%	19 / 84%	4 / 20%	44%	100 / 100%	63 / 82%
R-V	24%	21 / 89%	7 / 42%	23%	99 / 100%	63 / 90%
R-M	41%	41 / 94%	14 / 56%	41%	100 / 100%	70 / 85%
V-M	46%	60 / 99%	13 / 58%	42%	100 / 100%	65 / 82%

Table 3.4: Transferability performance of SIA. [S-T] column lists the architectures of the surrogate [S] and target [T] models.

In Table 3.4, we present the results on crafting AEs with random targets against CIFAR-10 models. We first note that, for similar ASR levels, SIA overall performs better than PGD against the detectors. Moreover, compared to the white-box setting, the attacks are more detectable and have less ASR. For example, when using two ResNet-50 models as both the surrogate and the target, SIA achieves 46% ASR. On the AEs, the DDs achieve at most 84% DR_{100} against SIA; which was only 3% in the white-box setting. Further, we see that AEs are more transferable, for example, from ResNet-50 to MobileNet than from ResNet-50 to VGG-16. The similarities between different DNN architectures explain the transfer success between different models [105]. We believe combining SIA objective with the recent methods that craft more transferable AEs [106, 107] is a promising direction to boost the transferability of SIA.

Cross-Statistic Transferability. So far in this chapter, we have applied DDs that perform a 2ST using MMD as their test statistic. As SIA also uses MMD to craft indistinguishable AEs, we ask whether these AEs would evade a 2ST that uses a different statistic. To this end, we apply a

classifier two-sample test (**C2ST**) by Lopez-Paz et al. [108] on the representations at each layer. A C2ST first constructs a dataset by pairing the samples in X_a and X_n with positive and negative labels, respectively. It then trains a binary classifier on this dataset and uses its accuracy on test samples to compute the p-value, which we use to obtain the DR. We use 300 samples to train the classifier and 300 samples to obtain the p-value. Applied on a CIFAR-10 ResNet-50 model, C2ST achieves at most 6% DR_{300} against the AEs we craft; whereas the MMD-based DD achieves 7%. This demonstrates that SIA is transferable to test statistics other than MMD.

3.3.3 The Factors Affecting the Success of SIA

Essentially, an adversarial attack adds perturbation δ to an initial sample $x \in X_0$, which moves x towards some end *destination* with respect to F_i, the model's representations. To enforce that the crafted AEs follow the natural distribution, our attack uses natural guide samples ($x_g \in X_g$) that specify a distribution over the possible destinations. The success of this procedure depends on two factors: (i) the sensitivity of F_i to input perturbations and (ii) the initial distance between $F_i(x)$ and its destination $F_i(x_g)$. The sensitivity of F_i grows with its Lipschitz constant L, i.e., $\|F_i(x) - F_i(x+\delta)\| \leq L\varepsilon$, which allows the perturbations to have a greater impact [109].

In Figure 3.8, we experiment with our ResNet-50 model on CIFAR-10 to evaluate how the factors (i) and (ii) affect the success of SIA in evading detection. We apply the method in [110] to measure the average local Lipschitz constant around the holdout samples with respect to each layer in the model ($\|\delta\|_\infty \leq \varepsilon = 0.03$). First, we see that the Lipschitz constants increase throughout the layers, aligning with [111]. Second, to isolate the impact of (ii), we apply SIA at each layer in two settings based on the average pairwise representation distances between the samples in

different classes. In the *closest* and the *furthest* settings, we set X_0 and X_g as the samples from the class pair with the least and the most average distance, respectively. We then report the detection rates (DR_{50}) of DDs that individually operate on the layer attacked by SIA. This reveals that SIA is effective when X_0 and X_g are close (e.g., the *dog* and *cat* classes) as it does not need to move the initial samples significantly with respect to the representations. However, the furthest setting shows that when X_0 and X_g are far away (e.g., the *airplane* and *frog* classes), SIA might fail to evade detection. Here, as the Lipschitz constants increase throughout the layers, the detection rates drop rapidly: 100% at the 7th, 82% at the 12th and 0% at the 17th layers. These results suggest that reducing the model's sensitivity, e.g., via adversarial training [110], and increasing the representation distances between different classes, e.g., via orthagonalization [112], are promising directions to improve detection performance.

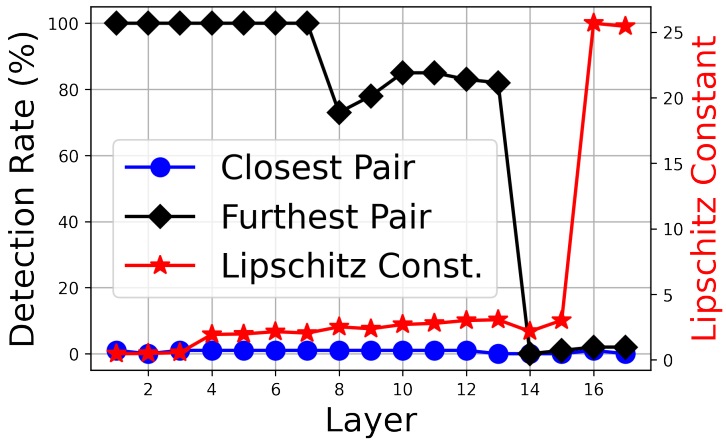

Figure 3.8: Evaluating the factors affecting the success of SIA.

3.4 Empirical Evaluation of SIA Against Individual Detectors

Although we have formulated SIA against DDs, individual detectors (IDs) that inspect a single sample are more realistic and, as a result, more common [66]. A typical ID quantifies how statistically similar a sample is to the known NEs and rejects it if this score is below a threshold. In this section, we aim to show how its formulation makes SIA effective against IDs as well.

We experiment with a ResNet-50 model on CIFAR-10, train the detectors on the whole holdout set and test them on a balanced set that contains 500 AEs and 500 NEs. We consider the AEs crafted by SIA (in Figure 3.4) and, as a baseline, by PGD (in Figure 3.1), both with near 100% ASR. We report the following popular metrics to assess the detection performance: (i) the false positive rate (FPR) at 95% true positive rate (**FPR95**); and (ii) standardized partial area under the receiver operating characteristic curve below 20% FPR (**pAUC**). These metrics reflect the realistic requirements of a detector, e.g., low FPR and high TPR, and remove the need for tuning the threshold manually. The scores for a chance level detector are 0.5 pAUC and 95% FPR95. For the perfect detector, they are 1.0 pAUC and 0% FPR95.

We evaluate the following four detectors from the literature.

Density artifacts [78] is one of the early detectors which assumes that the AEs lie far from the natural data manifold. It trains a kernel density model based on the PLR of the holdout samples and rejects a sample if its density is smaller than a threshold.

I-Defender [83] characterizes the natural data distribution at the PLR using a class-wise Gaussian mixture model (GMM). It rejects a sample if its likelihood estimated using GMMs is less than a threshold. For this detector, we report the average results over all classes.

The odds are odd [67] relies on the distribution of a DNN's logit values, in particular, it

assumes that the logits on AEs are less robust to noise than on NEs. Prior work has shown that the adaptive evaluation in this paper is incomplete and designed an attack against the defense [69]. We select this method to demonstrate how SIA, with no additional effort, can evade detectors that apply input noise.

JTLA [68] is the most recent detector we experimented on, using the implementation by its authors. JTLA (stands for joint statistical testing across DNN layers for anomalies) computes statistics regarding whether a given sample is similar to NEs in terms of its nearest neighbors at a layer. It then aggregates the statistics from all layers into the detection score for this sample.

	PGD		**SIA**	
Detector	FPR95	pAUC	FPR95	pAUC
Density	40%	0.75	95%	0.53
I-Defender	34%	0.77	92%	0.61
Odds	56%	0.71	88%	0.65
JTLA	83%	0.62	90%	0.60

Table 3.5: We present the detection results against PGD and SIA by reporting two metrics: FPR95 (lower is better) and pAUC (higher is better)

.

We present the results in Table 3.5. We see that all detectors have significantly worse performance against SIA, compared to their performance against PGD. For example, Density detector has near-chance level performance against SIA. Regarding JTLA, we observe that it already performs poorly against PGD, compared to other detectors. As a sanity check, we apply JTLA on the AEs crafted by DDN (in Figure 3.1), which results in over 0.9 pAUC. This shows

JTLA has inconsistent performance even against standard attacks. Overall, these results show that SIA, without customization, can be used as a benchmark to evaluate individual detectors as well as distributional detectors.

3.5 Attacking Other Detection Scenarios With SIA

The previous sections show how SIA evades methods designed to detect AEs. Here, we apply SIA in other detection scenarios to demonstrate its flexibility.

3.5.1 Dataset Shift Detectors

Seemingly subtle changes in the data distribution are known to hurt the performance of modern DNNs [113]. Prior work has developed methods to detect such *dataset shifts* that give a critical opportunity for practitioners to act [114]. We focus on two different shift scenarios and highlight the security risks of shift detectors as a new threat model.

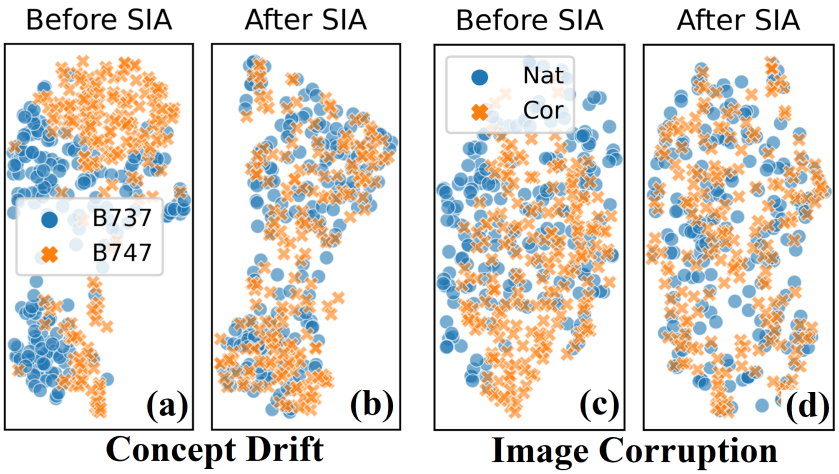

Figure 3.9: Attacking dataset shift detection with SIA.

Concept Drift Detection. Concept drift refers to the problem when the relationship between the

underlying data distribution and concept being learned changes over time. Adapting the setup by Kirchler et al. [115] and using the dataset by Maji et al. [116], we consider the drift from the images of the 'Boeing-737' aircraft to the 'Boeing-747' aircraft. In this setup, the detector applies a C2ST on the features it extracts from the images using the PLR of a pre-trained DNN on ImageNet. Figure 3.9 (*a*) shows how the two aircraft types have different feature distributions. As a result, with 100 samples from each distribution, the detector easily detects the drift with 100% DR. To evade this detector, we include the B-737 samples in our guide sets and perturb the B-747 samples using SIA against the DNN. We omit the second term in SIA objective as the detector discards the DNN's predictions. Figure 3.9 (*b*) hints that the perturbed B-747 and the B-737 distributions are the same. This brings the DR down to 0%, even with 300 samples, and confirms that SIA has made the drift detection ineffective.

Corruption Detection. Another type of shift occurs when environmental factors start corrupting the inputs. As a result, image corruptions, such as fog or snow, are shown to hurt the performance of DNNs [113]. This makes detecting such corruptions crucial in safety-critical DNN applications [114]. In our setup, we consider the snow corruption [113] applied on the test samples in Tiny-ImageNet. To detect the shift, we apply a DD on the PLR of a ResNet-50 Tiny-ImageNet model. Figure 3.9 (*c*) shows how the corrupted samples follow a different distribution than uncorrupted NEs. As a result, the DD reaches 100% DR_{100} and effectively detects the shift. Similar to the previous scenario, we use NEs in our guide sets and apply SIA to perturb the corrupted samples. The resulting perturbed samples have the same distribution as the NEs, as we show in Figure 3.9 (*d*). This leads to 0% DR_{300} and prevents the detection.

3.5.2 Out-of-Distribution (OOD) Detection

Due to its importance in enhancing the safety of DNNs, detecting OOD inputs has received much attention lately. ReAct [117], the state-of-the-art OOD detector, rectifies the penultimate-layer activations at an upper limit c. The authors claim that this empirically and theoretically leads to a better separation between the energy scores [118] of OOD and in-distribution inputs. Following the original setup, we apply ReAct on a ResNet-50 CIFAR-10 model and use the SVHN [119] dataset as the OOD inputs. An energy-based detector achieves 24% FPR95, which goes down to 19% with ReAct. We then perturb the SVHN samples with SIA, using CIFAR-10 samples in the guide sets. We reduce the perturbation-bound to $\varepsilon = 0.01$, instead of $\varepsilon = 0.03$, for better imperceptibility. As a result, ReAct achieves 95% FPR95 on the perturbed OOD samples, i.e., chance level performance. This shows that SIA can turn OOD inputs into in-distribution inputs, which exposes the vulnerabilities of many OOD detectors that rely on representation statistics.

3.6 Takeaways and Conclusions

In this chapter, we have introduced the *statistical indistinguishability attack* (SIA) as a general-purpose adaptive attack against a wide range of detectors, including distributional and individual ones. We have identified that typical constraints in the conventional adversarial examples (AEs) threat model, i.e., based on input-space imperceptibility, lead to detectable, and hence inconsequential, attacks. This has been causing a false sense of security as weaker attacks are used to evaluate AE detection methods. As a solution, SIA implements a more realistic constraint that forces the adversary to craft AEs that closely follow the distribution of natural samples with respect to hidden layer DNN representations. This allows SIA to avoid common detectors that

rely on observing statistical differences between adversarial and natural inputs. Thanks to its generic formulation, SIA compromises detection methods in various settings, with little-to-no customization. As detecting AEs is becoming increasingly critical for safeguarding DNNs, we believe our work will provide a reliable baseline tool to evaluate the security of detectors against adversaries who wish to avoid them.

Chapter 4: Natural Leverage of Adversaries Over Malware Detection With Machine Learning

In the previous chapters, we have described more realistic extensions to the goals (Section 2) and the constraints (Section 3) in the conventional adversarial examples threat model. Typically, the adversary is assumed to have the ability to perturb an existing input to a machine learning model to craft adversarial examples and pursue their goals. This post-hoc ability constitutes the leverage of the adversary over the defender. In this chapter, we will study an application where adversaries gain **natural leverage** over the defender as an innate outcome of the application: malware detection from runtime behaviors. Security analysts typically execute programs in a restricted environment (a sandbox) to safely collect their runtime behaviors, which are then processed to extract features and fed to a machine learning model that can determine whether a program is malware judging by its behaviors. We identify that the adversary gains two main sources of leverage against a malware detector primarily due to this practice. First, the set of programs (benign and malware) a typical detector is trained on does not represent the set of programs this detector will need to operate on in practice. Second, program behaviors are environment dependent, which introduces major differences between behaviors in sandboxes and behaviors in real-world environments.

Overall, these discrepancies introduce a distribution shift that makes a detector's performance in a controlled setting not a reliable indicator of its success in the real world. This distribution shift

creates a massive gap between the performance of a detector on sandbox traces (i.e., the controlled setting) and its performance on the in-the-wild (ITW) traces of the undetected samples (i.e., the realistic setting). Unlike the conventional threat model, this shift is inherent to the domain, which provides the attacker a natural advantage due to how malware detectors are designed and trained.

Our work in this chapter also explores modern machine learning methods to enhance a detector's robustness to distribution shift, which provides moderate performance improvements. Moreover, we also provide an in-depth study into various operational approaches a defender can take when designing a malware detector in this setting. Our nuanced study highlights the trade-offs and caveats in these approaches, in hopes of finding an optimal strategy to reduce the adversary's natural leverage. All in all, our evidence suggests that detecting malware in the wild is challenging and requires further attention from the community.

4.1 Background

Malware Detection In Practice. Detection of malware threats is crucial for governments, enterprises, and end users as there are significant (and growing) financial and safety harms of malware infections [120]. This has made malware analysis a lucrative $7 Billion industry in 2022 with many major players that offer various anti-malware solutions [121]. Malware detection appears to be remarkably effective: industry-standard evaluations of commercial anti-malware products [122], and prior research on malware detection using machine learning (ML) methods [123, 124], routinely report that over 99% of malware samples in a standard corpus can be detected, with very few false-positives.

In practice, malware detectors rely on a chain of techniques [125, 122] that starts with

conventional static methods that operate on the binary code of a program without executing it, such as allowlists/blocklists, reputation systems, and signatures. As static methods can be bypassed through obfuscation or polymorphism [126, 127], dynamic analysis has become standard, offered by most major companies [128]. This approach relies on monitoring a sample's run-time actions to identify whether it displays malicious behaviors. The typical paradigm for dynamic analysis is to *detonate* a large set of known samples (malware and benign) in a controlled environment (a *sandbox*), collect their execution *traces*, and learn to separate malicious and benign behaviors. The last step often involves using these traces to train an ML model that can infer the class (malware or benign) of an unknown sample based on its behaviors [129, 130, 124, 131]. In prior research and industrial practice, ML-based dynamic malware detectors are usually trained with only one trace per sample, collected from a pre-configured sandbox.

Dynamic Malware Analysis. Most work in dynamic analysis focuses on analyzing the behaviors of a sample detonated in controlled environments, such as sandboxes [132, 133]. As dynamic analysis has become a staple, malware has started including checks that suppress malicious activities if the environment is fingerprinted as a sandbox, known as evasive malware [134]. Although many strategies have been developed over the years to prevent fingerprinting [135], this is still an ongoing arms race with no end in sight [136]. This arms race motivates us to study the possibility of detecting malware using traces from the wild. Stepping outside sandboxes, researchers have developed methods to analyze samples in *bare metal* environments that eliminate the detectable elements of a sandbox [137, 138]. A recent large-scale measurement study over variable program behaviors in the wild has found that a given malware sample can behave significantly differently across time and in different real-world hosts [139].

ML for Malware Detection. Both in research and practice, ML-based methods are widely used with both static [123] and dynamic features [140]. In detection with dynamic features, methods that essentially treat a program's execution trace as a natural language document (a sequence of tokens) and adapt off-the-shelf methods from the ML community, have been shown successful [129, 131, 141]. Most of these methods are trained and evaluated on the traces from a pre-configured sandbox on samples collected from public repositories, some of the most popular ones can be found here [142].

Distribution Shifts in ML. Distribution shift occurs when the training and testing distributions of a model have significant differences, which hurts the performance [143]. For example, as new malware variants emerge and old ones disappear over time, the performance of a detector that has not been kept up-to-date will deteriorate, an example of distribution shift [144]. Although there is no silver bullet, the ML community has proposed many ideas to tackle distribution shifts in different contexts, such as learning domain invariant features [145, 146], distributionally robust optimization [147] or continuous learning [148].

Limitations of ML for Security. Research suggests that popular ML methods have many pitfalls when applied to security tasks [149]. For example, in malware detection, most work has been found to overestimate the success of the methods due to impossible time splits of training and testing sets [144]. A line of work also focuses on how ML methods successful in lab-only evaluations break down when they are deployed in the real world due to distribution shifts, for example, in the context of network anomaly detection [150], website fingerprinting [151] or malware detection [152, 153]. The models in these contexts are known to learn spurious features, such as specific IP addresses [150], that are the artifacts of the experiment setup in the lab and do

not apply to realistic settings.

4.2 Malware Detection in the Wild

Problem Statement. Research suggests that malware behaviors are *environment sensitive*, meaning that a sample may produce different traces when executed on different hosts or at different times [134, 137, 139]. While we might expect that this behavior variability causes a performance degradation when ML-based detectors are used in the real world, the extent of this degradation, and what we can do about it, remain open questions. Researchers have proposed a-priori strategies to address the problems that stem from variability, such as combining traces from multiple sandboxes [137] or randomizing sandbox configurations [154, 136]. As a last line of defense, some commercial malware detectors monitor the executions of hard-to-classify samples on hosts in the wild for signs of malicious behaviors [139]. It is difficult to determine the effectiveness of these approaches, and how to develop better detectors, without understanding how sandbox-based models perform in the wild. This also makes it impossible to assess if the apparent effectiveness of malware detectors in industrial and research evaluations is correlated to their effectiveness in deployment, as the success of ML methods in security tasks is often overestimated [144, 149, 150].

In this chapter, we present a quantitative study on the implications, for a typical behavior-based detector, of the variability between the sandbox (where the detector is trained) and real-world environments (where the detector is deployed). We focus on two recent deep learning-based detectors, MalDy [124] and Neurlux [131], that are highly successful when evaluated on sandbox traces, e.g., over 90% true-positive rate at 1% false-positive rate (90% TPR @1% FPR). To understand how these models fare in the wild, we use a dataset of around 1 million execution

traces from over 25 thousand samples (both malware and benign), recorded in real-world hosts of a commercial anti-malware vendor. Note that the vendor did not know whether a sample was benign or malicious when its traces were recorded. The samples were encountered and executed by hosts naturally and the vendor recorded their traces for dynamic analysis as a last line of defense, after which they remained undetected. As a result, our dataset contains malware infections that happened in the wild from the samples that evaded the defenses of the time. This allows us to perform *retrospective measurements* such as the number of real infections a model could have prevented if the vendor deployed it at a certain time. Through these measurements, we compare different approaches in multiple detection scenarios. Ultimately, this data reflects the realistic threats in the extremes with which the behavior-based detection models need to combat.

Terminology. A *sample* is an executable binary program, each identified by its unique SHA-256 hash. A *trace* is a sequence of behavioral actions (such as file or process creations) performed by a sample when it is executed in a computing environment. Our work focuses on the Windows operating system environments and samples. We refer to an in-the-wild (**ITW**) execution environment as a host. In our ITW dataset, the anti-malware system in each host recorded the traces of unknown samples executed by the host and each sample can have multiple traces recorded at multiple hosts at different timestamps. A sandbox (**SB**) refers to a controlled execution environment that provides tools to analyze samples and record their traces. Our sandbox dataset contains traces collected from three different sandboxes (SB1, SB2, and SB3).

Notation. As our setting in this chapter is different than the previous ones, we define a new notation that is more expressive for malware detection with ML. We denote the set of samples in our dataset as $\mathbf{P}$, P_i is the i-th program and y_i is its associated label; where $y = 0$ and $y = 1$

indicate a benign and malware sample, respectively. If P_i is a malware sample, it is also tagged with a family label s_i that is useful for grouping samples with similar characteristics. A trace of P_i is x_i^j, where j denotes the execution environment, specifically, $j=0$ refers to the ITW hosts and $j \in \{1,2,3\}$ refers to the sandboxes. As there are multiple ITW traces per sample, we refer to the k-th one as $x_{i,k}^0$, and the ITW traces of a sample are sorted according to their timestamps. The timestamp of its earliest observed trace—$x_{i,0}^0$—marks the first time the sample was first seen in the wild, and $\mathbf{x}_{i,(t<h)}^0$ is the set of all ITW traces recorded within h hours of this.

An ML-based detection model takes a trace x_i of P_i as its input and aims to infer y_i. We split a model—f—into two parts, an encoder—enc—and a classifier—g. The encoder produces a vector embedding—$z = enc(x_i)$—of the input trace and $f(x) = g(enc(x_i))$ is the model's predicted probability (*score*) that P_i is malware. The predicted label $\hat{y}$ is $\hat{y}=1$ if $f(x) \geq \tau$, or $\hat{y}=0$, otherwise. Here, τ is a threshold tuned based on the defender's requirements, where a higher τ results in fewer false-positives in exchange for fewer true-positives. In Section 4.2.2, we discuss the tuning of τ in more detail.

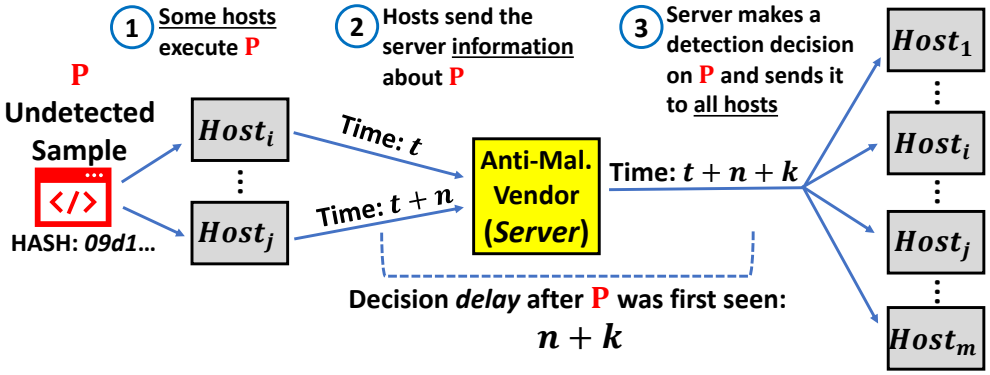

Figure 4.1: The workflow of malware detection in the wild.

4.2.1 Characterizing ITW Detection Scenarios

In Figure 4.1, we provide an overview of the workflow for malware detection in the wild. In Step (1), a new sample P is naturally encountered by some hosts (e.g., from an email attachment) at different timestamps. If the sample is *undetected* (neither as malware nor as benign) by its defenses, the anti-malware system records the trace x while a host executes P. In Step (2), after the execution finishes, a host sends *information* about P to the server operated by the anti-malware vendor. This information includes the sample P itself and its trace x at the host. Anti-malware vendors often have terms-of-service that allow them to collect such information from their hosts [155]. The timestamp when the vendor first receives information about P (t in our diagram) marks the time P was *first seen*. In Step (3), after receiving from at least one host about P, the vendor makes a decision on P (malware or benign) and sends it out to all hosts. Decision delay ($n + k$ in our diagram) is the gap between P was first seen and the decision on it was sent out. If P is classified as malware, any subsequent attempts of hosts to execute P will be blocked by the anti-malware system. As our study is concerned with the ML aspect of this workflow, we make assumptions such as perfect communication in Step (2) and instant updates in Step (3).

From the vendor's perspective, there are two main phases: *training* and *testing*. In the training phase, the vendor collects samples (either public samples from sources such as MalwareBazaar [156], or the undetected samples from their hosts) and the traces of these samples (either from sandboxes or from the hosts). If the samples came from public sources, the vendor can label them using a public ground truth source, such as VirusTotal [157]. On the other hand, the vendor has to allocate resources to label undetected samples, often done by malware analysts [158]. At the end of the training phase, the vendor trains a model for malware detection with dynamic

features; and deploys it to make decisions on the undetected samples encountered by the hosts in the testing phase. Although we view them as sequential, these phases can be interwoven as vendors continuously update their models on new data [148]. According to this workflow, we taxonomize various detection scenarios along five dimensions. The dimensions concerning the samples in the training phase are **(i)** their source; **(ii)** their execution environment; and **(iii)** how they are labeled. The dimensions concerning the samples in the testing phase are **(iv)** their source, and **(ii)** their execution environment.

Detection	Training Phase			Testing Phase	
Scenario	Samples	Env.	Labels	Samples	Env.
S-0: SB-Only	Public	SB.	VTotal	Public	SB.
S-1: SB-on-ITW	Public	SB.	VTotal	Undet.	Hosts
S-2: ITW-on-ITW	Undet.	Hosts	Analyst	Undet.	Hosts
S-3: SB-on-SB	Public	SB.	VTotal	Undet.	SB.

Table 4.1: Studied scenarios for malware detection in the wild.

In Table 4.1, we present the four scenarios we study based on this taxonomy. **S-0** is the scenario most prior work focuses on, both training and testing samples come from public sources and are executed in a sandbox [124, 131, 129, 153]. It is already known that malware detectors excel in this scenario. In **S-{1,2,3}**, instead of on the samples from public repositories, the detector is deployed on the undetected samples encountered by the real-world hosts. In **S-1**, the model is trained on the sandbox traces of public samples, similar to **S-0**. The performance gap between **S-0** and **S-1** quantifies the performance degradation of a sandbox-only detector in the wild. We focus heavily on this gap to demystify the success of typical detectors in the wild. In

S-2, the model is trained on the traces of undetected samples collected from real-world hosts. Here, in the training phase, the vendor needs to rely on analysts to label these samples using their ITW traces as they cannot be labeled as conveniently via VirusTotal as public samples. Moreover, the detectors in this scenario have a cold start problem as the vendor needs to wait for some time to collect enough ITW traces (and allow for infections) for training. Finally, in **S-3**, the model is trained the same way as in **S-{0,1}**, however, the vendor detonates the ITW samples in a sandbox and uses the resulting traces to make a decision. Here, the vendor essentially avoids using the detector on ITW traces altogether. The required sandbox execution will introduce a delay [158], which we will account for in our evaluation.

4.2.2 Success Metrics for In-the-Wild Detection

The conventional goal of a detector is to detect as many malware samples as possible, i.e., the true-positive rate (TPR), while minimizing the ratio of benign samples misclassified, i.e., the false-positive rate (FPR). The defender can control the TPR and FPR by tuning the detection threshold τ, a higher τ implies a lower TPR and FPR. A model with random decisions would achieve K%TPR @K% FPR. Following prior work [159], we evaluate a model with respect to its TPR @1%FPR (and @5%FPR) on its test sets. This provides us with an unambiguous way to compare our models in different scenarios.

The Impact of Decision Delays. Metrics such as TPR/FPR operate at the level of individual samples, whereas malware detection in the wild is also concerned with preventing real-world infections. Figure 4.2 (*left*) shows that a few malware samples have over 200 traces, each corresponding to a real-world infection. This implies that not all true-positives are equal in terms

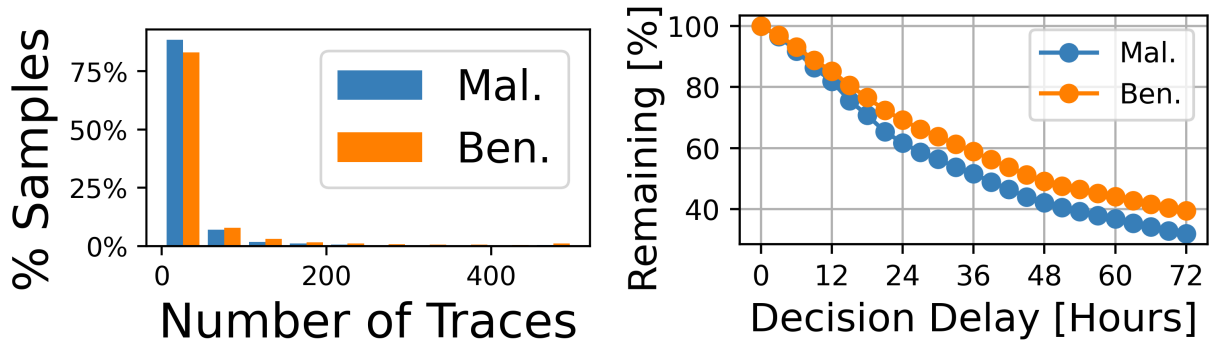

Figure 4.2: Statistics on the ITW traces in our dataset. (*Left*) The number of traces per sample. (*Right*) The average ratio of remaining traces per sample as a function of time.

of the number of infections prevented. Further, in Figure 4.2 (*right*), we see that over 60% of the infections of an average malware sample happen within 48 hours of its first-seen time. Prior work also suggests that most malware samples die within days [158]. In consequence, a true-positive on a sample might be essentially inconsequential in terms of prevented infections if the decision delay is too high. Overall, we quantify this notion with the *prevented infections ratio* (**PIR**) which is given by the total number of prevented infections over the number of all infections.

On the other hand, a false-positive implies that the regular activities of hosts (i.e., executions of benign samples) might be interrupted by the anti-malware system. To quantify this notion, we define the *interrupted activities ratio* (**IAR**) that is given by the total number of interrupted activities over the number of all activities. Similar to TPR/FPR, we evaluate a model with respect to its PIR at 1% and 5% IAR on its test sets. We use PIR/IAR to compare various detection strategies that involve different degrees of delay.

4.3 Technical Details

4.3.1 Datasets

	Training		Testing	
Dataset	Mal.	Ben.	Mal.	Ben.
ITW (Samples)	0.5K	16.5K	0.4K	8.2K
ITW (Traces)	11.5K	509.5K	9.7K	396.5K
SB1	35.6K	4.8K	20.3K	15.0K
SB2	32.3K	7.8K	13.4K	7.2K
SB3	–	–	2.1K	0.2K
SB1 ∩ SB2	7.4K	1.6K	6.3K	5.3K
SB-ITW Overlap (Disjoint From SB Dataset)				
SB1 ∩ ITW	16	461	48	1118
SB2 ∩ ITW	76	1015	58	634

Table 4.2: The summary of our two datasets used in Chapter 4 and their overlap.

In-the-Wild Dataset. Our ITW dataset is a collection of around 1M execution traces of a well-known anti-malware vendor collected across its Windows hosts in over 100 countries between January and July 2018. The data is collected by the vendor's behavior-based detection engine which records high-level traces of samples that execute until they naturally terminate. There are traces of around 900 malware and 25K benign Windows executable samples that were unknown

to the vendor at the time of collection. We can view this set of samples as a result of a filter that removes the samples that could be detected by traditional means, such as static analysis, and leaves only the samples that need to be detected via behavioral analysis. Even after behavioral analysis, these samples still remained undetected (either as malware or benign) and they were allowed to execute in the host, causing *real-world infections* when the sample turned out to be malware. Each item in our dataset consists of a sample's trace, along with this sample's SHA-256 hash and a timestamp that indicates when the vendor received this trace from the host. A limitation of our data is that it does not include network-related actions and only includes file, registry, process, and mutex actions. We do not expect this to affect our results significantly as ML-based detectors can still perform well without network actions [131].

Sandbox Dataset. A common prior practice is curating many samples, detonating them in a sandbox, and using the traces for training a model [124, 131, 129, 141]. We, however, cannot follow this practice as our ITW dataset is collected 5 years ago, in 2018. The samples we can find in 2023 would violate causality [144] as we will be using our ITW data as a test set. Moreover, even if we collect samples released in 2018, their behaviors now would not be representative of their original behaviors, as a sample starts showing different behaviors over time [139] or it simply stops functioning [158].

Our solution is collecting our traces from VirusTotal [157] where third-party sandbox vendors publicly share behavior reports on samples. To this end, we curate a list of SHA-256 hashes of Windows samples from popular public malware detection benchmarks, including EMBER [160] and SOREL [161], released in 2017 or 2018. We then query VirusTotal with these hashes to collect their sandbox traces when available. We only keep a trace if its execution

timestamp is at most a week after the date when the sample was first seen by VirusTotal to ensure that the trace represents the sample's original behaviors. This process results in traces from three different sandbox vendors for around 120K samples (90K malware and 30K benign). The reason for the class imbalance is that sandbox vendors on VirusTotal are more inclined to share traces of malware samples. Moreover, we also have queried VirusTotal for the samples in our ITW dataset and found the sandbox traces for around 3K of them, which we refer to as the SB-ITW overlap set in Table 4.2.

Temporal Splitting and Labeling. We split our datasets into two chunks based on the timestamps of the samples (the first-seen dates) to ensure that the train and test sets are temporally disjoint, to avoid a common pitfall in prior work [144, 149]. We select April 1st, 2018 as the split date, i.e., the date that separates the vendor's training and testing phases. Samples seen before this date, along with their traces, are in the *Training* chunk and samples seen after it are in the *Testing* chunk, on which we never train. As labels of samples are known to change over time [162], we make the best effort to assign the training samples historical labels that were available before the split date. For testing samples, we query VirusTotal to obtain the most recent detection reports and label samples that are detected by more than 5 anti-malware engines as malware, following [162]. Further, we use AVClass2 [163] to assign family labels to our malware samples based on their VirusTotal detection reports. We tag the samples that were labeled as malware but did not receive a family label as `generic` malware. Overall, this methodology ensures that our evaluation is realistic and reflects the conditions in which the anti-malware vendor had to operate when our ITW data was collected.

Trace Pre-Processing. The traces we have from four different sources (ITW and three sandboxes)

all have different formats and conventions. For interoperability (e.g., train on a sandbox and test on ITW), we convert all of our traces to a standardized format. First, we only keep the action types that all formats share, which are file creation, registry key creation and deletion, process creation and injection, and mutex creation. Second, we clean up the strings (e.g., file names) in each trace by removing white spaces, capitalization, punctuation, non-ASCII characters, and so on. Third, we tokenize the entries in each trace (e.g., split a full file path into its components) and save a trace as a sequence of tokens, following [131]. Finally, when we train a model, we only keep the top-10K tokens in terms of frequency in the training traces and replace the remaining tokens with special tokens, such as `<rare_file_name>`, again following [131]. This makes the task more suitable for an ML model by eliminating uninformative tokens and reducing the feature dimensionality. Moreover, this standardized format might allow us to implement a leaderboard in the future for malware detection in the wild by providing all participants with an expected input format that their submitted detectors should accept.

4.3.2 Machine Learning Details

We adapt two recent ML approaches in malware detection with dynamic features: **(i)** MalDy: a bag-of-n-gram-based model [124] and **(ii)** Neurlux: an attention-based sequence model [131]. In MalDy [124], we convert each trace into a list of 2-grams. For example, the file path `a/b/c.jpg` is turned into three 2-grams: `<a/b>`, `<b/c>` and `<c.jpg>`. As this results in an intractable number of unique 2-grams (mostly very rare), we apply the hashing trick [164] that assigns a numerical value up to 2^{14} to each 2-gram. The final feature vector—x—for a trace is a 2^{14}-dimensional vector and each dimension is set to the number of occurrences of the corresponding

2-gram in the trace. We use a ResNet-based architecture [165] to train on these features. In Neurlux [131], we treat a trace as a natural language document, and train an attention-based sequence classification model. The benefit of this approach is eliminating the need for feature engineering (unlike the n-gram approach) and it can extract useful features from long sequences thanks to the attention mechanism. Note that our goal is *not* developing a better model architecture or feature processing routine but to evaluate state-of-the-art ones on the problem of malware detection in the wild. We make our ML improvements without changing the backbones of these existing approaches.

Hyperparameter Tuning. We set aside 10% of the samples in the training chunk as our validation sets (three sets total: SB1, SB2 and ITW). We perform a grid search over possible hyperparameter settings, such as the number of layers, layer sizes, learning rate, batch size, and so on, and select the best setting based on the resulting model's performance (TPR @1%FPR and TPR @5%FPR) on these validation sets, individually. By using validation sets from different environments, we can study the potential trade-offs between the model's performance, for example, on SB1 traces vs. on ITW traces.

4.4 Sandbox-Only Evaluation Gives a False Sense of Security in the Wild

In this section, we aim to understand how well a typical model trained only on sandbox traces performs on the ITW traces of undetected samples. This corresponds to the performance gap between **`S-0`** and **`S-1`** in Table 4.1. We select the very first recorded ITW trace of each undetected sample to make a decision on this sample, meaning that we simulate making decisions without any delay. In Table 4.3 and Table 4.4, we share the results of using two different sandboxes as

our training environment. Here, we train both N-gram-based and attention-based models and tune the hyper-parameters to achieve the best performance on SB1, SB2, and ITW validation sets, as described in Section 4.3.2. Across the board, when the model is tuned on the validation set of the training sandbox, we see a huge performance gap between the test set of the training sandbox and the ITW test set, e.g., 92.4% vs 3.8% TPR @1%FPR for the N-gram model trained on SB1. This gap often shrinks when the model is tuned on the validation set of another sandbox, e.g., 84.7% vs 10.3% for the N-gram model trained on SB1 and tuned on SB2. However, the gap is still substantial even in the ideal case when we tune the model on the ITW validation set, e.g., 80.3% vs 13.6% for the N-gram model trained on SB1.

TUNE / TEST	SB1 VALID.		SB2 VALID.		ITW VALID.	
	NGRAM	ATTN.	NGRAM	ATTN.	NGRAM	ATTN.
TRAINING ENVIRONMENT: SANDBOX 1						
`s-0` – SB1	92.4%	88.5%	84.7%	87.0%	80.3%	82.8%
`s-0` – SB2	39.4%	54.8%	55.0%	67.6%	51.4%	52.5%
`s-0` – SB3	34.0%	27.2%	11.9%	31.7%	29.3%	19.8%
`s-1` – ITW	3.8%	4.3%	10.3%	6.5%	13.6%	8.8%
TRAINING ENVIRONMENT: SANDBOX 2						
`s-0` – SB1	42.2%	42.8%	12.9%	7.5%	19.0%	36.4%
`s-0` – SB2	86.2%	63.4%	90.1%	90.0%	52.4%	83.6%
`s-0` – SB3	26.8%	9.4%	9.9%	42.7%	10.0%	36.0%
`s-1` – ITW	10.6%	4.8%	1.0%	2.3%	12.1%	6.8%

Table 4.3: The performance (TPR @1%FPR) of models trained only on sandbox traces on different sandbox and in-the-wild test sets.

TUNE / TEST	SB1 VALID.		SB2 VALID.		ITW VALID.	
	NGRAM	ATTN.	NGRAM	ATTN.	NGRAM	ATTN.
TRAINED ON SANDBOX 1 TRAINING SET						
S-0 – SB1	94.7%	92.6%	93.1%	92.6%	86.5%	91.8%
S-0 – SB2	63.2%	77.3%	79.7%	79.7%	76.1%	68.2%
S-0 – SB3	60.8%	44.4%	36.2%	62.6%	62.7%	37.4%
S-1 – ITW	13.3%	19.6%	30.9%	23.9%	34.7%	23.4%
TRAINED ON SANDBOX 2 TRAINING SET						
S-0 – SB1	47.1%	75.3%	52.4%	13.8%	25.2%	45.6%
S-0 – SB2	91.8%	83.9%	93.2%	93.2%	81.2%	91.6%
S-0 – SB3	63.1%	20.7%	50.1%	65.2%	28.1%	75.7%
S-1 – ITW	16.8%	20.3%	12.8%	14.3%	19.8%	20.3%

Table 4.4: The performance (TPR @5%FPR) of models trained only on sandbox traces on different sandbox and in-the-wild test sets.

Overall, this gap persists regardless of the training sandbox or the model type. More importantly, this gap is the largest for models that perform the best on the test traces from the sandbox that also produced the training set. In consequence, studies that rely on a single sandbox in **S-0** [129, 124, 141] are at a higher risk of giving a false sense of security the performance in **S-1**. Moreover, it is possible to close the gap slightly by tuning the model on a different sandbox than the one it was trained on. Finally, the n-gram-based model performs better in **S-1**, e.g., 13.6% vs 8.8% for the models trained on SB1, which, we believe, is because it is relatively simpler, easier to tune, and less inclined to overfit.

Next, we will investigate some of the causes of the gap and explore ML-based methods for

closing it. In particular, we aim to understand the impact of the *distribution shift* between the training and testing sets in **`S-1`** (sandbox traces of public samples vs. ITW traces of undetected samples). In our case, distribution shift implies a change in the process that generates the inputs, i.e., program traces for detecting malware, between the training and testing phases.

Takeaways. A model with a higher performance on a sandbox-only test set might end up actually performing worse on in-the-wild traces, potentially creating a false sense of security.

4.5 Sample Shift Hurts the Performance

As we discuss in Section 4.3.1, the training samples of a typical detector (in **`S-0`** and **`S-1`**) are collected from public repositories and benchmarks. This provides an almost *uniform* sample over all samples, malware or benign, seen in the wild during a time period. Conversely, the samples in our ITW dataset are *not* a uniform sample but are selected based on whether they were undetected (either as malware or as benign) by the anti-malware system during a time period. In consequence, the training samples in **`S-1`** do not follow the same distribution as the undetected samples collected from the hosts on which the detector will be deployed. We refer to this problem as sample shift and study it for both malware and benign samples.

Methodology. To measure the impact of sample shift, we take different subsets of the test sets according to some criteria, e.g., certain malware families only, and compare the model's performances on them. This allows us to simulate different distributions for a model's test samples and gives us control over sample shift. Due to its performance, we focus on the N-gram-based model trained on the SB1 training set and tuned on the ITW validation set.

4.5.1 Malware Family Shift

In Figure 4.3 (*top row*), we rank malware families according to their frequencies in the SB1 training set *(top left)* and in the ITW test set *(top right)*, and measure cumulatively the coverage (*Cov.*) of these families in different sets of samples. For example, the non-generic families that cover 90% of the SB1 training set, cover less than 20% of the ITW test set. As expected, the shift between the SB1 training and SB1 testing sets is much smaller. Moreover, the families that cover 100% of the ITW test set cover less than 20% of the SB1 training set and 20% of the families in the ITW test set are missing from the SB1 training set altogether. In the ITW test set, most frequently we see families such as Wannacry, Emotet and Gandcrab that were emerging and extremely harmful in 2018 [166]. On the other hand, the most frequent families in SB1 training set include Virut, Sivis and Wapomi that are older (from 2013-2014), well-understood, and whose variants still actively spread today. As a result, even though our sandbox and ITW datasets include samples seen around the same time (2018), they follow vastly different distributions.

After demonstrating a distribution shift in malware samples, in Table 4.5 (*first segment*), we answer how this impacts the performance of an ML detector. Here, we keep the benign samples fixed in each test set and select different subsets of the malware families when measuring the performance. In particular, we focus on three subsets: **Top-Train** is the top-100 families in the SB1 training set, **Top-ITW** is the top-30 families in the ITW test set and **Generic** is the malware samples that could not be assigned a family label. For example, on Top-Train vs. Top-ITW, there is a significant gap in the model's performance in **`S-0`**: 91.7% vs 73.2%, similarly in **`S-1`** as well: 18.1% vs 16.1%. Further, we see that the model does significantly worse on Generic compared to other subsets, as these samples are at the borderline [167]. Generic covers a much bigger portion

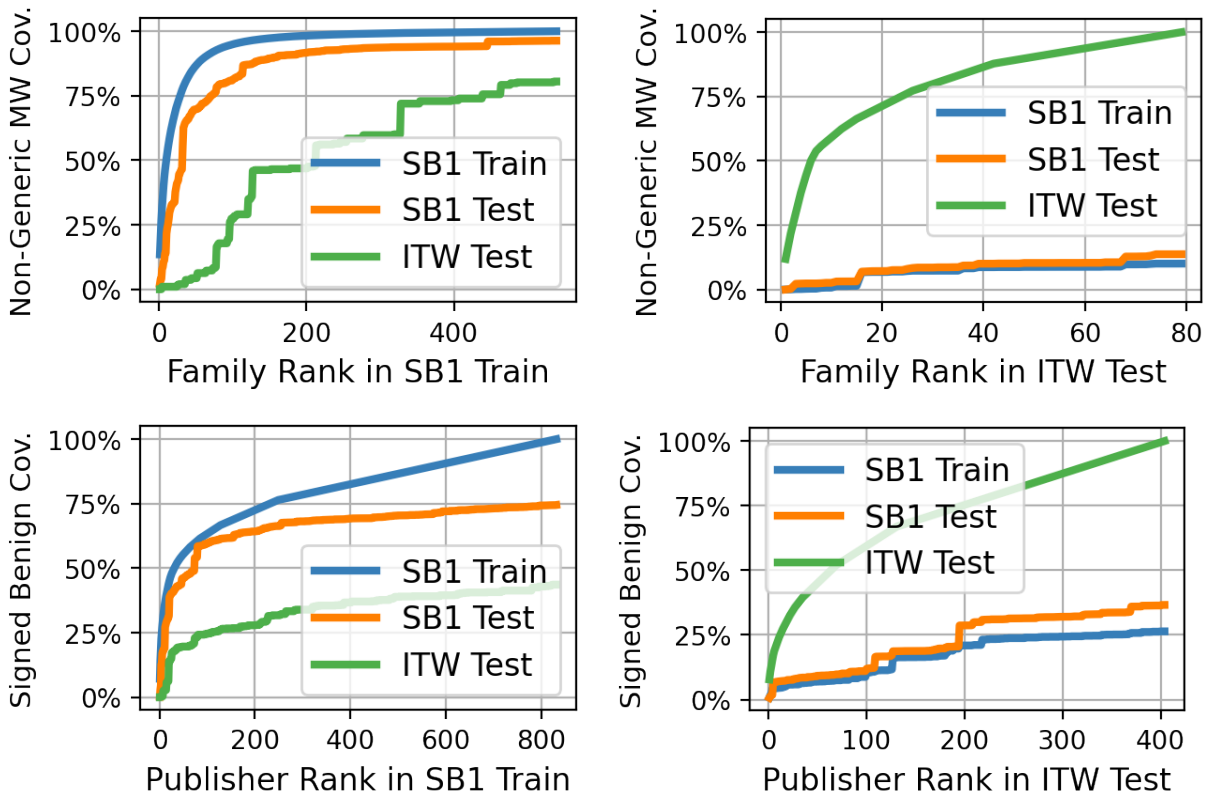

Figure 4.3: Quantifying the extent of sample shift in malware families (*top row*) and benign publishers (*bottom row*).

of the ITW test (24%), compared to the SB1 Test (6%), aligning with our observation that our ITW dataset contains more samples that could not be detected as easily. Overall, this shows that the model overall performs better on families that are more frequently represented in its training set, demonstrating the impact of sample shift.

SCE. / SUBSET	S-0 – SB1 TEST			S-1 – ITW TEST		
	COV.	1%FPR	5%FPR	COV.	1%FPR	5%FPR
SECTION 4.5.1 — MALWARE FAMILY SUBSETS						
TOP-TRAIN	77%	91.7%	95.9%	21%	18.1%	48.2%
TOP-ITW	8%	73.2%	89.1%	61%	16.1%	39.3%
GENERIC	6%	24.6%	35.0%	24%	7.4%	21.1%
ALL MAL.	100%	80.3%	86.5%	100%	13.6%	34.7%
SECTION 4.5.2 — BENIGN PUBLISHER SUBSETS						
TOP-TRAIN	21%	89.7%	91.9%	6%	25.9%	38.2%
TOP-ITW	4%	84.1%	90.1%	12%	22.4%	35.9%
UNSIGNED	48%	76.8%	86.8%	66%	14.3%	36.2%
ALL BEN.	100%	80.3%	86.5%	100%	13.6%	34.7%
SECTION 4.5.3 — DRO LOSS OVER MAL. FAMILIES ($\alpha = 1.0$)						
TOP-TRAIN	77%	24.6%	79.6%	21%	30.1%	72.3%
TOP-ITW	8%	50.8%	85.3%	61%	20.2%	50.8%
GENERIC	6%	22.1%	39.2%	24%	9.5%	18.9%
ALL MAL.	100%	27.2%	76.9%	100%	16.8%	39.9%

Table 4.5: The impact of sample shift on ML performance.

4.5.2 Benign Sample Shift

Although the focus of malware detection is on malware, detectors, to maximize the performance, also learn to recognize the features of benign samples in their training sets [168, 169]. However, most benign samples are often difficult to share or find in public sources due to legal reasons [160]. As a result, typical sandbox-only detectors are trained on benign samples that are scraped from accessible publishers [131], such as Microsoft or Google; whereas the undetected benign samples in the wild do not have such restrictions and can be from obscure publishers. In Figure 4.3 (*bottom row*), we quantify this mismatch and measure the shift in benign publishers in the SB1 training set *(bottom left)* and the ITW test set *(bottom right)*. We find the publisher information of a benign sample from its code-signing certificate if it is available, if not we call it unsigned and exclude it from this measurement. For example, the publishers that cover 80% of the SB1 training set, cover less than 30% of the ITW test set. Moreover, the publishers that cover 100% of the ITW test set cover less than 40% of the SB1 training set, and 55% of the publishers in the ITW test set are missing from the SB1 training set altogether. The most frequent publishers in the SB1 training set include Mozilla, Opera, Google and Microsoft, all highly reputable and publish easy-to-find samples, as expected. On the other hand, most of these publishers are not among the top in the ITW test set. We believe this stems from the fact that anti-malware vendors use allow-lists and reputation systems that bypass triggering behavioral analysis on well-known samples signed by reputable publishers [125, 141]. In consequence, the distribution of the benign samples that are commonly used for training sandbox-only models is not representative of the benign samples in our ITW dataset.

After demonstrating a distribution shift in benign samples, in Table 4.5 (*second segment*),

we answer how this impacts the performance of a detector. Here, we keep the malware samples fixed in our test sets and select different subsets of the benign publishers when measuring the performance. In particular, we focus on three subsets: **Top-Train** is the top-100 publishers in the SB1 training set, **Top-ITW** is the top-30 publishers in the ITW test set, and **Unsigned** is the benign samples that do not have a certificate. The patterns in the previous section also are visible here. For example, on Top-Train vs. Top-ITW, there is a gap in the model's performance in **`S-0`**: 89.7% vs 84.1%, similarly in **`S-1`** as well: 25.9% vs 22.4%. This also shows that the model's performance in **`S-1`** on Top-Train publishers is almost twice as much as its performance on all benign samples, 25.9% vs 13.6%. Further, we see the model does significantly worse on Unsigned compared to other subsets, 25.9% vs 14.3% in **`S-1`**. We believe, similar to generic malware, unsigned benign samples are also rarer, at the borderline, and generally harder to classify. Unsigned covers a bigger portion of the ITW test (66%), compared the the SB1 Test (48%), which shows that the ITW dataset contains more rare samples than the SB dataset. Overall, the model performs better on publishers that are more frequently represented in its training set, again demonstrating the impact of sample shift.

Our results highlight the (underappreciated) importance of the benign samples a detector is trained and tested on for its performance, especially for detection in the wild that deals with rarer benign samples. Anti-malware companies frequently share malware samples with each other, either through private feeds [167] or through platforms such as VirusTotal. However, sharing benign samples is problematic and often not allowed, which is a roadblock to training detectors on a realistic benign distribution. We believe methods such as private synthetic data generation [170, 171] or federated learning [172] are promising as solutions to this limitation.

4.5.3 Distributionally Robust Optimization (DRO)

In this section, we explore a potential solution to the sample shift problem. The ideal solution, of course, would be collecting enough samples from the ITW distribution and training our model on the sandbox traces of these samples. However, considering **(i)** the ITW distribution might be unknown to the vendor during the offline phase, and **(ii)** collecting more samples (especially benign) is challenging; we focus on a more practical, ML-based solution.

The main reason why the model performs well on the families frequently represented in the training set and not-so-well on the infrequent ones is the way standard ML models are trained: i.e., empirical risk minimization (ERM). ERM simply minimizes the following loss averaged over all training samples, $\{(x_1, y_1), ..., (x_N, y_N)\}$:

$$\mathcal{L}_{ERM} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f(x_i), y_i)$$

The loss function $\mathcal{L}$, such as cross-entropy, measures how far off a model's output on x_i is from the ground truth label y_i. ERM, despite being the standard, comes with a caveat: the average loss is dominated by the frequent groups (e.g., malware families) in the training set. As a result, ERM is unfavorable to infrequent groups as producing a high error on them does not change the average error significantly [173]. The ML community commonly refers to this as the group robustness problem and studies this in the context of fairness to underrepresented demographic groups in the training set [173]. A promising solution to this problem is distributionally robust optimization (DRO) [174]. Instead of the average loss over all samples, DRO computes the average loss over each group of samples individually and minimizes the maximum of these losses. This ensures that the model performs similarly on each group in the training set even in the

worst case. In our case, this would mean equalizing the model's performance on each malware family (or benign publisher), regardless of its frequency in the training set, and remedying the sample shift problem. However, it is known that DRO is not robust to noise in group labels [175], whereas, malware family labels are known to be incomplete (e.g., generic malware [176]) and fairly noisy [163]. For example, in our SB1 training set, there are many very likely noisy malware family tags such as `vbkryjetor` (3 samples). To avoid overfitting on these, we tailor DRO in three ways: **(i)** we only include malware families that have over 5 samples and **(ii)** we minimize a linear combination of the DRO and ERM loss function, balanced with a hyper-parameter α, and **(iii)** we minimize the average loss over each family, instead of the maximum loss. We experimentally have observed that these modifications make our application of DRO more robust to the noise in family tags. Overall, we define our final loss function as follows:

$$\mathcal{L}_{DRO} = \frac{1}{|G|} \sum_{g \in G} \left(\frac{1}{|g|} \sum_{(x_i, y_i) \in g} \mathcal{L}(f(x_i), y_i) \right)$$

$$\mathcal{L}_{Final} = (1 - \alpha)\mathcal{L}_{ERM} + \alpha \mathcal{L}_{DRO}$$

Here, each $g \in G$ is a set that contains the training samples corresponding to a family that we include in the DRO loss.

In Table 4.5 (*third segment*), we present the results of applying this loss function over malware families. We focus on showcasing DRO and leave its applications for other types of groups, such as benign publishers, to future work. First, note how the overall performance in **`S-0`** has dropped significantly as our training no longer minimizes the error on an average training sample. The model now performs better on Top-ITW than on Top-Train in **`S-0`**, hinting that DRO

is effective at preventing frequent families from dominating. Most importantly, the model's TPR @1%FPR and TPR @5%FPR in **`S-1`** have gone up by 23% and 15%, respectively, over our best model trained with ERM. This indicates that DRO can be a promising solution to the sample shift problem.

The Impact of α. In our hyperparameter search, $\alpha = 1$ has resulted in the best performance on the ITW validation set. This means that we have trained our model only on $\mathcal{L}_{DRO}$. Overall, the ideal α will depend on the extent of the sample shift between the training and the testing sets. In cases with less extreme shifts than ours, a smaller α will likely work better by balancing ERM and DRO.

Takeaways. Samples (both malware and benign) on which a malware detector in the wild needs to operate follow a different distribution than the samples that can be collected from public sources, on which sandbox-only models are often trained. To boost the performance in the wild, we propose an ML method for making a model more robust to such distribution shifts.

4.6 The Impact of Environment Shift

We have studied the differences in the sample distributions between our sandbox and ITW datasets. However, even the same sample can behave differently in different environments [137]. Such behavior differences will still cause a distribution shift between sandbox and ITW traces even without sample shift. We refer to this as environment shift and, in this section, study its implications.

4.6.1 Sandbox-Specific Artifacts

A common way to study environment shift is by detonating a sample in multiple sandboxes with different configurations and comparing the resulting traces [134]. For example, if the traces show semantic differences, it could be concluded that a malware sample might be trying to evade one of the sandboxes. We follow a similar procedure by using the traces for ∼9K samples we have overlapping between our SB1 and SB2 training sets (SB1 ∩ SB2 in Table 4.2). In particular, we ask whether there are features highly prevalent (seen in many traces) and predictive (seen only in malware traces) in one sandbox but not in the other. Such features, if exist, are exploited by ML models to minimize the loss, and other predictive features fail to be discovered [177]. Unless these features are also prevalent and predictive in the ITW traces, this would hurt the model's ITW performance as the model learns features useful only for SB traces.

In Table 4.6, we present four of these features (called *artifacts*) that we have found in SB1 ∩ SB2. We specifically focus on executable file names in either file creation or in process injection actions. Process injection is commonly done by malware to execute arbitrary code in the address space of a separate process to evade defenses. For each artifact, we report its prevalence (**Prv.**) and its malware ratio (**MalR.**) in SB1, SB2 and ITW test sets. Prv. is the percentage of samples in which the artifact is present and MalR. is the percentage of these samples labeled as malware.

Understanding the Artifacts. `SogouExplorer` is a Chinese web-browser and 6.5% of the samples in the SB1 testing set (all malware) interact with it, whereas no trace in the SB2 and very few traces in the ITW testing sets contain it. The majority of samples that interact with this artifact belong to families such as Sivis, Memery and Neshta, all tagged as file infectors that attach their code to other programs or files to propagate. Considering that SB1 is developed by a Chinese

File Name \ Data	SB1 Test		SB2 Test		ITW Test	
	Prv.	MalR.	Prv.	MalR.	Prv.	MalR.
Artifacts Found in Sandbox 1 Traces						
SogouExplorer	6.5%	100%	0.0%	—	0.2%	0.0%
PersonalBankPortal	3.4%	99.9%	0.0%	—	0.0%	—
Artifacts Found in Sandbox 2 Traces						
Spotify	0.01%	0.0%	3.3%	100%	0.1%	0.0%
Python	0.0%	—	2.9%	100%	0.01%	0.0%

Table 4.6: Some artifacts of maliciousness found in sandboxes.

vendor, we believe that, they pre-install `SogouExplorer` to their sandboxes to generate an analysis environment representative of Chinese hosts. This, however, creates features very specific to this sandbox as malware samples interact with the programs in the environment. Although this artifact exists in a few ITW hosts located in China, its prevalence is almost zero in the wild.

`PersonalBankPortal`, according to our research, is a program distributed by a Chinese bank to its customers. The samples that inject into this program belong to families such as Sality, Tinba and Ramnit, all considered as banking trojans that specifically exfiltrate sensitive banking data. We believe the sandbox vendor pre-installs this program essentially as a honeypot that lures malware samples into displaying their behaviors. Although this practice is useful for analyzing a sample, it causes artifacts that will not be observed in the wild, potentially hurting the performance.

Looking at the artifacts found in SB2, we see `Spotify`, a popular music streaming service, and `Python`, the interpreter for Python programming language. Both of these programs are

targeted and injected by file infectors, similar to `SogouExplorer` in SB1. These programs are not nearly as prevalent in the ITW hosts as they are in SB2. We believe the US-based vendor of SB2 pre-installs them to create an environment representative of the hosts in the US.

Overall, our case studies show that vendors configure their sandboxes according to their specifications, often based on the location of their hosts. This practice boosts the detection performance on the sandbox test sets by eliminating the effects of environment shift. However, as the real-world hosts do not follow these specifications, these strong signals observed in sandboxes fail to be representative of the behaviors in the wild.

Sandbox Evasion. Our previous discussion shows that sandbox configurations are predictable, which causes behavior artifacts. This also allows malware samples to easily fingerprint them (e.g., check if `SogouExplorer` and `PersonalBankPortal` exist together) to evade being analyzed [178]. Although it is generally difficult to detect sandbox evasion, a reasonable heuristic is using the number of actions performed, i.e., the sample might be evasive if it is too low [141]. Using this heuristic, we compare the number of average registry key creations in the ITW traces and the sandbox traces (SB1) of certain malware families. We focus on registry key creations as they could be recorded unambiguously, unlike actions such as process injections that might have vendor-specific definitions. For the Emotet family, the ITW traces and sandbox traces have an average of 4.6 vs. 3.4 actions, respectively. For the Loadmoney family, this is 7.2 vs. 1.1 and for Wannacry it is 5.3 vs 8.9. This shows that for certain families, there are significantly more actions in the ITW traces, indicating evasion. As another result of environment shift, we believe, this could particularly hurt the sandbox-only model's performance on the ITW traces for certain families.

4.6.2 Learning Environment-Invariant Features

In this section, we explore a solution to environment shift. Ideally, vendors would generate realistic sandbox configurations on-the-fly for each detonation to prevent the problems we discuss in the previous section. This, however, is known to be challenging and not always practical or even effective [179]. This motivates us to focus on developing an ML-based algorithmic solution.

Towards a solution, in Table 4.6, we first observe that the artifacts of SB1 are not present in SB2, and vice versa, as they are configured sufficiently differently. This means that a model can be made immune to these artifacts if it is trained to ignore the predictive features of a sample that exist only in its trace from one sandbox and not in the others. In other words, this model is *invariant* to the differences among the traces of the same sample.

In the ML community, invariant learning is widely studied in the context of self-supervised learning, e.g., viewpoint-invariant visual features [180]. Siamese loss is a very popular method to learn invariant features [181] that essentially feeds pairs of semantically related inputs to the model and forces it to produce similar embeddings (in terms of a distance metric) on each pair. In our case, each pair consists of two traces of the same sample from different sandboxes—(x_i^1, x_i^2). To implement this, we rely on the training traces in the SB1 $\cap$ SB2 set, which gives us around 9K pairs of traces. The resulting invariance loss function $\mathcal{L}_{INV}$, and our final loss function for training our model are defined as follows:

$$\mathcal{L}_{INV} = \frac{1}{|SB1 \cap SB2|} \sum_{(x_i^1, x_i^2) \in (SB1 \cap SB2)} \|enc(x_i^1) - enc(x_i^2)\|_2$$

$$\mathcal{L}_{Final} = (1-\alpha)\mathcal{L}_{ERM} + \alpha\mathcal{L}_{DRO} + \beta\mathcal{L}_{INV}$$

Here, β is a hyper-parameter to control the intensity of invariance loss: if it is too high, the model's embeddings might collapse into a single point. We tune β on our ITW validation set. In Figure 4.4, we present the results of training with $\mathcal{L}_{INV}$ in **`S-1`**, compared to the best model in Section 4.4. We improve the performance (TPR@ 1%FPR) of the base model (only $\mathcal{L}_{ERM}$) by 44%, from 13.6% to 19.6%, and the DRO model by 17%, from 16.8% to 19.6%. This hints that invariant learning is promising to deal with the environment shift problem.

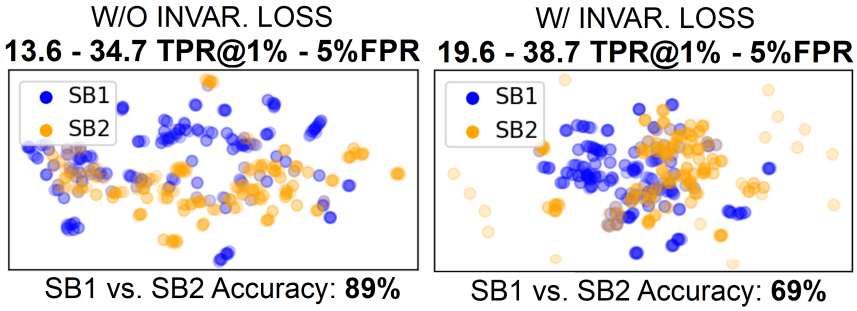

Figure 4.4: The effects of invariance loss over sandboxes on the model's embeddings. 2D visualization using t-SNE [182].

The Effect of Invariance on the Embeddings. One of the goals of $\mathcal{L}_{INV}$ is to make the traces from SB1 and SB2 indistinguishable from the model's perspective. In Figure 4.4, we visualize the embeddings of the model on the traces in SB1 $\cap$ SB2, before and after applying $\mathcal{L}_{INV}$. Moreover, we measure the success of a linear classifier in separating these embeddings as coming from SB1 or SB2. We see that before $\mathcal{L}_{INV}$ (*left*) SB1 and SB2 embeddings are easily separable and the linear model achieves 89% accuracy. After $\mathcal{L}_{INV}$ (*right*), however, SB1 and SB2 embeddings almost merge into a single distribution, and the linear model, achieving 69% accuracy, struggles to separate them. This shows that the invariance loss is effective at making the model more robust to environment shift.

4.6.3 Environment Shift in the Wild

So far, we have focused on a no-delay scenario where a decision on a sample is made using its very first ITW trace—$x^0_{i,0}$. However, research shows that a sample's behaviors might vary from one ITW host to another [139], indicating environment shift. To study how this impacts the model, we define $S_{i,h}$ as the set of a model's prediction scores on the ITW traces of a sample P_i within the first h hours after P_i was first seen—$S_{i,h} = \{f(x_{i,j}) \mid x_{i,j} \in \mathbf{x}^0_{i,\,(t<h)}\}$.

In Figure 4.5, we present a histogram on the standard deviations of $S_{i,\,h=24}$ (σ_i) for the samples in the ITW test set. For over 80% of the malware and 60% of the benign samples $\sigma_i > 0$, meaning that environment shift among ITW hosts has an impact for these samples. Moreover, σ_i for an average malware sample is higher than for a benign sample, aligning with prior findings that malware samples behave more variably across hosts [139]. In Section 4.7.2, we explore improving the detection performance by using an aggregation function that takes $S_{i,h}$ as its input and outputs a single score that will be used to make a decision on P_i.

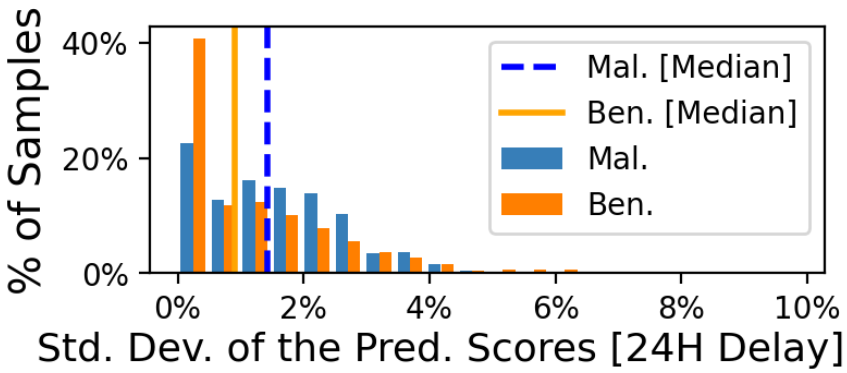

Figure 4.5: The histogram over the standard deviations of the scores on the traces of a sample observed within 24 hours.

Takeaways. The configuration of a sandbox, such as its pre-installed programs, might result

in very predictive features of maliciousness that are in fact artifacts and not present in other environments. We propose an ML method for learning invariant features over multiple traces of a sample to make a model more robust to such artifacts and boost its performance in the wild.

4.7 Going Beyond Sandbox-Only Training

In the previous sections, we have focused on **`S-1`** (detailed in Section 4.2.1) where the vendor has to train a model on a sandbox-only training set that will be deployed on the ITW traces. Our practical ML improvements have boosted the ITW performance of a standard model in this scenario from 13.6% TPR@1%FPR to 19.6%. Considering that the samples in the ITW dataset represent the most extreme cases where all defenses of the anti-malware vendor failed (effectively achieving 0% TPR), these improvements are substantial. In this section, our goal is to explore less restricted scenarios (**`S-2`** and **`S-3`**) to pursue the best ITW performance we can achieve.

4.7.1 **`S-2`**: Training on Labeled ITW Traces

In this section, we explore training our model directly on ITW traces collected during the training phase (i.e., the ITW samples in the training chunk in Table 4.2). Here, we assume that the vendor allocates resources to label the samples on which they have ITW traces sent by hosts. We believe this is feasible as similar tasks are frequently performed by malware analysts who label samples using dynamic analysis [158]. We evaluate the models in this scenario along three dimensions: **(i)** the number of ITW samples for training; **(ii)** the number of traces per training sample; and **(iii)** training only on the ITW traces from scratch vs. fine-tuning a sandbox-only model.

We share the results in Figure 4.6. First, below 10% of the ITW data (for TPR @1%FPR) for

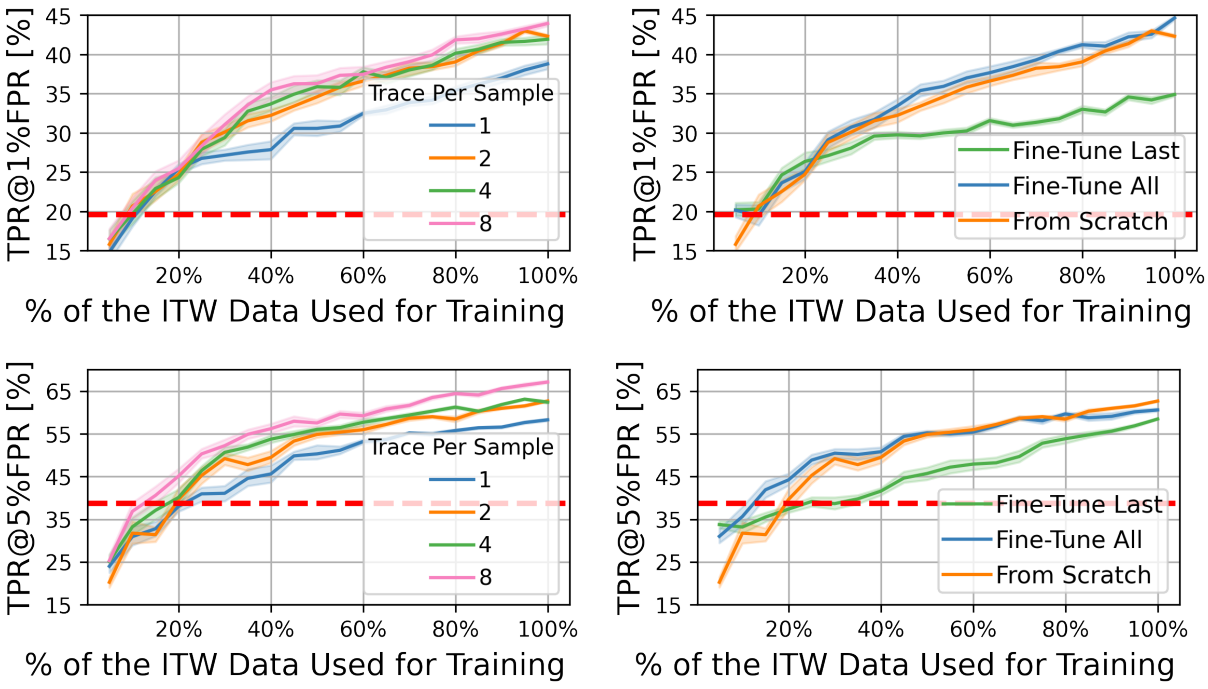

Figure 4.6: The performance of models in **`S-2`** for TPR @1% FPR (*top row*) and for TPR @5%FPR (*bottom row*). The impact of the number of traces per training sample (*left*), and comparing training from scratch vs. fine-tuning the sandbox-only model (*right*). The dashed line is for the best model in **`S-1`**.

training (which corresponds to around 1500 benign and 45 malware samples), our sandbox-only model in Section 4.6 performs better and after the 10% threshold the benefits of training on ITW traces emerge. For TPR @5%FPR this threshold is at 20%. Second, using multiple traces per training sample is effective going from 1 trace per sample to 2, however, it gives diminishing returns after that. As we show in Section 4.6.3, the model treats different traces of the same sample differently. In consequence, including more traces per sample for training provides better coverage of the behaviors in the wild. If the number of total training traces is limited, it is better to train on more samples with 1 trace per sample than on fewer samples with more traces per sample.

Fine-Tuning vs Training From Scratch. Here, in Figure 4.6 (*right*), we select two traces per sample and implement two fine-tuning strategies: **(i)** freezing the encoder layer enc and tuning only the classification layer g, and **(ii)** tuning all layers without freezing. We see that, in low-data regimes (below 20% of the ITW data), fine-tuning has an edge over training from scratch. Moreover, in this regime, fine-tuning only g performs slightly better than fine-tuning all layers, however, with more data it starts to perform significantly worse, due to being less flexible in learning from the ITW traces. Notably, fine-tuning all layers performs better than training from scratch with 1-3% higher TPR@1%FPR. These results demonstrate the benefits of training a well-performing model in a sandbox-only setting and fine-tuning it on the ITW data, especially when the ITW data is scarce; further motivating our efforts in Section 4.5 and Section 4.6.

Overall, in **`S-2`**, we have achieved $\sim$46% TPR @1%FPR, a significant jump over our best model in **`S-1`** (20% TPR). The best model in **`S-2`** was trained on $\sim$15.5K ITW samples (15K benign and 500 malware), collected between January and April 2018. Around 25% of these samples could have been labeled on VirusTotal by April 2018, leaving around 12K samples for the vendor to label. This means that by labeling under 135 undetected samples a day using their traces (in the training phase), an anti-malware vendor could have achieved over double the performance of a sandbox-only model on the samples that they could not detect (in the testing phase). We believe this is practical for companies and highlights the benefits of focusing on ITW traces over sandbox traces for training detectors.

4.7.2 **S-3**: Sandbox Detonation of ITW Samples

Our efforts so far have been on using ITW traces to make decisions on the samples seen in the wild that the vendor could not detect. This would enable making instant decisions on a sample right after a host finishes executing this sample. In this section, we explore an alternative: in the testing phase, the vendor detonates the undetected samples hosts encounter in a sandbox and uses the resulting trace to make a decision on this sample. To simulate this scenario, we make use of the overlapping samples between our SB1 and ITW test sets (SB1 $\cap$ ITW in Table 4.2). Note that this is only a small subset of the samples in our ITW test set as we could not find the sandbox traces for most of them. This gives us 48 malware and 1118 benign samples for measuring the performance.

The best model on the SB1 validation set in **S-0** (from Section 4.4) achieves 43%TPR @1%FPR on the SB1 traces of these samples. On the other hand, the best model on the ITW validation set in **S-2** (from Section 4.7.1) achieves 39% on the first-seen ITW traces of these samples. This immediately suggests that the performance could be improved if the vendor actually avoids making decisions on the ITW traces altogether and relies only on sandbox traces.

The Impact of Decision Delays. Although, in terms of TPR, **S-3** seems to be superior to **S-2**, this does not consider the fact that detection in **S-3** will incur an additional delay that stems from sandbox detonation, which we call the *sandbox delay*. On the other hand, it is possible to make an instant decision in **S-1** and **S-2**. As we discuss in Section 4.2.2, decision delays impact the success of a detector in terms of preventing real-world infections. To this end, on top of TPR @1% FPR, we compare **S-2** and **S-3** in terms of PIR @1%IAR (defined in Section 4.2.2), which quantifies the detector's success in preventing infections, at different levels of delay. We present

this comparison in Figure 4.7. First, note how the TPR in **S−2** increases over time (39% when $h=0$, 41% when $h=6$, and 45% when $h=21$). This is because, for an undetected sample P_i, the vendor receives multiple traces over h hours and ends up with a set of prediction scores ($S_{i,h}$ in Section 4.6.3), one for each trace. We simply take the average over $S_{i,h}$ to obtain a score that we use to make a decision on P_i. This boosts the performance by dampening the impact of behavior variability of a sample across different hosts in the wild [139]. On the other hand, the TPR in **S−3** stays fixed as it relies on a single trace from a sandbox.

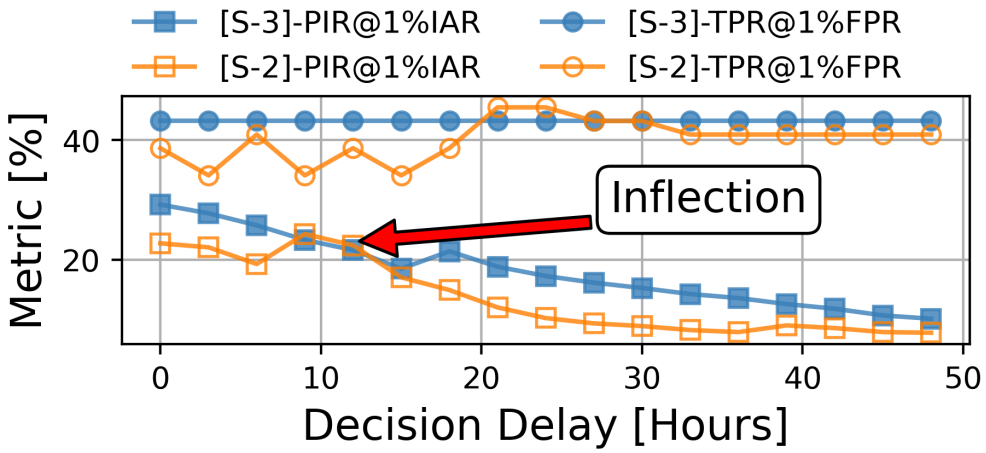

Figure 4.7: Comparing the detection performances (PIR and TPR) in **S−2** and **S−3** with respect to decision delay.

More importantly, it is possible to make an instant decision in **S−2** as it relies on traces sent by the hosts (the PIR is 23% when $h=0$). Assuming this was possible in **S−3** (no sandbox delay), the PIR in **S−3** would be 29%, surpassing the performance in **S−2**. However, this is not practical and sandbox execution will introduce some delay [158]. If this delay is over 12 hours, the PIR in **S−3** drops below 23%. This makes $h=12$ an *inflection* point: any sandbox delay over it makes **S−2** more attractive for detecting the samples in the wild. Overall this highlights how malware detection in the wild is a race against time, unlike conventional sandbox-only scenarios. It is

critical to consider the impact of decision delays by using metrics such as PIR when evaluating different approaches by finding the inflection points in their comparative performance curves.

Estimating the Real-World Sandbox Delay. Here, we perform a case study on SB1 and measure the time gap between when a sample was first seen on VirusTotal and when it was executed in SB1. We present a histogram of these gaps over all SB1 test samples in Figure 4.8. The average delay for malware and benign samples is 33 hours and 23 hours, respectively; significantly higher than the inflection point that puts **`S-2`** over **`S-3`**. Note that this is a rough estimate as we do not have any visibility into how the vendor of SB1 operates, i.e., there could be a delay between when a sample is seen on VirusTotal and when the vendor can access it.

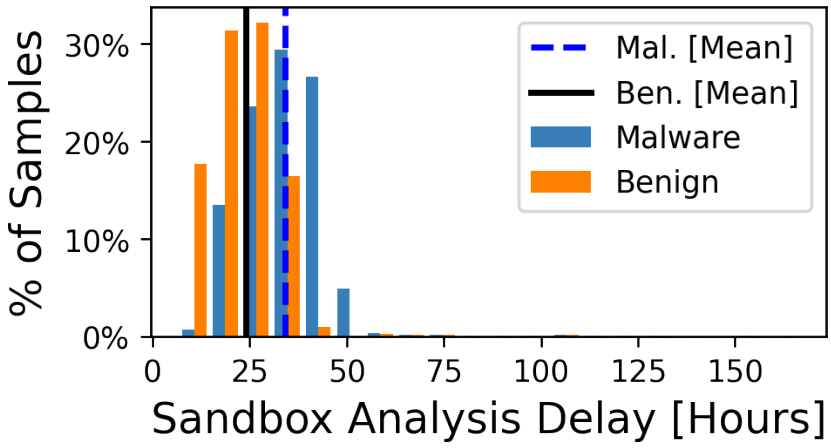

Figure 4.8: Estimating sandbox detection delay through a case study on a real-world vendor.

Takeaways. It is critical to see beyond standard success metrics when comparing approaches. For example, training on ITW traces is effective but it places a burden on the vendor to label the undetected samples seen in the wild; or using the sandbox traces of these samples might outperform using their ITW traces if detonation does not cause delays over 12 hours.

4.8 Discussion

The Impact of Environmental Factors on ITW Performance. It is known that programs, especially malware, behave differently depending on environment factors such as the host's operating system (OS) [139] or its geographical location [183]. For example, a specific strain of banking malware only targets the hosts in Brazil and it stays idle in other countries [184]. To this end, vendors often configure their sandboxes specifically to observe such environment-specific behaviors. To understand the impact of these factors, we measure the model's performance (in **`S-1`** and **`S-2`**) on different subsets of ITW traces, selected based on the metadata of the hosts from which they are collected. In particular, we focus on the impact of the operating system version and the geographical location of the host. Note that we might not have a trace for every program in our ITW dataset for each metadata type (e.g., we only have Windows 8.1 traces for only 80 malware samples out of 398). We present the results in Table 4.7. We see that the model in **`S-1`** performs noticeably worse on Windows 10 traces (that cover 39% of our dataset) and better on Windows 7 traces (that cover 51% of our dataset). We believe this is because the training sandbox (SB1) might be configured with Windows 7 and, as a result, the traces collected from it are more representative of the Windows 7 traces in the wild. Moreover, on traces from Russia (covers 5%) and Germany (covers 5%), the models perform significantly better than on traces from China (covers 14%) and Japan (covers 7%). We believe this might stem from the type of malware specifically targeting different countries and how easy it is to detect them. For example, Russian hosts are pre-dominantly infected by Loadmoney (installs unwanted software after exploiting a vulnerability) and Gandcrab (ransomware) families; whereas, Chinese hosts are infected by

Chindo (which is a generic name for Chinese software bundler-type malware). Overall, these results indicate that environmental factors do have implications for malware detection in the wild and, also, sandbox configurations. We believe a promising direction is using an invariance loss function (similar to the one in Section 4.6.2) towards models more robust to such factors, e.g., a model that is invariant to the differences between the traces of a sample collected from the US and China.

Model / Env.	Data Stats		Scenario 1		Scenario 2	
	Cov.	#Mal.	1%FPR	5%FPR	1%FPR	5%FPR
Operating System of the Host						
Win. 7	51%	346	22.0%	39.9%	46.8%	66.8%
Win. 8.1	5%	80	11.2%	17.5%	38.8%	55.0%
Win. 10	39%	171	7.6%	21.6%	35.1%	50.3%
All	100%	398	19.6%	39.2%	46.2%	68.1%
Geographical Location of the Host						
USA	38%	252	23.8%	42.9%	46.8%	70.2%
China	14%	91	8.8%	27.5%	29.7%	56.0%
Japan	7%	32	18.8%	21.9%	34.4%	37.5%
Germany	5%	28	21.4%	35.7%	57.1%	67.9%
Russia	5%	87	28.2%	56.3%	59.8%	79.3%
All	100%	398	19.6%	39.2%	46.2%	68.1%

Table 4.7: Impact of environmental factors.

PUP Detection in the Wild. Aside from malware and benign, there is a third class of samples, potentially unwanted programs (PUPs), that users agree to download although they offer few or no benefits and can serve as adware or spyware. PUP detection is often studied separately from malware detection [185]. In our ITW dataset, we have around 5K samples tagged as PUP by AVClass2 [163], and following the same scenarios in Section 4.2.1, we ask whether PUP detection also suffers from the same problems as malware detection. We present the detection results in Table 4.8. We see that, even in **`S-0`**, the performance is overall worse than the performance we have seen for malware detection, indicating that detecting PUP might be more challenging as PUP samples are closer to the boundary. To see if the sample shift problem also exists in PUP detection, we present a comparison of the PUP family distributions in Figure 4.9. Here, we see that a few PUP families, such as InstallCore (a family of software bundlers) or DealPly (adware that targets Windows Browsers) have high representation (over 50%) in both our sandbox and ITW datasets. Overall, the patterns we have observed in malware detection (e.g., the gap between **`S-0`** and **`S-1`** and sample shift), also exist in PUP detection, perhaps to a lesser extent.

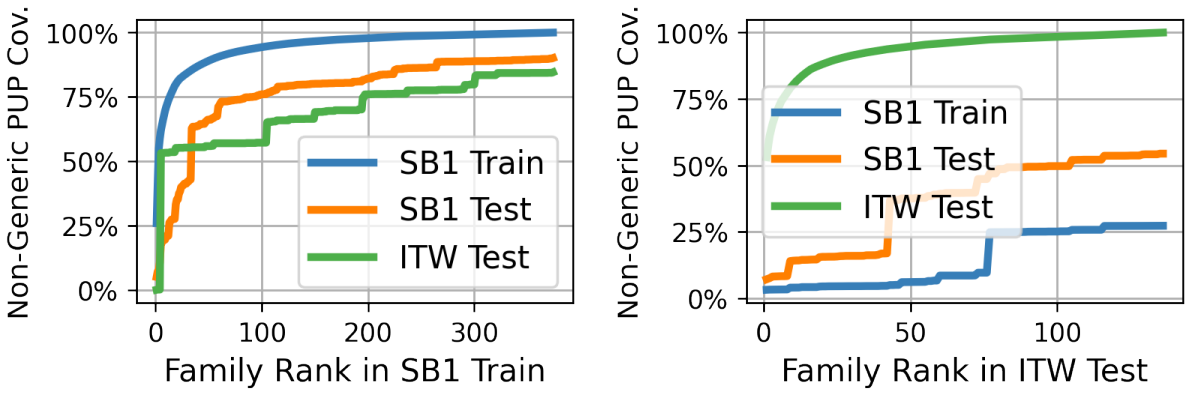

Figure 4.9: Quantifying the extent of sample shift in PUP families between sandbox and ITW datasets.

Subset \ Test	Sandbox 1 Test			In-the-Wild Test		
	Cov.	1%FPR	5%FPR	Cov.	1%FPR	5%FPR
PUP Family Subsets						
Top-Train	68%	78.9%	92.2%	54%	24.8%	47.9%
Top-ITW	14%	57.3%	96.3%	86%	18.7%	39.2%
Unseen-ITW	2%	38.6%	62.4%	15%	3.2%	16.0%
Generic	11%	77.5%	82.0%	5%	8.6%	32.0%
All Mal.	100%	72.0%	86.1%	100%	16.8%	36.5%

Table 4.8: Impact of family distribution shift for PUP detection in the wild.

Real-Time Malware Detection. In our study, a detector uses the whole execution trace of a sample that terminates naturally in a host. A strong capability, offered by most anti-malware vendors [128], is catching a malware sample on its tracks *before* it terminates. The benefit of these offerings is containing the harm of a malware sample by cutting its execution short, e.g., before a ransomware starts encrypting personal files. As this is a strictly more difficult task than detecting malware using its full trace, our results can be viewed as an upper-bound on the performance of real-time malware detection. In consequence, we do not expect a real-time detector that is trained on sandbox traces to perform well in the wild.

4.9 Takeaways and Conclusions

Malware detection with dynamic features serves as a last line of defense, providing security when all other measures have failed. When this last line is also breached, real-world malware infections

happen and cause significant harm. The standard practices in this domain, e.g., training detectors on sandbox executions, break down when deployed in the real world, which we have attributed to distribution shift, Without having the ability to perturb the inputs explicitly, the adversary gains a natural leverage, i.e., an advantage, against defenders in this setting. This highlights that adversarial pressure on machine learning models can exist naturally and in ways that cannot be captured by the conventional adversarial examples threat model. A long-standing ambition in the security community is to find the best way to detect malware in the wild to prevent infections. Our work in this chapter provides some clarity through a systematic exploration of various scenarios. Training directly on real-world behaviors is promising, however, it might create a labeling burden on the defender. This makes relying exclusively on sandbox behaviors attractive, though, the delay introduced by sandbox executions might mean losing the race against time in preventing infections in the wild. Our work suggests that there is no silver bullet and this challenging problem requires collaboration between the industry and research, for example, through data sharing. We have also provided new metrics, taxonomies, and insights to aid the pursuit of an optimal approach.

Chapter 5: Conclusion

This dissertation has explored how the threat models in traditional cyber security research can manifest themselves in the rapidly developing field of machine learning. The research into vulnerabilities of machine learning has converged into a few dominant threat models, such as adversarial examples, which is mainly studied in the machine learning community. In our work, we have studied three realistic threats that originated in security literature in the context of the three dimensions of the conventional adversarial examples threat model.

In Chapter 2, we have focused on the goals of an adversary against machine learning. Conventionally, adversarial machine learning research has been focusing mainly on causing misclassifications as a goal, however, our work shows that this fails to capture the lessons from the security literature. In particular, security literature suggests that adversaries often target the availability of a victim system using denial-of-service or algorithmic complexity attacks. We have shown that a new advancement, i.e., input-adaptive architectures, makes this classical threat viable against machine learning. Our work is one of the first to recognize this risk and develop the tools, including new attacks and metrics, to study it. Our most alarming finding is that adversarial training, the de facto standard defense to conventional attacks, provides limited robustness to this emerging threat. Moreover, adversarial training we tailor specifically to defeat slowdown attacks has become ineffective against conventional attacks. Identifying this inherent tension between

these two robustness notions has guided the follow-up work to develop novel ad-hoc solutions to slowdown attacks.

In Chapter 3, we have focused on the constraints of an adversary who performs an adversarial attack against machine learning. The conventional threat model constraints the adversary in terms of the amount of perturbation the attacks can introduce to the inputs, e.g., in terms of ℓ_∞-norm. This, however, leads to inherently detectable attacks that has created an arms-race between detection defenses and attack algorithms. In security literature, mimicry attacks are optimized for both success, e.g., compromising a network, and avoiding an anomaly detector, e.g., a network intrusion detection system. As an analogue of these attacks in machine learning, we have developed statistical indistinguishability attack (SIA) that provides an intuitive slider that balances between two objectives: attack success and evasiveness. We have showcased SIA as an effective benchmark by applying it, with no customization, to break a wide range of published detectors. We also have identified the properties of detectors that force SIA to compromise between its objectives, *i.e.*, successful but detectable attacks vs.undetectable but not-so-successful attacks. These properties highlight promising research directions for more robust detectors.

In Chapter 4, we have focused on the leverage of an adversary who performs an attack against machine learning. The conventional threat model assumes that the adversary can introduce minor perturbations to the inputs to gain leverage over the model, e.g., to force it to misclassify the inputs. We have studied a domain, i.e., malware detection with dynamic features, where the adversary gains leverage more naturally, without perturbing the inputs explicitly. We have shown that the performance of a malware detector that is trained and evaluated in a controlled, non-adversarial setting rapidly degrades when it is evaluated in a real-world setting where the adversary's leverage exists. In particular, the distribution shift from the controlled setting to the

real world, i.e., in-the-wild, setting deteriorates the model and allows costly malware infections to happen. We have explored various machine learning approaches to address this distribution shift for boosting malware detectors, achieving moderate performance gains. Overall, our investigation in this chapter reveals that the conventional methods designed for making a model robust are not as effective when they are applied to a domain where adversarial pressure exists naturally in complex ways that are outside the simplified threat models studied in machine learning. This also highlights that to investigate the realistic limitations of machine learning, it is critical to evaluate it in classical security settings, such as malware detection, where adversaries and their incentives are well documented.

Given all this, we can conclude that although adversarial machine learning research has converged into a few threat models that are being treated formally, security literature suggests that adversaries do not follow our expectations as defenders. Looking into classical threat models has led to attacks outside these conventional threat models that cannot be easily defended by existing defenses. We believe that the problem is rooted in the fact that the machine learning community measures progress based mainly on standardized benchmarks, e.g., model robustness. Researchers have an incentive to focus on conventional threat models to make progress on known problems, rather than asking what type of goals, constraints, and capabilities adversaries might have in the real world. However, this practice, as we have shown in our dissertation, leads to a discrepancy between the hard-earned lessons from the security literature and current research trends.

To alleviate this discrepancy, we hope that the work we have presented in this dissertation can reinforce the importance of considering comprehensive threat models in machine learning for properly assessing its security risks. Moreover, in addition to standard machine learning domains, such as image and language, we believe it is paramount to bring advances in adversarial

machine learning to the applications of machine learning for security, such as malware detection. These applications have real, well-documented adversaries that actively look for ways to break models. Currently, existing defenses, such as adversarial training, offer limited success in real-world settings as they have been primarily developed on standardized benchmarks. Our future outlook is stepping away from benchmarks and focusing on adapting modern defensive solutions to real-world security tasks to allow for machine learning's benefits to take full effect.

Bibliography

[1] Kaspersky. Machine learning for malware detection, 2021.

[2] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR 2014*, 2014.

[3] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

[4] Anne Adams and Martina Angela Sasse. Users are not the enemy. *Commun. ACM*, 42(12):40–46, dec 1999.

[5] Prahlad Fogla, Monirul I Sharif, Roberto Perdisci, Oleg M Kolesnikov, and Wenke Lee. Polymorphic blending attacks. In *USENIX security symposium*, pages 241–256, 2006.

[6] David Wagner and Paolo Soto. Mimicry attacks on host-based intrusion detection systems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 255–264, 2002.

[7] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[9] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[11] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[12] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Weinberger. Multi-scale dense networks for resource efficient image classification. In *International Conference on Learning Representations*, 2018.

[13] S. Teerapittayanon, B. McDanel, and H. T. Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469, 2016.

[14] Yiğitcan Kaya, Sanghyun Hong, and Tudor Dumitraş. Shallow-Deep Networks: Understanding and mitigating network overthinking. In *Proceedings of the 2019 International Conference on Machine Learning (ICML)*, Long Beach, CA, Jun 2019.

[15] Ting-Kuei Hu, Tianlong Chen, Haotao Wang, and Zhangyang Wang. Triple wins: Boosting accuracy, robustness and efficiency together by enabling input-adaptive inference. In *International Conference on Learning Representations*, 2020.

[16] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *CoRR*, abs/1608.08710, 2016.

[17] Ron Banner, Itay Hubara, Elad Hoffer, and Daniel Soudry. Scalable methods for 8-bit training of neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 5145–5153. Curran Associates, Inc., 2018.

[18] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[19] Ben Taylor, Vicent Sanz Marco, Willy Wolff, Yehia Elkhatib, and Zheng Wang. Adaptive deep learning model selection on embedded systems. *ACM SIGPLAN Notices*, 53(6):31–43, 2018.

[20] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE, 2009.

[21] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[23] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[24] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[25] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[26] Surat Teerapittayanon, Bradley McDanel, and HT Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *Pattern Recognition (ICPR), 2016 23rd International Conference on*, pages 2464–2469. IEEE, 2016.

[27] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q Weinberger. Multi-scale dense convolutional networks for efficient prediction. *CoRR, abs/1703.09844*, 2, 2017.

[28] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

[29] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

[30] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 372–387, 2016.

[31] Elias De Coninck, Tim Verbelen, Bert Vankeirsbilck, Steven Bohez, Pieter Simoens, Piet Demeester, and Bart Dhoedt. Distributed neural networks for internet of things: The big-little approach. In *International Internet of Things Summit*, pages 484–492. Springer, 2015.

[32] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519, 2017.

[33] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57, 2017.

[34] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

[35] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 274–283, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[36] Florian Tramèr, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. The space of transferable adversarial examples. *arXiv preprint arXiv:1704.03453*, 2017.

[37] Nathan Inkawhich, Wei Wen, Hai (Helen) Li, and Yiran Chen. Feature space perturbations yield more transferable adversarial examples. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[38] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[39] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*, 2017.

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

[41] Fangzhou Liao, Ming Liang, Yinpeng Dong, Tianyu Pang, Xiaolin Hu, and Jun Zhu. Defense against adversarial attacks using high-level representation guided denoiser. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[42] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana. Certified robustness to adversarial examples with differential privacy. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 656–672, 2019.

[43] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. SkipNet: Learning Dynamic Routing in Convolutional Networks. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[44] Michael Figurnov, Maxwell D. Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry Vetrov, and Ruslan Salakhutdinov. Spatially Adaptive Computation Time for Residual Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[45] Mirazul Haque, Anki Chauhan, Cong Liu, and Wei Yang. Ilfo: Adversarial attack on adaptive neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[46] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition, 2014.

[47] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, 2017.

[48] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[49] Florian Tramèr and Dan Boneh. Adversarial training and robustness for multiple perturbations. In *Advances in Neural Information Processing Systems*, pages 5866–5876, 2019.

[50] Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. Ead: elastic-net attacks to deep neural networks via adversarial examples. *arXiv preprint arXiv:1709.04114*, 2017.

[51] Yuzhe Yang, Guo Zhang, Dina Katabi, and Zhi Xu. Me-net: Towards effective adversarial robustness with matrix estimation. *arXiv preprint arXiv:1905.11971*, 2019.

[52] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[53] Jérôme Rony, Luiz G Hafemann, Luiz S Oliveira, Ismail Ben Ayed, Robert Sabourin, and Eric Granger. Decoupling direction and norm for efficient gradient-based l2 adversarial attacks and defenses. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4322–4330, 2019.

[54] Se Jin Park, Murali Subramaniyam, Seoung Eun Kim, Seunghee Hong, Joo Hyeong Lee, Chan Min Jo, and Youngseob Seo. Development of the elderly healthcare monitoring system with iot. In *Advances in Human Factors and Ergonomics in Healthcare*, pages 309–315. Springer, 2017.

[55] Jiasi Chen and Xukan Ran. Deep learning with edge computing: A review. *Proceedings of the IEEE*, 107(8):1655–1674, 2019.

[56] Florian Tramèr, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. The space of transferable adversarial examples. *arXiv preprint arXiv:1704.03453*, 2017.

[57] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[58] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[59] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS'17, page 615–629, New York, NY, USA, 2017. Association for Computing Machinery.

[60] En Li, Zhi Zhou, and Xu Chen. Edge intelligence: On-demand deep learning model co-inference with device-edge synergy. In *Proceedings of the 2018 Workshop on Mobile Edge Communications*, MECOMM'18, page 31–36, New York, NY, USA, 2018. Association for Computing Machinery.

[61] Li Zhou, Hao Wen, Radu Teodorescu, and David H.C. Du. Distributing deep neural networks with containerized partitions at the edge. In *2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)*, Renton, WA, July 2019. USENIX Association.

[62] Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. BERT Loses Patience: Fast and Robust Inference with Early Exit. In *Advances in Neural Information Processing Systems*, 2020.

[63] Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. DynaBERT: Dynamic BERT with Adaptive Width and Depth. In *Advances in Neural Information Processing Systems*, 2020.

[64] C. Hu, W. Bao, D. Wang, and F. Liu. Dynamic adaptive dnn surgery for inference acceleration on the edge. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 1423–1431, 2019.

[65] Weiwen Jiang, Edwin H.-M. Sha, Xinyi Zhang, Lei Yang, Qingfeng Zhuge, Yiyu Shi, and Jingtong Hu. Achieving super-linear speedup across multi-fpga for real-time dnn inference. *ACM Trans. Embed. Comput. Syst.*, 18(5s), October 2019.

[66] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM workshop on artificial intelligence and security*, pages 3–14, 2017.

[67] Kevin Roth, Yannic Kilcher, and Thomas Hofmann. The odds are odd: A statistical test for detecting adversarial examples. In *International Conference on Machine Learning*, pages 5498–5507. PMLR, 2019.

[68] Jayaram Raghuram, Varun Chandrasekaran, Somesh Jha, and Suman Banerjee. A general framework for detecting anomalous inputs to DNN classifiers. In *International Conference on Machine Learning*, pages 8764–8775. PMLR, 2021.

[69] Florian Tramer, Nicholas Carlini, Wieland Brendel, and Aleksander Madry. On adaptive attacks to adversarial example defenses. *Advances in Neural Information Processing Systems*, 33, 2020.

[70] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[71] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[72] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[73] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer networks*, 31(23-24):2435–2463, 1999.

[74] Christina Warrender, Stephanie Forrest, and Barak Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *Proceedings of the 1999 IEEE symposium on security and privacy (Cat. No. 99CB36344)*, pages 133–145. IEEE, 1999.

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

[76] Maksym Andriushchenko, Francesco Croce, Nicolas Flammarion, and Matthias Hein. Square attack: A query-efficient black-box adversarial attack via random search. In *European Conference on Computer Vision*, pages 484–501. Springer, 2020.

[77] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations. In *Proceedings of 5th International Conference on Learning Representations (ICLR)*, 2017.

[78] Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410*, 2017.

[79] Ali Shafahi, W Ronny Huang, Christoph Studer, Soheil Feizi, and Tom Goldstein. Are adversarial examples inevitable? In *International Conference on Learning Representations*, 2018.

[80] Leslie Rice, Eric Wong, and Zico Kolter. Overfitting in adversarially robust deep learning. In *International Conference on Machine Learning*, pages 8093–8104. PMLR, 2020.

[81] Xingjun Ma, Bo Li, Yisen Wang, Sarah M Erfani, Sudanthi Wijewickrema, Grant Schoenebeck, Dawn Song, Michael E Houle, and James Bailey. Characterizing adversarial subspaces using local intrinsic dimensionality. In *International Conference on Learning Representations*, 2018.

[82] Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. *Advances in neural information processing systems*, 31, 2018.

[83] Zhihao Zheng and Pengyu Hong. Robust detection of adversarial attacks by modeling the intrinsic properties of deep neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 7924–7933, 2018.

[84] David Miller, Yujia Wang, and George Kesidis. When not to classify: Anomaly detection of attacks (ADA) on DNN classifiers at test time. *Neural computation*, 31(8):1624–1670, 2019.

[85] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick McDaniel. On the (statistical) detection of adversarial examples. *arXiv preprint arXiv:1702.06280*, 2017.

[86] Ruize Gao, Feng Liu, Jingfeng Zhang, Bo Han, Tongliang Liu, Gang Niu, and Masashi Sugiyama. Maximum mean discrepancy test is aware of adversarial attacks. In *Proceedings of the 38th International Conference on Machine Learning*, pages 3564–3575, 2021.

[87] Arthur Gretton, Karsten Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alex Smola. A kernel method for the two-sample-problem. *Advances in neural information processing systems*, 19:513–520, 2006.

[88] Kacper P Chwialkowski, Dino Sejdinovic, and Arthur Gretton. A wild bootstrap for degenerate kernel tests. In *Advances in neural information processing systems*, pages 3608–3616, 2014.

[89] Feng Liu, Wenkai Xu, Jie Lu, Guangquan Zhang, Arthur Gretton, and Danica J Sutherland. Learning deep kernels for non-parametric two-sample tests. In *International Conference on Machine Learning*, pages 6316–6326. PMLR, 2020.

[90] Jonas Kübler, Wittawat Jitkrittum, Bernhard Schölkopf, and Krikamol Muandet. Learning kernel tests without data splitting. *Advances in Neural Information Processing Systems*, 33, 2020.

[91] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Großberger. UMAP: Uniform manifold approximation and projection. *Journal of Open Source Software*, 3(29), 2018.

[92] Jérôme Rony, Luiz G Hafemann, Luiz S Oliveira, Ismail Ben Ayed, Robert Sabourin, and Eric Granger. Decoupling direction and norm for efficient gradient-based l2 adversarial attacks and defenses. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4322–4330, 2019.

[93] Florian Tramer and Dan Boneh. Adversarial training and robustness for multiple perturbations. *Advances in Neural Information Processing Systems*, 32:5866–5876, 2019.

[94] Dougal J Sutherland, Hsiao-Yu Tung, Heiko Strathmann, Soumyajit De, Aaditya Ramdas, Alexander J Smola, and Arthur Gretton. Generative models and model criticism via optimized maximum mean discrepancy. In *ICLR (Poster)*, 2017.

[95] Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. Shallow-deep networks: Understanding and mitigating network overthinking. In *International Conference on Machine Learning*, pages 3301–3310. PMLR, 2019.

[96] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International Conference on Machine Learning*, pages 3519–3529. PMLR, 2019.

[97] Sara Sabour, Yanshuai Cao, Fartash Faghri, and David J. Fleet. Adversarial manipulation of deep representations. In *ICLR (Poster)*, 2016.

[98] Shiqing Ma and Yingqi Liu. NIC: Detecting adversarial samples with neural network invariant checking. In *Proceedings of the 26th Network and Distributed System Security Symposium (NDSS 2019)*, 2019.

[99] Ronald Aylmer Fisher et al. 224a: Answer to question 14 on combining independent tests of significance. 1948.

[100] Morton B Brown. 400: A method for combining non-independent, one-sided tests of significance. *Biometrics*, pages 987–992, 1975.

[101] Yogesh Balaji, Tom Goldstein, and Judy Hoffman. Instance adaptive adversarial training: Improved accuracy tradeoffs in neural nets. *arXiv preprint arXiv:1910.08051*, 2019.

[102] Xuwang Yin, Soheil Kolouri, and Gustavo K Rohde. GAT: Generative adversarial training for adversarial example detection and robust classification. In *International Conference on Learning Representations*, 2020.

[103] Florian Tramèr. Detecting adversarial examples is (nearly) as hard as classifying them. In *ICML 2021 Workshop on the Prospects and Perils of Adversarial Machine Learning*, 2021.

[104] Florian Tramèr, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. The space of transferable adversarial examples. *arXiv*, 2017.

[105] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. In *5th International Conference on Learning Representations*, 2017.

[106] Nathan Inkawhich, Wei Wen, Hai Helen Li, and Yiran Chen. Feature space perturbations yield more transferable adversarial examples. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7066–7074, 2019.

[107] Dongxian Wu, Yisen Wang, Shu-Tao Xia, James Bailey, and Xingjun Ma. Skip connections matter: On the transferability of adversarial examples generated with ResNets. In *International Conference on Learning Representations*, 2020.

[108] David Lopez-Paz and Maxime Oquab. Revisiting classifier two-sample tests. In *International Conference on Learning Representations*, 2017.

[109] Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. In *International Conference on Machine Learning*, pages 854–863. PMLR, 2017.

[110] Yao-Yuan Yang, Cyrus Rashtchian, Hongyang Zhang, Russ R Salakhutdinov, and Kamalika Chaudhuri. A closer look at accuracy vs. robustness. *Advances in neural information processing systems*, 33:8588–8601, 2020.

[111] Aladin Virmaux and Kevin Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. *Advances in Neural Information Processing Systems*, 31, 2018.

[112] Guanhong Tao, Yingqi Liu, Guangyu Shen, Qiuling Xu, Shengwei An, Zhuo Zhang, and Xiangyu Zhang. Model orthogonalization: Class distance hardening in neural networks for better security. In *2022 IEEE Symposium on Security and Privacy (SP). IEEE*, volume 3, 2022.

[113] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *International Conference on Learning Representations*, 2018.

[114] Stephan Rabanser, Stephan Günnemann, and Zachary C Lipton. Failing loudly: An empirical study of methods for detecting dataset shift. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 1396–1408, 2019.

[115] Matthias Kirchler, Shahryar Khorasani, Marius Kloft, and Christoph Lippert. Two-sample testing using deep learning. In *International Conference on Artificial Intelligence and Statistics*, pages 1387–1398. PMLR, 2020.

[116] S. Maji, J. Kannala, E. Rahtu, M. Blaschko, and A. Vedaldi. Fine-grained visual classification of aircraft. Technical report, 2013.

[117] Yiyou Sun, Chuan Guo, and Yixuan Li. React: Out-of-distribution detection with rectified activations. *Advances in Neural Information Processing Systems*, 34, 2021.

[118] Weitang Liu, Xiaoyun Wang, John Owens, and Yixuan Li. Energy-based out-of-distribution detection. *Advances in Neural Information Processing Systems*, 33, 2020.

[119] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[120] James Reddick. Us treasury: Financial institutions reported $1.2 billion in ransomware losses in 2021, Nov 2022.

[121] IMARC Group. Malware analysis market: Global industry trends, share, size, growth, opportunity and forecast 2023-2028, 2022.

[122] AV-Comparatives. Malware protection test march 2023, 2023.

[123] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K Nicholas. Malware detection by eating a whole EXE. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[124] ElMouatez Billah Karbab and Mourad Debbabi. Maldy: Portable, data-driven malware detection using natural language processing and machine learning techniques on behavioral analysis reports. *Digital Investigation*, 28:S77–S87, 2019.

[125] VMRay. Now, near, deep: The power of multi-layered malware analysis detection, 2021.

[126] Andreas Moser, Christopher Kruegel, and Engin Kirda. Limits of static analysis for malware detection. In *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, pages 421–430. IEEE, 2007.

[127] Yingbo Song, Michael E Locasto, Angelos Stavrou, Angelos D Keromytis, and Salvatore J Stolfo. On the infeasibility of modeling polymorphic shellcode. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 541–551, 2007.

[128] Kaspersky. Behavior-based protection.

[129] Brad Miller, Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Rekha Bachwani, Riyaz Faizullabhoy, Ling Huang, Vaishaal Shankar, Tony Wu, George Yiu, et al. Reviewer integration and performance measurement for malware detection. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings 13*, pages 122–141. Springer, 2016.

[130] Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. MAMADROID: Detecting Android Malware by Building Markov Chains of Behavioral Models. In *Annual Symposium on Network and Distributed System Security (NDSS)*, 2017.

[131] Chani Jindal, Christopher Salls, Hojjat Aghakhani, Keith Long, Christopher Kruegel, and Giovanni Vigna. Neurlux: dynamic malware analysis without feature engineering. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 444–455, 2019.

[132] Carsten Willems, Thorsten Holz, and Felix Freiling. Toward automated dynamic malware analysis using cwsandbox. *IEEE Security & Privacy*, 5(2):32–39, 2007.

[133] Ulrich Bayer, Imam Habibi, Davide Balzarotti, Engin Kirda, and Christopher Kruegel. A view on current malware behaviors. In *Proceedings of the 2nd USENIX Conference on Large-Scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More*, LEET'09, page 8, USA, 2009. USENIX Association.

[134] Martina Lindorfer, Clemens Kolbitsch, and Paolo Milani Comparetti. Detecting environment-sensitive malware. In *International Workshop on Recent Advances in Intrusion Detection*, pages 338–357. Springer, 2011.

[135] Songsong Liu, Pengbin Feng, Shu Wang, Kun Sun, and Jiahao Cao. Enhancing malware analysis sandboxes with emulated user behavior. *Computers & Security*, 115:102613, 2022.

[136] Najmeh Miramirkhani, Mahathi Priya Appini, Nick Nikiforakis, and Michalis Polychronakis. Spotless sandboxes: Evading malware analysis systems using wear-and-tear artifacts. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 1009–1024. IEEE, 2017.

[137] Davide Balzarotti, Marco Cova, Christoph Karlberger, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. Efficient detection of split personalities in malware. In *NDSS*. Citeseer, 2010.

[138] Dhilung Kirat, Giovanni Vigna, and Christopher Kruegel. BareCloud: Bare-metal analysis-based evasive malware detection. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 287–301, San Diego, CA, August 2014. USENIX Association.

[139] Erin Avllazagaj, Ziyun Zhu, Leyla Bilge, Davide Balzarotti, and Tudor Dumitraș. When malware changed its mind: An empirical study of variable program behaviors in the real world. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 3487–3504, 2021.

[140] Wenyi Huang and Jay Stokes. Mtnet: A multi-task neural network for dynamic malware classification. In *Proceedings of 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA 2016)*, pages 399–418. Springer, July 2016.

[141] Alexander Küchler, Alessandro Mantovani, Yufei Han, Leyla Bilge, and Davide Balzarotti. Does every second count? time-based evolution of malware behavior in sandboxes. In *NDSS*, 2021.

[142] Awesome malware analysis, 2023.

[143] Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, et al. Wilds: A benchmark of in-the-wild distribution shifts. In *International Conference on Machine Learning*, pages 5637–5664. PMLR, 2021.

[144] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. TESSERACT: Eliminating experimental bias in malware classification across space and time. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 729–746, 2019.

[145] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In *International conference on machine learning*, pages 1180–1189. PMLR, 2015.

[146] Boqing Gong, Kristen Grauman, and Fei Sha. Connecting the dots with landmarks: Discriminatively learning domain-invariant features for unsupervised domain adaptation. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 222–230, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[147] Shiori Sagawa, Aditi Raghunathan, Pang Wei Koh, and Percy Liang. An investigation of why overparameterization exacerbates spurious correlations. In *International Conference on Machine Learning*, pages 8346–8356. PMLR, 2020.

[148] Yizheng Chen, Zhoujie Ding, and David Wagner. Continuous learning for android malware detection. *arXiv preprint arXiv:2302.04332*, 2023.

[149] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. Dos and don'ts of machine learning in computer security. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3971–3988, Boston, MA, August 2022. USENIX Association.

[150] Arthur S Jacobs, Roman Beltiukov, Walter Willinger, Ronaldo A Ferreira, Arpit Gupta, and Lisandro Z Granville. Ai/ml for network security: The emperor has no clothes. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1537–1551, 2022.

[151] Giovanni Cherubin, Rob Jansen, and Carmela Troncoso. Online website fingerprinting: Evaluating website fingerprinting attacks on tor in the real world. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 753–770, 2022.

[152] Sanjeev Das, Jan Werner, Manos Antonakakis, Michalis Polychronakis, and Fabian Monrose. Sok: The challenges, pitfalls, and perils of using hardware performance counters for security. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 20–38. IEEE, 2019.

[153] Hojjat Aghakhani, Fabio Gritti, Francesco Mecca, Martina Lindorfer, Stefano Ortolani, Davide Balzarotti, Giovanni Vigna, and Christopher Kruegel. When malware is packin'heat; limits of machine learning classifiers based on static analysis features. In *Network and Distributed Systems Security (NDSS) Symposium 2020*, 2020.

[154] Z Cliffe Schreuders, Thomas Shaw, Mohammad Shan-A Khuda, Gajendra Ravichandran, Jason Keighley, and Mihai Ordean. Security scenario generator (secgen): A framework for generating randomly vulnerable rich-scenario vms for learning computer security and hosting ctf events. In *ASE@ USENIX Security Symposium*, 2017.

[155] Broadcom. Submitting symantec endpoint protection telemetry to improve your security, 2023.

[156] Malwarebazaar, 2023.

[157] Virustotal, 2023.

[158] Miuyin Yong Wong, Matthew Landen, Manos Antonakakis, Douglas M Blough, Elissa M Redmiles, and Mustaque Ahamad. An inside look into the practice of malware analysis. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 3053–3069, 2021.

[159] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss*, volume 14, pages 23–26, 2014.

[160] Hyrum S Anderson and Phil Roth. EMBER: An open dataset for training static PE malware machine learning models. *arXiv preprint arXiv:1804.04637*, 2018.

[161] Richard Harang and Ethan M. Rudd. Sorel-20m: A large scale benchmark dataset for malicious pe detection, 2020.

[162] Shuofei Zhu, Jianjun Shi, Limin Yang, Boqin Qin, Ziyi Zhang, Linhai Song, and Gang Wang. Measuring and modeling the label dynamics of online Anti-Malware engines. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2361–2378. USENIX Association, August 2020.

[163] Silvia Sebastián and Juan Caballero. Avclass2: Massive malware tag extraction from av labels. In *Annual Computer Security Applications Conference*, pages 42–53, 2020.

[164] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 1113–1120, 2009.

[165] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems*, 34:18932–18943, 2021.

[166] TrendMicro. Gandcrab comes up with new tricks, 2018.

[167] Xabier Ugarte-Pedrero, Mariano Graziano, and Davide Balzarotti. A close look at a daily dataset of malware samples. *ACM Transactions on Privacy and Security (TOPS)*, 22(1):1–30, 2019.

[168] Íñigo Íncer Romeo, Michael Theodorides, Sadia Afroz, and David Wagner. Adversarially robust malware detection using monotonic classification. In *Proceedings of the Fourth ACM International Workshop on Security and Privacy Analytics*, pages 54–63, 2018.

[169] Michael Cao, Sahar Badihi, Khaled Ahmed, Peiyu Xiong, and Julia Rubin. On benign features in malware detection. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pages 1234–1238, 2020.

[170] Ping-Jui Chuang, Chih-Fan Hsu, Yung-Tien Chu, Szu-Chun Huang, and Chun-Ying Huang. Towards a utopia of dataset sharing: A case study on machine learning-based malware detection algorithms. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, pages 479–493, 2022.

[171] Zhikun Zhang, Tianhao Wang, Ninghui Li, Jean Honorio, Michael Backes, Shibo He, Jiming Chen, and Yang Zhang. PrivSyn: Differentially private data synthesis. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 929–946. USENIX Association, August 2021.

[172] Mohammad Naseri, Yufei Han, Enrico Mariconti, Yun Shen, Gianluca Stringhini, and Emiliano De Cristofaro. Cerberus: Exploring federated prediction of security events. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2337–2351, 2022.

[173] Tatsunori Hashimoto, Megha Srivastava, Hongseok Namkoong, and Percy Liang. Fairness without demographics in repeated loss minimization. In *International Conference on Machine Learning*, pages 1929–1938. PMLR, 2018.

[174] Shiori Sagawa*, Pang Wei Koh*, Tatsunori B. Hashimoto, and Percy Liang. Distributionally robust neural networks. In *International Conference on Learning Representations*, 2020.

[175] Serena Wang, Wenshuo Guo, Harikrishna Narasimhan, Andrew Cotter, Maya Gupta, and Michael Jordan. Robust optimization for fairness with noisy protected groups. *Advances in neural information processing systems*, 33:5190–5203, 2020.

[176] Platon Kotzias, Leyla Bilge, Pierre-Antoine Vervier, and Juan Caballero. Mind your own business: A longitudinal study of threats and vulnerabilities in enterprises. In *NDSS*, 2019.

[177] Mohammad Pezeshki, Oumar Kaba, Yoshua Bengio, Aaron C Courville, Doina Precup, and Guillaume Lajoie. Gradient starvation: A learning proclivity in neural networks. *Advances in Neural Information Processing Systems*, 34:1256–1272, 2021.

[178] Thomas Roccia. Evolution of malware sandbox evasion tactics – a retrospective study, Oct 2019.

[179] Alan Mills and Phil Legg. Investigating anti-evasion malware triggers using automated sandbox reconfiguration techniques. *Journal of Cybersecurity and Privacy*, 1(1):19–39, 2020.

[180] Senthil Purushwalkam and Abhinav Gupta. Demystifying contrastive self-supervised learning: Invariances, augmentations and dataset biases. *Advances in Neural Information Processing Systems*, 33:3407–3418, 2020.

[181] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15750–15758, 2021.

[182] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(11), 2008.

[183] VMRay. Analyzing location-based malware with geo anonymization, 2018.

[184] Marcus Botacin, Hojjat Aghakhani, Stefano Ortolani, Christopher Kruegel, Giovanni Vigna, Daniela Oliveira, Paulo Lício De Geus, and André Grégio. One size does not fit all: A longitudinal analysis of brazilian financial malware. *ACM Transactions on Privacy and Security (TOPS)*, 24(2):1–31, 2021.

[185] Platon Kotzias, Leyla Bilge, and Juan Caballero. Measuring pup prevalence and pup distribution through pay-per-install services. In *USENIX Security Symposium*, pages 739–756, 2016.