Generative Methods, Meta-learning, and Meta-heuristics for Robust Cyber Defense

Marc W. Chale

Generative Methods and Meta Learning for Cybersecurity

DISSERTATION

Marc W. Chalé, Captain, USAF

AFIT-ENS-DS-22-S-056

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENS-DS-22-S-056

GENERATIVE METHODS AND META LEARNING FOR CYBERSECURITY

DISSERTATION

Presented to the Faculty

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy in Operations Research

Marc W. Chalé, BS, MS, MS

Captain, USAF

September 2022

Wright-Patterson Air Force Base, Ohio

AFIT-ENS-DS-22-S-056

GENERATIVE METHODS AND META LEARNING FOR CYBERSECURITY

DISSERTATION

Marc W. Chalé, BS, MS, MS
Captain, USAF

Approved:

AFIT-ENS-DS-22-S-056

Abstract

Cyberspace is the digital communications network that supports the internet of battlefield things (IoBT), the model by which defense-centric sensors, computers, actuators and humans are digitally connected. A secure IoBT infrastructure facilitates real time implementation of the observe, orient, decide, act (OODA) loop across distributed subsystems. Successful hacking efforts by cyber criminals and strategic adversaries suggest that cyber systems such as the IoBT are not secure. Three lines of effort demonstrate a path towards a more robust IoBT. First, a baseline data set of enterprise cyber network traffic was collected and modelled with generative methods allowing the generation of realistic, synthetic cyber data. Next, adversarial examples of cyber packets were algorithmically crafted to fool network intrusion detection systems while maintaining packet functionality. Finally, a framework is presented that uses meta-learning to combine the predictive power of various weak models. This resulted in a meta-model that outperforms all baseline classifiers with respect to overall accuracy of packets, and adversarial example detection rate. The National Defense Strategy underscores cybersecurity as an imperative to defend the homeland and maintain a military advantage in the information age. This research provides both academic perspective and applied techniques to further the cybersecurity posture of the Department of Defense into the information age.

Acknowledgements

Just try to make the world a better place for your having been here.

James Doolittle

Thank you to the faculty who invested in me over four years. I look up to all of you. Jeffery Weir, Bruce Cox, Eric Brooks, Ray Hill

Thank you to Nate Bastian who worked overtime to advise my research and mentor me as an officer.

Marc W. Chalé

Preface

This dissertation document is compiled in the *K-papers* format. Each research contribution, Chapters III-V and Appendix A, are taken from a manuscript that has either been submitted for publication or is intended to be submitted for publication. Chapter III [1] and Appendix A [2] have been published and are included with permission from the publisher according to their respective author rights agreements.

MARC CHALÉ is a graduate student in the Department of Operational Sciences at the Air Force Institute of Technology (AFIT) pursuing a Ph.D. degree in Operations Research. He also earned M.S. degrees in both Industrial Engineering and Operations Research. Prior to attending AFIT, Mr. Chalé was assigned to Air Force Operational Test and Evaluation Center as a senior operational test analyst conducting test for Air Force One, AWACS, and the Light Attack acquisitions programs.

In conjunction with his program of study at AFIT, Mr. Chalé is a research associate at the Army Cyber Institute. His recent research has involved the use of discriminative and generative machine learning, Bayesian methods, and meta-learning to provide cybersecurity models that are robust to adversarial attack. His email address is `marc@marcchale.com` His website is `http://www.marcchale.com/resume.html`

Table of Contents

Page

Page

List of Figures

List of Tables

Table Page

GENERATIVE METHODS AND META LEARNING FOR CYBERSECURITY

I. Dissertation Introduction

1.1 Motivation

The United States Department of Defense (DoD) anticipates future wars will be fought largely over the cyber domain against both strategic competitors and non-state actors. Because the United States has never fought a full scale cyber war, the “rules of the road” are not well understood [6]. Adversaries are likely to target American interests through both known and unknown threat vectors. The effects of these attacks may be either non-kinetic, meaning unpermitted access or control of information systems, or kinetic, implying the attack leads to the destruction of physical assets, damage to infrastructure, or death. Many legacy cyber-physical systems are built with no foresight to cyber vulnerabilities [7]. As the internet of battlefield things grows to include more of these systems, the potential cyber threat exposure also grows. Imagine the confusion as a soldier’s wearable device malfunctions in combat due to cyber attack. It is critical that we address our military’s cybersecurity shortcomings with novel techniques before adversaries exploit them. Generative machine learning and meta-learning are emerging fields that may offer solutions to some longstanding roadblocks in cybersecurity research.

Intrusion detection systems (IDS) are an approach to deter and defend from cyber attacks [7]. Unfortunately, IDS require large data sets for training [2]. Organic cyber attack data, with labelled entries, is notoriously scarce. The NSL-KDD [8] attempts to rectify issues in the well cited KDD-CUP benchmark dataset, however, even the improved version is outdated and limited in scope.

Generative machine learning is a field of artificial intelligence with potential to address unsolved problems in new ways. Methods such as Markov Chain Monte Carlo, Autoencoders and Generative Adversarial Networks (GANS) and autoencoders are used to estimate unknown probability distribution functions. Applications for diverse and realistic generated data are pressing, especially for cyber. Generative methods provide an avenue to analyze and synthesize cyber data, while the combination of generative methods with meta-learning offer an opportunity to protect from certain cyber attacks.

The remainder of this chapter introduces three research topics promoting the security of United States cyber systems. Chapter 2 provides an an overarching literature review of related topics and a quick reference table of hand picked sources that may be especially valuable to readers. Chapters 3-5 provides completed research manuscripts corresponding to contributions 1, 2, and 3. Previously published research is Chapter VI concludes by summarizing the primary findings of the research and how they impact modern defense. The appendices provide additional information not suitable for the main document. Appendix A is a related study in meta-learning NIDS that does not fit into stated contributions. Appendix B is a table of referenced AFIT theses. Appendix C includes data tables supporting contribution 1.

1.2 Research Contributions

Three research topics are presented to support modernization of the military's security posture. Although each topic could have been pursued independently, this dissertation takes a sequential approach where results of the early research enhances the later work. The overarching goal of this dissertation is to demonstrate significant progress towards an intrusion detection system robust against adversarial attacks.

Contribution 1: Generate Realistic Synthetic Cyber Data.

The first research goal is to model the probability distributions of modern cyber data and generate additional, realistic, data from the baseline distributions. The intended generative model may be explicit, in the form of a probability distribution function, or implicit as with a GAN. Generative methods are discussed in Chapter 2.2. Regardless, the realistic data generated by the model must provably match the distribution of the baseline data. Unlike other benchmark data sets such as NSL-KDD [8], KDD-CUP [9], UNSW-NB15 [10], critiqued in Chapter 4.2, the generated data must be representative of network traffic in modern government systems, including examples for both authorized and malicious actors, in appropriate proportion. Malicious traffic must be representative of modern cyber attacks and reflect unobserved examples from the original distribution. A possible strategy is to train a generative model from real cyber data collected in an adversarial environment or alternatively collected in a realistic high fidelity simulation. Then the baseline data can be used to train a generative model capable of creating new, realistic examples from the same distribution as the baseline.

In particular, generative models should emphasize resilience against mode collapse and should model macro level correlation between variables. If successful, the realistic generated cyber data will be used as a starting point for creating adversarial examples. The enlarged, generated data sets are preferred over small real sets because it demonstrates the feasibility of generative methods to overcome the lack of data in novel cyber attacks. As new phenomena are discovered in cyber log data, they will be reproduced to a greater quantity, facilitating creation of adversarial examples and robust IDS. If the generative methods do not yield realistic data, then objective two can be pursued using lager quantities of baseline data, which is expensive and laborious to obtain. Two archival peer reviewed papers have been submitted and accepted in support of contribution 1. *Challenges and Opportunities for Generative Methods in the Cyber Domain* has been accepted to *Proceedings of the 2021 Winter Simulation Conference*, and *Generating Realistic Cyber Data for Training and Evaluating Machine Learning Classifiers for Network Intrusion Detection Systems* has been submitted to *Expert Systems with Applications.* Both of these works were authored by Marc Chalé (primary author) with contributions by committee members in support of dissertation research. The work supporting contribution 1 is presented in Chapter III and Appendix C.

Contribution 2: Generate Adversarial Examples.

The second research objective is to produce adversarial examples capable of evading a modern IDS. The adversarial examples must be created using novel techniques, including generative methods where applicable. The adversarial examples must go beyond work such as [11] by enforcing immutable aspects of cyber data [12] and enable an end-to-end attack. Solving this challenge may increase the effectiveness of state-of-the-art cyber attacks against current IDS, but once these techniques are identified, they can be addressed in robust IDS. Despite recent progress creating adversarial attacks in the computer vision domain, adversarial attack generation in the cyber domain is especially challenging [12]. In order for perturbed internet protocol (IP) packets to facilitate an end-to-end cyber attack, the packets must maintain their specialized data structures as well as their original function when executed. While images can be perturbed with no restrictions and result in a valid image file, IP data packets transmitted over the internet would be corrupted during perturbation, resulting in an ineffective end-to-end attack. Although initial research on adversarial attack in the cyber network domain [11] [13] [14] focused on perturbing feature vectors of the cyber data, a more difficult task would be to perturb the actual payload of a cyber packet while maintaining its original function [13] [15] [12]. Alternatively, an adversarial feature vector may be generated and then reverse engineered into a functional IP data packet that evades the IDS. In working towards an end-to-end black box attack, it is imperative that we demonstrate adversarial examples can be constrained to the standards of the cyber domain. This goal is achieved in *Constrained Optimization Based Adversarial Example Generation for Transfer Attacks in Network Intrusion Detection Systems*, a journal article submitted to *Computers & Industrial Engineering.* This work was authored by Marc Chalé (primary author) with contributions by committee members in support of dissertation research. The work supporting contribution 2 is presented in Chapter IV and Appendix D.

Contribution 3: Demonstrate a Robust Intrusion Detection System.

Intrusion detection systems play a major role in protecting the confidentiality, integrity, and availability of data on networked systems, however they have fundamental flaws. The detection rate of several popular rule based IDS against malware is surprising low in practice. One study finds that Zeek detects only 52% of malware attacks with its rule based alert system [16]. This lackluster performance may have motivated the recent developments in machine learning intrusion detection systems. While IDS capabilities have increased in recent years, adversaries also innovate their methods. Further, the rate of reported breaches in the United States has been increasing since 2005. The majority of IDS breaches are believed to be the result of evasion attacks where the IP packets are modified to seem innocuous, but are in fact harmful [17]. In modern times, evasion attacks such as [11] use heuristics to perturb characteristics of IP packets and fool the IDS.

Therefore, the final research objective is to improve both the classification performance and the robustness of machine learning based IDS such as [2] leveraging technologies such as GML and meta-learning. By classification performance, we specifically point to the metrics recall (detection rate) and accuracy. Robustness is the tendency for an algorithm to generalize well with examples drawn from a different distribution than the examples used for training [18]; it is an increasingly important characteristic of models in today's cyber environment.

While contribution 2 exposes a security vulnerability for ML based IDS, contribution 3 offers a solution. This research objective is achieved in *MADFACTS: Meta-learning Augmented Defense For Adversarial Cyber Techniques*, a completed full length article pending submission to publications such as *Computers & Security*, *Future Internet*, or *Optimization Letters*. This work was authored by Marc Chalé (primary author) with contributions by committee members in support of dissertation research. The work supporting contribution 3 is presented in Chapter IV.

Impact.

The research objectives outlined above have a synergistic impact on cyber defense for the IoBT and national security at large. Contribution 1 seeks to resolve the long standing lack of labelled high quality training data in the cyber domain. Contribution 2 provides a technical edge against cyber criminals and adversaries who wish to develop novel adversarial attacks against the IoBT. The succes of contributions 1 and two have enhanced work on contribution 3, where a robust IDS defeats adversarial examples. These accomplishments fit into a greater vision of military strategy that enables freedom of maneuver across all domains including cyber, space, land, air, and sea. Strengthening the cyber security across the IoBT is imperative for commanders to inflict desired impacts in modern cross domain warfare where command, control, intelligence, and recognisance is the backbone of decision making and is increasingly digitized. This research offers one promising path to improve robustness against the constantly evolving threat of adversarial attack.

II. Dissertation Literature Review

2.1 Cybersecurity

In 1972, a panel of experts prepared a comprehensive report on cybersecurity threats for the Air Force Systems Command. The consensus in the report was that a secure computer network would require designing new systems with security controls ingrained into the operating system. Adding security features onto existing systems would not provide sufficient protection from malicious attacks. Further, users should only be granted a level of freedom within the system commensurate with their credentials. The report presents cost estimates for securing systems and concludes that the potential savings from defeated attacks would justify such costs. The report served as a serious wake up call for military leaders in the new information age [19]. By 1980, [20] reports that the common reasons for an attacker to compromise cyber assets are to access information, corrupt information, or degrade the system. Flaws in system design become vulnerabilities for attack. Even true outsiders who have no insider knowledge of a system can penetrate if if they are skilled and well resourced. One such situation occurs when a wire tap into a private network is performed. The report recommends increased logging and auditing of network activity. Illegal users masquerading as legitimate users will behave differently than legitimate users. Therefore by performing outlier analysis on logged parameters, it is possible for a surveillance program to detect penetrations [20]. An online database of seminal papers in cybersecurity is offered by [21].

Intrusion Detection Systems.

Intrusion detection systems (IDS) are ubiquitous in modern networks and have been the subject of prominent research [22] [23] as well as a series of recent works conducted by AFIT affiliated Scholars [24] [11] [25] [2]. The function of the IDS is to monitor the records of traffic within a cyber network and detect the records of malicious actors [26]. IDS that react to the threat are known as intrusion detection and prevention systems [27]. Traditionally, anomaly based and signature based are the broad categories of IDS. Anomalies are abnormalities in a the traffic data set which are assumed to indicate malicious actors [22]. Alternatively, signature based IDS leverage previous knowledge to determine if an example is malicious [22]. Open source security monitor tools like Zeek and SGUIL use rule based signature approaches [28].

[29] provides a broad survey of anomaly detection techniques including a discussion oriented towards IDS. There are several categories of anomalies, each with their own qualities. Point anomalies are defined by their large distance from all other points. Contextual anomalies are noteworthy because of their context, or neighborhood where the point occurs. The contextual anomally might appear normal if relocated, so it is a quality like time or location that qualifies it as anomalous. Collective anomalies are characterized by data points that relate to each other in an unusual way. For instance, the actions *buffer-overflow, ssh, ftp* are routinely logged by a computer. However it is unusual for them to occur in sequence many times; the repetition of that pattern is anomolous and may indicate a remote attack. Although Chandola et al. [29] catalogues many anomaly detection techniques, there are several considerations that play a roll in IDS. The temporal nature of cyber data should be utilized for anomaly detection. Many detectors struggle with imbalanced data sets and missing labels. As a result, supervised, semi-supervised, and unsupervised methods can contribute to anomaly detection for cyber data [29].

[22] demonstrates that machine learning has become a standard tool set for creating an IDS. [30] reports that the majority of recent IDS publications utilize machine

learning and that artificial neural networks are the most popular choice of machine learning model. These are primarily signature based approaches. The meta-learning approaches of [2] and [31] leverage the strengths of a variety of machine learning algorithms for improved IDS. There are also strategies for using machine learning models to perform anomally detection.

[32] offers a strategy to perform anomaly detection that leverages an undercomplete autoencoders, a type of generative machine learning (GML) discussed in Chapter 2.2. A recent AFIT thesis [25] implements the autoencoder strategy for Network IDS (NIDS) using data collected by sensors across the DoD Information Network. Data examples that have high mean square error (MSE) when regenerated by the autoencoder are believed to be anomalous because they do not lie in the distribution generalized by the autoencoder. Therefore, MSE of test data points are used as outlier factor score. Butt suggests that if the outlier factor score of a test example could also be generated by testing an example of Gaussian noise, the real test example is an outlier. Non-outliers would have lower outlier factor scores. This threshold needs deeper consideration since a relatively large number of test points meet this criteria. The proper way to determine the threshold is to treat it as a varied hyperparameter and use validation testing to select the threshold with best classification performance. Butt's model is never validated because labels were removed prior to experimentation and never reintroduced. Butt used DOE to minimize the reconstruction error in data examples. It is also unclear why it is beneficial to globally minimize MSE before assessing outlier factor score on test data. It is always possible to reduce MSE on a data set by adding capacity to the ANN, but this results in overfitting. The goal of this experiment should be to fit a generalizable model and compare error between examples, not minimize all error. Butt concludes by recommending future anomaly based IDS experimentation with using sparse, overcomplete autoencoders [25].

Adversarial Attacks.

Well trained machine learning models can perform exceptionally well at classification when tested on examples of a similar distribution from the training examples [33]. These models however are often unable to perform when small, yet deliberate, perturbations are induced induced into the data set. The perturbations are often imperceivable by a human analyst, though detrimental to the classifier model. Attackers leverage this technology to inflict harm onto the users of the target model [34][35]. Adversarial examples are believed to provide some insight onto the differences between human and machine intelligence [3].

The field of adversarial machine learning (AML) stems from seminal research by Szegedy et al. [36] that demonstrated small, yet deliberate, perturbations to training data can consistently result in future misclassifications. There are four AML attack types including *evasion*, *extraction*, *poisoning*, and *inference.* Evasion attacks, in particular, reflect a malicious data packet that attempts to bypass an operational classifier [18]. The general approach of evasion adversarial attacks is to perturb malicious data examples in such a way that a classifier classifies them as normal [18]. In this way, hackers can achieve illegal access to systems, interfering with the proper confidentiality, integrity, and availability of data in the system [28].

Early demonstrations of evasion adversarial attacks focus on the computer vision domain[37]. Adversarial attacks in the computer vision domain are interesting because the adversary can slightly increase or decrease the intensity of any pixel slightly without a human detecting the changes; there are no domain specific constraints to the perturbations.

Adversarial attacks in the cyber domain are more challenging because perturbations to internet protocol (IP) packets may degrade their functionality when deployed in a cyber system. Real world adversarial attacks, therefore, become a constrained optimization problem [12] [38]. We consider the constraints of IP packets that maintain the original packet function and allow the packet to flow through the cyber network to the destination IP address, and client application.

Constraints to Perturbations in Cyber Domain.

One approach to creating adversarial examples of IP packets involves modifying the features of packets that are extracted by IDS software or network security monitoring software. The adversarial examples would have the same fields as the real data set, but some values would be fabricated [39]. [39, 33] use features generated by Zeek HTTP logs. The features used by [33] are shown in Table 2. The field called *Status* is the class label and the other fields are indicator variables. Notably, 9 of 11 indicator fields are discrete. Only *request_body_len* and *response_body_len* are continuous in this feature scheme. The *hp* field is an IP address and [40] shows several alternative representations to the bit vector shown in Table 2. Continuous representations are certainly possible. The other features used by [33] are categorical, and cannot be represented in a continuous form. The fields of Zeek logs are customizable and can be adapted for specific analysis applications.

Table 2. **Data Fields and Feature Engineering Technique**

Data Field	**Example**	**Engineered Feature**
connection	open, close, none	Label encoded
id_orig_h_cc	US	Label encoded
id_resp_h_cc	US	Label encoded
severity	H, M, L	Label encoded
method	GET, POST	Label encoded
request_body_len	10	minimax
response_body_len	10	minimax
hp	0000000000000000	16 bit vector
id_orig_h_org	external	Label encoded
id_resp_h_org	internal	Label encoded
host	my.connection.edu	Label encoded
status	normal, bad	Label encoded

There is a required structure that IP packets must adhere to in order to reach their destination. The Internet Assigned Numbers Authority (IANA) published standards [41] for communication over IP versions 4 and 6. The standards focus on packet structures and header fields and do not discuss rules for packet payloads. IP addresses are a fundamental field of cyber information packets. They are formatted as a dot decimal number of four 8 bit integers. That is, each IP address contains four binary octets and each octet represents a number between 0 and 255 (223.255.255.255). Unsurprisingly, there are a quantity of reserved IP addresses that cannot be taken by ordinary users.

Constraints on perturbations to cyber data is discussed by [12] [41] [42] [43]. The issue is investigated experimentally by [42] who suggests that application specific constraints can be applied to specific features being perturbed. [43] uses a gradient descent approach called Feasible Evasion Attacks on Neural Networks in Constrained Environments (FENCE). Since many cyber features are discrete, it is not possible to perturb in any direction or any distance. The perturbation generation algorithm uses model gradients to identify direction of optimal perturbation, but projects the gradient in a feasible direction at each step. There is also the important issue of maintaining inter-feature feasibility while pursuing feature perturbations, and this is difficult to enforce [44]. The steps of FENCE are only accepted if they satisfied predefined interdepency requirements of features. [43] provides a relationship between *number of packets*, *number of bytes*, and *connection duration* as an exemplary family of dependent features. FENCE attacks were effective in tests and adversarial training improved robustness of ANNs under attack.

[35] report that modifying features alone would not be sufficient for a realizable adversarial attack because there is no way to reconstruct the actual IP packet from information in the features. In order for an end-to-end attack to succeed, perturbation must result in a realizable IP packet to send through the cyber network. The perturbed packet must arrive at the client application and perform the malicious function. [44] describes a concept called *power* of adversarial attack. Training data, feature set, detection model, oracle, and manipulation depth are the elements of power. Attacks that possess all aspects of power are typically premised on unrealistic assumptions. For example, it is almost impossible for an attacker to have read and write privilege of NIDS training data. Therefore, poisoning attacks are extremely impractical. It is also unlikely that an attacker knows the exact features of a proprietary NIDS model. If features are somehow inferred, it is possible to manipulate packets to achieve some value in a particular feature. For instance, a NIDS may use net flow data, and there may be a feature describing connection duration. Packets can be

sent at specified schedule to achieve the desired connection duration [43]. It is not reasonable to assume that an attacker is powerful enough to make these perturbations on the correct features, and achieve the desired result on the real detection model.

Raw traffic classification is an alternative to feature based NIDS. This type of classifier bypasses the layer of obscurity imposed by feature engineering or feature selection. [44] notes that adversarial attacks using feature data is inferior to adversarial attacks using raw traffic data because it is not reasonable to assume the attacker has knowledge of the features used by the target NIDS. Raw traffic NIDS produced by [45] produces a classification decision from the binary data comprising an IP packet[45]. [45] reports that feature based NIDS are vulnerable to feature spoofing but raw traffic NIDS are less vulnerable. Common features such as source IP address are found in packet headers and can easily be modified with open sourced tools such as scapy. [46] expands on the work of [45] but removes header information from training data and uses only a binary payload to train deep learning NIDS. Raw traffic classification is also highly reliable with experimental detection rate as high as 100% [45]. Raw packet perturbation is quite intriguing because, if properly constrained, it would result in a fully defined IP packet. [35] challenges the community to create an end-to-end adversarial attack that yields functional attack traffic, fools classifiers, and maintains malicious behavior. There is no evidence that this effect has been achieved yet in either feature based or raw traffic NIDS.

Generating Adversarial Examples.

Equation 1-2 demonstrates that the overarching goal of an adversarial example is to find the data vector that optimally misclassifies the example while constraining the amount of change. [42] uses a gradient based approach which incrementally perturbs the feature vectors of an example. Each perturbation step optimally increases the loss function of the victim classifier as shown in Algorithm 1. If successful, the perturbed example is misclassified at test time. They demonstrated this approach to add malicious strings of text to PDF files, but they did not demonstrate removing text or other possible attack vectors. The approach also requires that the loss function is differential, which may not be applicable to changing categorical features.

$$\mathbf{x}^* = arg \min_{\mathbf{x}} : \hat{g}(\mathbf{x}) \tag{1}$$

$$S.T.||\mathbf{x}^* - \mathbf{x}||_\infty \leq \epsilon \tag{2}$$

Algorithm 1: Framework for a gradient based evasion attack [42]

```
Input: $x^0$, initial attack point, t, step size $\lambda$, trade off parameter, $\epsilon$, small
  constant
Output: $x^*$, the final, perturbed example
Initialize $m = 0$
while $F(\mathbf{x}^m)) - F(\mathbf{x}^{m-1})) < \epsilon$, do
    m=m+1
    set $\nabla F(\mathbf{x}^{m-1})$ to a unit vector with direction
      $\nabla(g(x^{m-1}) - \lambda \nabla p(\mathbf{x}^{m-1}|y^c = -1))$
    $\mathbf{x}^m \leftarrow x^{m-1} - t\nabla F(\mathbf{x}^{m-1})$
    if $d(\mathbf{x}^m, \mathbf{x}^0) > d_{max}$
      then
        project $\mathbf{x}^m$ into feasible region
return x*
```

It is typically not possible for an attacker to know what ML classifier is being used by an IDS [47], however, it has been shown that adversarial examples are largely transferable between models [48] [11] [49] [36]. The phenomena of transferability lends itself to *virtual adversarial training* where neither the true model nor full class labels are available. In this case, perturbations are informed from querying a surrogate model [50].

[26] provides evidence that GAN based adversarial attacks are feasible in the cyber domain, supporting the conclusions of [11] that any unconstrained domain is vulnerable. [26] also notes however that it is difficult to optimize GAN convergence and that over-manipulation is counterproductive for evasion. There also appears to be dependency on feature engineering that is not well understood [26]. Four strategies of generating adversarial examples for evading IDS are compared in [11]. Monte Carlo, Particle Swarm Optimization, Genetic Algorithm, and Generative Adversarial Network (GAN) each demonstrate success against a variety of IDS. A black box approach to generating adversarial cyber examples with GANs is demonstrated by [47] to achieve almost perfect evasion. Several additional methods are outlined in [18].

2.2 Generative Machine Learning

Generative machine learning (GML) is a rapidly advancing field of machine learning that seeks to model the joint statistical distribution of the data in a dataset directly. GML methods heavily leverage Bayes theorem to provide a posterior distribution of the data that accounts for prior beliefs. GML stands in contrast to discriminative machine learning (DML) which does not directly model the distribution of data, but nonetheless models the conditional relationship of class label given predictor information. DML methods typically generate decision boundaries for class labels using such techniques as support vector machine and logistic regression. Often, DML models are well suited to perform classification tasks because relatively simple models perform very well. GML algorithms, which are typically more complex, offer the additional capabilities of using the joint distribution model to estimate missing data, and fabricate entirely new data records [51]. Several emerging uses of generative models that show potential for cyber security.

The explicit probability distribution created by GML is in many ways more informative that than the models from DML. Further, it is possible to test the correctness of the generated distribution against real data [52]. Work by [53] proposes a paradigm called generative adversarial networks (GANS) to learn the high dimensional probability distribution of the dataset and simultaneously generate new examples from the same distribution. GANS will be explored in greater detail in Chapter 2.2.

Generative methods have also expanded reinforcement learning models in several ways. First, Generative models that extract the distribution for time series data can be used to explore future states of the system given the current state. A sequence of decisions can be made based on querying a set of feasible actions and selecting the steps that are likely to lead to a desirable state. As another application, [54] has demonstrated agent can learn in a non-existing environment that is generated from the probability distribution. Learning in such an environment offers an inexpensive methods of training but does not harm the agent should it enter a dangerous state. Inverse reinforcement learning is an emerging technique that seeks to model the reward policy of a human or real system [55].

Many of the most popular classification algorithms today are supervised learning algorithms; they require training data with either no missing data or very little missing data. Semi-supervised learning is a training strategy that can leverage unlabelled data in addition to labelled data to better generalize the true distribution. Many generative methods can either learn from examples with missing data or impute estimates of the missing data to match the original probability distribution [52].

Since GML trains to the underlying probability distribution of a system, they can model multi-modal data in a way that other machine learning techniques cannot. For example, a probability distribution for a system may indicate multiple predictions are reasonably correct. Since many algorithms in statistical learning fit a model by minimizing a loss function, the resulting model will recommend the best guess of the example's class. DML models perform poorly for image processing because they output the average of likely responses, even if the value is implausible. Generative models, in contrast, learn the likelihood of each possible class, promoting a reasonable prediction [52].

GML is useful in any situation that requires more data than is naturally available [52]. For example, one may wish to increase the resolution of an image. A GML model will generate one plausible output, rather than average all plausible outputs into something unrealistic. A GANs model, for instance, has been used to generate realistic art that was not specifically created by a human. Image-to-image translation has been used to synthesise a photograph from a sketch and to produce a map from an aerial photograph [56]

Generative Adversarial Network.

Generative adversarial nets (GANS) are a central area of research due their efficiency in generating realistic data that is similar to the source data. Goodfellow et al introduced GANS and demonstrated generation of fake but realistic images [53]. Research and applications of GANS have since increased, and Goodfellow [57] revisits the topic with recent theoretical insight on the mechanisms allowing GANS to work. The premise of the GAN is an adversarial game between two artificial neural networks. The first network, called the generator, is initialized with random weights, and receives white noise vector as an input. The network transforms the vector and outputs a modified synthetic vector. The other network, known as the discriminator, learns to distinguish the synthetic output vectors from real data vectors. Early in the process, the discriminator will recognize the random vectors produced by the generator. Through a backpropagation feedback process, the generator adjust weights to create more compelling, realistic output vectors. Simultaneously, the discriminator gains experience to better distinguish real and fake vectors. After sufficient training, the generator produces high quality synthetic data that consistently fools the generator, and sometimes humans [53]. Convolutional neural networks are commonly used for the generator and discriminator in GANs that produce images [52, 53]. GANs have also be used for adversarial example generation of cyber features. In this case, discriminators report realness and predicted class of the cyber feature vector [11]. This algorithm carefully constrains immutable features to ensure realness of the resulting cyber data [11].

Maximum Likelihood Estimation.

Although not every GML technique uses the maximum likelihood estimation to build the model, maximum likelihood is fundamental to many important GML algorithms [52]. The definition and required condition of a maximum likelihood estimation is given by [58].

Definition: Let $(\theta) = f(x_1, ..., x_n; \theta), \theta \in \Omega$ be the joint pdf of $_1, ..., X_n$. For a given set of observations, $(x_1, ..., x_n)$, a value $\hat{\theta}$ in Ω at which (θ) is a maximum is called a **maximum likelihood estimate** (MLE) of θ. That is, $\hat{\theta}$ is a value θ that satisfies $f(x_1), ..., x_n; \theta) = \max_{\theta \in \Omega} f(x_1, ..., x_n; \theta)$ [58]. Equation 3 is the derivative methods of solving maximum likelihood parameter. Equation 4 is the derivative of the log likelihood function, which is often more computationally efficient than taking the derivative of the raw likelihood function. Both strategies yield the same parameter estimate [58].

$$\frac{\mathrm{d}}{\mathrm{d}\theta^2} L(\theta) = 0 \tag{3}$$

$$\frac{\mathrm{d}}{\mathrm{d}\theta^2} \ln L(\theta) = 0 \tag{4}$$

Boltzmann Machine.

Bolztmann machines are a machine learning model that universally approximates probability distributions of data. Training algorithms typically optimize the maximum likelihood function by changing the model's weight parameters. The function itself is intractable so the approximate gradient descent techniques of [3] Chapter 18 are used.

The binary Boltzmann machine is one fundamental variant. These models were proposed by [59] and developed further in [60] [61]. Consider a binary random vector of dimension d, $\mathbf{x} \in \{0, 1\}^d$. The probability of a specific outcome us defined by the exponential of the negative energy function divided by the regularizing partition term, Z, which ensures a cumulative distribution of 1 [3].

$$P(\mathbf{x}) = \frac{\exp(-E(\mathbf{x}))}{Z} \tag{5}$$

The energy function for any outcome is defined as

$$E(\mathbf{x}) = -\boldsymbol{x}^\top \boldsymbol{U}\boldsymbol{X} - \boldsymbol{b}^\top \boldsymbol{x} \tag{6}$$

such that U is a weight matrix and of the parameters in the model and b is the bias vector for the parameters [3].

In many systems, some random variables are unknown or unobservable. The joint probability distribution, Equation 5, actually represents both visible and hidden variables in the system. Although the hidden variables are not specifically known for each data example, we may have information to characterize the joint probability of their state. We therefore decompose the vector, $\mathbf{x}$, into $\mathbf{v}$ the measurable variables, and $\mathbf{h}$, the latent variables. The energy functions can be decomposed as

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{v}^\top \mathbf{R}\mathbf{v} - \mathbf{v}^\top \mathbf{W}\mathbf{h} - \mathbf{h}^\top \mathbf{S}\mathbf{h} - \mathbf{b}^\top \mathbf{v} - \mathbf{v}^\top \mathbf{h}. \tag{7}$$

where $\mathbf{R}, \mathbf{W}, \mathbf{S}, \mathbf{b}$ and $\mathbf{c}$ are learned linear weights for the energy function. For a given distribution, variables that are likely to be mutually related are linked with a large weight, while variables not likely to be dependently activated will not have a large weight. Equation 7 provides a more detailed model of visible variables than Equation 5 because it incorporates interaction weights between hidden and visible variables. To estimate the probability of a state of visible variables, we can sum the marginal probabilities of the visible event, given all possible hidden states. Evidently, such a Boltzmann is extremely useful to characterize distributions when some information is missing. The method is flexible enough to model any binary distribution with approximately linear dependencies between variables [3].

A restricted Boltzmann machine is an important variant in which weights connect visible variables to hidden variables, but variables within any one layer are not connected to each other. Figure 1 a is a depiction of a restricted Boltzmann machine. The undirected bipartite graph has weighted connections between visible variables and hidden variables; variables of the same type are not connected [3].

The equations defining the probability and the energy of the restricted Boltzmann machine mirror the equations of the general Boltzmann machine. Joint probability distribution is defined as

$$P(\mathbf{v} = \boldsymbol{v}, \; \mathbf{h} = \boldsymbol{h}) = \frac{1}{Z} exp(-E(\boldsymbol{v}), \boldsymbol{h})), \tag{8}$$

and the energy is given as

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\boldsymbol{b}^\top \boldsymbol{v} - \boldsymbol{c}^\top \boldsymbol{h} - \boldsymbol{v}^\top \boldsymbol{W}\boldsymbol{h}, \tag{9}$$

and once again, the normalizing term Z is the sum of all energy states

$$Z = \sum_{\boldsymbol{v}} \sum_{\boldsymbol{h}} \exp\{-E(\boldsymbol{v}, \boldsymbol{h})\} \tag{10}$$

[3].

It is typically not tractable to calculate $P(\boldsymbol{v})$ directly, however, the bipartite structure of the weight graph allows us to calculate $P(\mathbf{h}|\mathbf{v})$ and $P(\mathbf{v}|\mathbf{h})$ according to the steps shown in Equations 11-15.

$$P(\mathbf{h}|\mathbf{v}) = \frac{P(\boldsymbol{h}, \boldsymbol{v})}{P(\boldsymbol{v})} \tag{11}$$

$$= \frac{1}{P(\boldsymbol{v})} \frac{1}{Z} \exp\{\boldsymbol{b}^\top \boldsymbol{v} + \boldsymbol{c}^\top \boldsymbol{h} + \boldsymbol{v}^\top \boldsymbol{W}\boldsymbol{h}\} \tag{12}$$

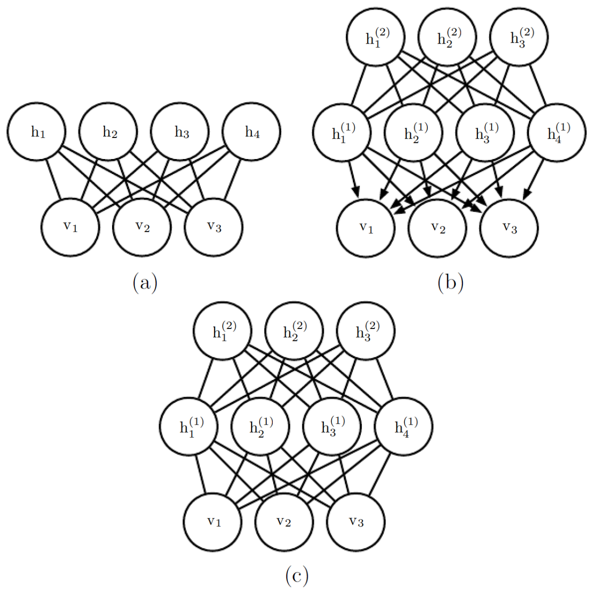


Figure 1. (a) The restricted Boltzmann machine model is visualized with bipartite weighted graph where there are connections between hidden and visible variables. (b) A deep belief network. (c) A deep Boltzmann machine. Figure credit: [3]

$$= \frac{1}{Z'} \exp\{\boldsymbol{c}^\top \boldsymbol{h} + \boldsymbol{v}^\top \boldsymbol{W}\boldsymbol{h}\} \tag{13}$$

$$= \frac{1}{Z'} \exp\{\sum_{j=1}^{n_h} c_j h_j + \sum_{j=1}^{n_h} \boldsymbol{v}^\top \boldsymbol{W}_{:,j} h_j\} \tag{14}$$

$$= \frac{1}{Z'} \prod_{j=1}^{n_h} \exp\{c_j h_j + \boldsymbol{v}^\top \boldsymbol{W}_{:,j} h_j\} \tag{15}$$

We train the latent variables of the Boltzmann machine using the maximum likelihood estimate of $\boldsymbol{h}|\boldsymbol{v}$ using the visible parameters as the prior. Similarly, the Boltzmann machine also allows us to estimate $\boldsymbol{h}|\boldsymbol{v}$ for unseen data sets where some visible variables are missing [3].

Deep Belief Network.

Deep belief networks (DBN) were an early success in the field of deep learning. They were the first deep architecture that were computationally tractable to optimize at scale and they competed well with other popular kernel based optimization algorithms. Although DBNs have waned in popularity, they warrant study for their influence in the field of deep learning. The architecture of a DBN is illustrated in Figure 1 (b). It is comprised of one layer of visible variables and several layers of hidden variables. Variable nodes are fully connected to adjacent layers but are not connected within layers. Arcs to the visible layer are directed while all other arcs are undirected. Visible variables may belong to either the binary set or the real set of numbers but the latent variables are generally restricted to the binary set. The DBN will require one weight matrix $\boldsymbol{W}^{(k)}$ for each of l hidden layers and one bias vector for each of the $l+1$ total layers. The joint distribution of a DBM is expressed

$$P(\boldsymbol{h}^{(l)}, \boldsymbol{h}^{(l-1)}) \propto \exp\{{b^{(l)}}^\top h^{(l)} + {b^{(l-1)}}^\top h^{(l-1)} + {h^{(l-1)}}^\top \boldsymbol{W}^{(l)} \boldsymbol{h}^{(l)}\}. \tag{16}$$

Note that a deep belief network with one layer of hidden variables is a RBM.

Deep Boltzman Machine.

The Deep Boltzmann Machine is a generative machine learning model similar to the preceding methods but with undirected edges and multiple hidden layers. This results in a bipartite topology. The bipartite structure allows for powerful models of a particular layers variables conditioned on the variables of a connected layer. While variables within each layer are independent, the conditional relationship to neighboring layers is very valuable. Deep Boltzmann machines traditionally employ binary units, thought it has been extended to real valued units as well. Just like the previous methods, the energy function is used to characterize the distribution [3].

The probability distribution for a deep Boltzmann machine with one visible layer v and hidden layers h^1h^2 and h^3, the probability distribution function takes the form

$$P(\boldsymbol{v}, \boldsymbol{h}^1, \boldsymbol{h}^2, \boldsymbol{h}^3) = \frac{1}{Z(\boldsymbol{\theta})} \exp\left(-E\left(\boldsymbol{v}, \boldsymbol{h}^1, \boldsymbol{h}^2, \boldsymbol{h}^3; \boldsymbol{\theta}\right)\right) \tag{17}$$

and the energy function, with bias terms omitted, E is given by

$$E\left(\boldsymbol{v}, \boldsymbol{h}^1, \boldsymbol{h}^2, \boldsymbol{h}^3; \boldsymbol{\theta}\right) = -\boldsymbol{v}^\top \boldsymbol{W}^{(1)} \boldsymbol{h}^{(1)} - \boldsymbol{h}^{(1)\top} \boldsymbol{W}^{(2)} \boldsymbol{h}^{(2)} - \boldsymbol{h}^{(2)\top} \boldsymbol{W}^{(3)} \boldsymbol{h}^{(3)} \tag{18}$$

Training a deep Boltzmann machine is very efficient, requiring at least two updates with Gibbs sampling, and sampling can be performed simultaneously and independently for variables in adjacent layers. Unlike deep belief networks, however, deep Boltzmann machines require sampling for variables on each layer. Deep Boltzmann machines have gained favor over deep belief networks for their good performance approximating the conditional posterior distributions [3].

Autoencoders.

Autoencoders are algorithms that learn to encode an input data set to a smaller summary code, then process the encoded data such that the original set is recovered with minimal error. The encoder function, $\mathbf{h}$, is typically a single hidden layer artificial neural network, where $\mathbf{h} = f(\mathbf{x})$. The decoder, $\mathbf{r} = g(h)$ is the complimentary network responsible for transforming h back to its original form; it typically has a single hidden layer as well. The models are typically trained with minibatch gradient descent and back-propogation. A properly trained autoencoder learns $g(f(\mathbf{x})) = \mathbf{x}$ for all $\mathbf{x}$ in the entire domain [3].

It is not inherently useful, however, to perform the summary and reconstruction transformations on data. The primary advantage of an autoencoder is that we can learn which data features are most important to describe the system. These are the features passed through the encoding operation $f(x)$. Features that do not vary across input sets are not encoded but are learned by the generator transformation, $g(f(x))$ [3].

The technique has also been used as a strategy for dimensional reduction. An autoencoder that produces a code with smaller dimension than the input data is called undercomplete and has a loss function of the form

$$L(\mathbf{x}, g(f(\mathbf{x}))) \tag{19}$$

where the loss is measured as mean square error of the recreated data from the generator [3]. In the case of a linear transformation, h, learns the same model as principal component analysis (PCA), for the k most important components. Deep authoencoders, however tend to exhibit less reconstructive error than PCA [62]. Non-linear autoencoders have the advantage of additional capacity, however, they may risk overfitting to the training data without learning the important features of the data. Overcomplete autoencoders model the input data with a higher dimension than the original form. Overcomplete autoencoders can be especially useful for feature extraction if they are properly regularized [3].

Figure 2 depicts the concept of manifold learning that applies to many autoencoders. Autoencoders, like other generative methods, seek to discover a low dimension model of a system that provides the most possible information, assuming the point lies on the manifold. By this, we consider a manifold to be a subset of highly likely data points in the d-dimensional space. The model tends to behave well on data points on the manifold, and behaves poorly for data offset from the manifold. We characterize manifolds by the set of all tangent planes of points on the manifold. At the bottom of Figure 2, is an example of a defining vector for the one dimensional tangent plane. By travelling an infinitesimally small distance in this dimension, we remain on the manifold, but travelling orthogonal to the vector would result in a point off the manifold. The perpetually competing forces of an autoencoder include learning an accurate representation $\boldsymbol{h}$ of $\boldsymbol{x}$ near the manifold that is recoverable using a decoder, but also regularizing the representation unto a reasonably low dimension subspace.

Sparse Autoencoders.

Sparse autoencoders include a penalty $\Omega(\mathbf{h})$ in the loss function to promote convergence of transformations with many zeros and fewer non-zero weights. They are used in practice to select features for other machine learning tasks. The resulting loss function is if the form

$$L(\mathbf{x}, g(f(\mathbf{x}))) + \Omega(h) \tag{20}$$

where $g(\cdot)$ is the generator and the encoder is the transformation $\mathbf{h} = f(x)$.

Other regularization techniques such as weight decay have a Bayesian interpretation such that the maximum likelihood estimator $\theta|\mathbf{x}$ models the data. Sparse autoencoders cannot be interpreted in this way because the regularizer term depends on the input data itself and is not technically derived from the prior distribution [3]. The proper intuition of a sparse autoencoder is a framework for training a generator with unseen latent variables. We should consider that the explicit joint distribution of the data comprises of two parts; $p_{model}(\mathbf{x}, \mathbf{h}) = p_{model}(\mathbf{h})p_{model}(\mathbf{x}|\mathbf{h})$ where $\mathbf{x}$ is the set of visible variables and $\mathbf{h}$ is the set of latent variables. From this perspective, $p_{model}(\mathbf{h})$ is prior distribution, or belief knowledge of the latent variables prior to seeing input data. This perspective breaks convention that the Bayesian belief is $p(\theta)$, the training weights of the model. In practice, the log-likelihood function for

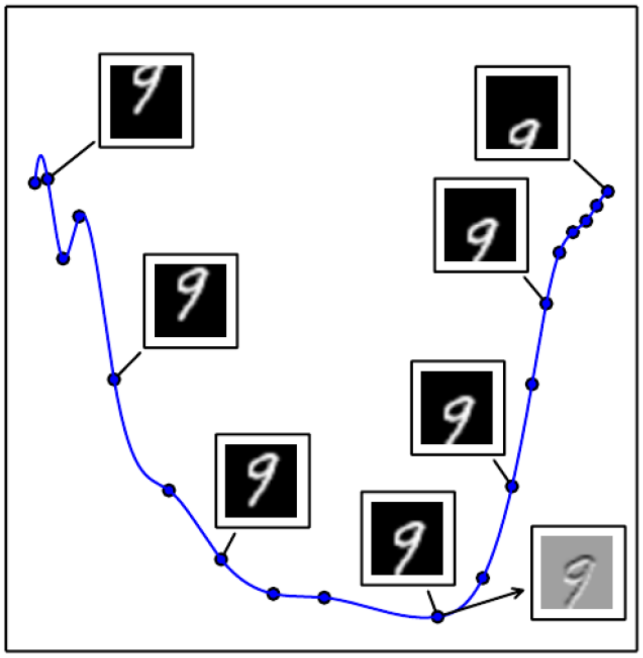


Figure 2. The blue function represents a one dimensional manifold, where the component (dimension) corresponds to vertical translation of a baseline image. Note that the manifold is represented in fewer dimensions than the original image. Figure credit: [3].

the model is

$$logp_{model}(\mathbf{x}) = log \sum_{h} p_{model}(\mathbf{h}, \mathbf{x}). \tag{21}$$

Since the goal is to select a point estimate of highly likely array **h**, the model is formed by maximizing

$$logp_{model}(\mathbf{h}, \mathbf{x}) = logp_{model}(\mathbf{h}) + p_{model}(\mathbf{x}|\mathbf{h}). \tag{22}$$

The term $logp_{model}(\mathbf{h})$ drives the sparsity property. A Laplace prior may be used to specify the regularization penalty value where the model is defined by

$$p_{model}(h_i) = \frac{\lambda}{2} e^{-\lambda |h_i|} \tag{23}$$

and the absolute value penalty is calculated as

$$\Omega(\mathbf{h}) = \lambda \sum_i |h_i| \tag{24}$$

Therefore, we expand the log likelihood model as

$$-\,log p_{model}(\mathbf{h}) = \sum_i (\lambda |h_i| - log \frac{\lambda}{2}) = \omega(\mathbf{h}) + const \tag{25}$$

Note that the constant term does not depend on $\mathbf{h}$, but only the hyperparameter λ, so the term does not affect the maximum likelihood estimate [3].

Stochastic Autoencoders.

Some advanced autoencoders generate stochastic codes that follow some distribution $p_{encoder}(\mathbf{h})|\mathbf{x}$. Since the encoding function, $\mathbf{h}$, is hidden, the decoder training is generalized over the distribution. The decoder follows a conditional distribution $p_{decoder}(\mathbf{x})|\mathbf{h}$.

Various functions such as Student's T distribution, or a binomial may be used as the prior in practice depending on whether the targets are real valued or binary. The choice of objective function and activation function tend to depend on the form of the prior. Stochastic autoencoders seek a more robust model and they have been applied as denoising autoencoders [3].

Score Matching.

As an alternative to the maximum likelihood method, score matching can be used to estimate the parameters that define distributions. We consider the score, the gradient field of the log distribution, log p_{data}

$$\nabla_{\boldsymbol{x}} \log\, p(\boldsymbol{x})$$

as a strategy for characterizing p_{data}.

As such, we must minimize the expectation of the squared difference between the first derivative of the log density of the model wrt. input data and the first derivative of the data's log density wrt. the input data. This difference is given by

$$L(\boldsymbol{x};\boldsymbol{\theta}) = \frac{1}{2}\|\nabla_{\boldsymbol{x}}\log\, p_{\text{model}}(\boldsymbol{x};\boldsymbol{\theta}) - \nabla_{\boldsymbol{x}}\log\, p_{\text{data}}(\boldsymbol{x})\|_2^2$$

A new objective describes the system as a function of only latent variables $\boldsymbol{\theta}$.

$$J(\boldsymbol{\theta}) = \frac{1}{2}\mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}L(\boldsymbol{x};\boldsymbol{\theta})$$

The solution to the score matching system is the vector $\boldsymbol{\theta}$ that minimizes $J(\cdot)$.

$$\boldsymbol{\theta}^* = \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

[3]

Variational Autoencoders.

The variational autoencoder learns a model, known as the approximate inference, to produce new samples of the learned distribution. First, a random variable, $\boldsymbol{z}$ is drawn from $p_{\text{model}}(\boldsymbol{z})$. The sample is then transformed by the directed network $g(\boldsymbol{z})$ generating output $\boldsymbol{x}$ from the distribution $p_{model}(\mathbf{x}|\mathbf{z})$.

During training, we take the following perspective of the model. The approximate inference network $q(\boldsymbol{z}|\boldsymbol{x})$ is the encoder that generates $\boldsymbol{z}$; and the decoder is the distribution $p_{model}(\boldsymbol{x}|\boldsymbol{z})$ from which the reconstructed output is drawn [3]. To properly train the variational autoencoder, we seek to maximize the variational lower bound, $\mathcal{L}(q)$, shown in Equation 26, for the sample $\mathbf{x}$.

$$\mathcal{L}(q) = \mathbb{E}_{\boldsymbol{z}q(\boldsymbol{z}|\boldsymbol{x})}\log p_{\text{model}}(\boldsymbol{z}, \boldsymbol{x}) + \mathcal{H}(q(\boldsymbol{z}|\boldsymbol{x})) \tag{26}$$

$$= \ \mathbb{E}_{\boldsymbol{z} \ q(\boldsymbol{z}|\boldsymbol{x})}\log p_{\text{model}}(\boldsymbol{x}|\boldsymbol{z}) - D_{\text{KL}}(q(\boldsymbol{z}|\boldsymbol{x})||p_{\text{model}}(\boldsymbol{z})) \tag{27}$$

$$\leq \log p_{\text{model}}((x)). \tag{28}$$

We note that in Equation 26 the two terms have tangiable meaning. First, we have the log likelihood of the visible and hidden variables given by the approximate posterior over all hidden variable. The following term is the entropy for the approximated posterior. When the entropy term is high, the variability of $\boldsymbol{z}$ is also high. In the case that q is normal, the standard deviation, or noise, around $\boldsymbol{z}$ is large for any vector that could have produced $\boldsymbol{x}$. That is, there is uncertainty around $\boldsymbol{z}$.

The subsequent equation, Equation 27, shows that the first term is the log likelihood for the reconstructed vector; the second term encourages similarity between the posterior and the prior distributions. The variational autoencoder is trained via back propagation, maximizing $\mathcal{L}$ by changing the parameters of the autoencoder. The expectations in $\mathcal{L}$ are found through Monte Carlo methods. Although the formulation for variational autoencoders is mathematically sound, researchers are still working towards ideal results. Some have noted that the Gaussian noise throughout the latent space lead to blurry images. MLE do not pinpoint precise values on outputs. Further, a relatively small potion of z is impacted by the encoding transformation. These deficiencies are not well understood and will likely be the focus of continued research [3].

Denoising Autoencoder.

The denoising autoencoder is used to learn a function that removes errors from a data set. Conventional autoencoders minimize a loss function in the form of

$$L(\mathbf{x}, g(f(\mathbf{x}))) \tag{29}$$

where $\boldsymbol{x}$ is the training data, f is the encoder, g is the decoder, and the loss function is mean squares or a comparable norm. The denoising autoencoder (DAE) minimizes the loss $L(\mathbf{x}, g(f(\cdot)))$ on $\tilde{\mathbf{x}}$, a corrupted version of the data. G is therefore incentivized to reverse, or invert, the operation that induced the noise. The properly trained DAE returns data nearly identical to the original uncorrupted data set. For any $\tilde{\mathbf{x}}$, the vector $g(f(\tilde{\mathbf{x}})) - \tilde{\mathbf{x}}$ translates the corrupted data point to the manifold of uncorrupted data, and the function $g(f(\cdot)))$ characterizes a vector field to the manifold of uncorrupted data.

Markov Chain Monte Carlo.

Markov Chain Monte Carlo (MCMC) sampling addresses several challenges within the field of Bayesian analysis. Many methods are unable to accommodate *high di-*

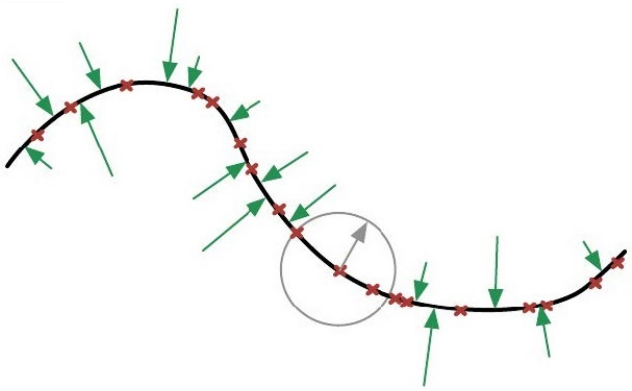


Figure 3. The vector field produced by a DAE, shown in green, transforms corrupted data back to the distribution manifold. Figure credit: [3].

mensional data, where in some cases the dimensionality is greater than the number of records. The complexity of these models lend to exponentially increasing computational time. Another situation is intractable inference, where a conditional distribution is estimated via an intensive summation over other variables marginal pdfs. Lastly, intractable normalization constant is problematic for certain function classes including logistic, where gradients are difficult to estimate. MCMC methods are appealing in these cases [3].

As a method within the broader field of Bayesian data analysis, we follow three high level steps of analysis. Firstly, the full probability model is defined in the form consistent with prior knowledge, including all observable and latent variables in the system. Second, we find the posterior distribution, that is the probability distribution of the latent variables conditioned on observed values. Estimates on the posterior distribution provide our approximate joint distribution. Finally, we evaluate the model according to fit, utility, and correctness of assumptions then assess any necessary revisions. Bayesian models can be improved by conditioning on known quantities. For example, if we have information about the genome of the parents of our test subjects, we can condition the probability of the child's gene expression on the known information of their parents [63].

The Bayesian model which conditions on observed data and incorporates prior beliefs is often more informative than the frequentist approach which calculates statistics from data alone [63].

One of the primary appeals of the Bayesian modelling framework is the compatibility with multi-level, or hierarchical modelling. These models incorporate parameters for phenomena at different levels in the system, each which plays a role in modelling the joint distribution. These models provide useful predictions in complex systems [63].

Markov Chain methods find model parameters θ that generate a posterior distribution $p(\theta|y)$ that matches a target distribution. It works by drawing parameters θ from a partially informative prior distribution, assessing the correctness of the draw, and progressing towards a better vector θ. Markov chain convergence occurs when the samples of θ exhibit a stationary distribution. The Markovian property implies that the sampling sequence leads to the same region of the parameter space regardless of initial point θ^0. Therefore, multiple chains achieving the same stationary distributions provide evidence that proper convergence has occurred.

Several sampling methods of MCMC modelling have unique advantages. Metropolis-Hasting [64] transitions across the parameter space according to a predefined sampling distribution and probabilistically tests the candidate parameter against an acceptance criteria before committing it to memory. The acceptance function promotes acceptance in high density regions of the joint distribution and discourages the algorithm from getting stuck at a local minima. The algorithm satisfies the condition of *detailed balance* [65] which theoretically guarantees the posterior distribution generated by the algorithm converges to the target distribution in the data after a large number of samples.

Gibbs sampling is an implementation of Metropolis-Hastings sampling such that a specified subset of variables are conditioned on other subset of variables [65, 63]. This sampling technique may be beneficial to converge more efficiently and model nuances of codependencies in the joint distribution.

No U-Turn Sampling (NUTS) is gradient based search and an extension of Hamiltonian Sampling [66]. Its advantage is that it eliminates several complications of tuning the Hamiltoniam model and it automatically terminates without need to set an exact number of draws. Although [67] advises efficient convergence using NUTS [66] for models of continuous variables, it also notes that the algorithm will fail to evaluate a gradient for datasets with discrete random variables [67]..

Metropolis-Hastings Sampling.

In some systems, the Bayesian probabilistic distribution of a variable follows a either an unknown form, or a known form that cannot be integrated across all variables. Gibbs sampling is a popular method for estimating the posterior distribution, however it does not address the issue of solving intractable integrals [68].

Consider a belief variable that follows a random distribution, and although the distribution parameters are unknown, the form of the distribution can be assumed. By searching for the parameters of the prior distribution, we can estimate the posterior distribution. The Metropolis-Hastings sampling method is a strategy to search for the maximum likelihood parameters for the prior distribution which in turn provides an estimation of the posterior distribution [68].

There are three overarching phases in each iteration of the Metropolis Hastings algorithm. First, a candidate parameter x^{cand} is proposed for the prior distribution. Next, the acceptance probability is calculated. Finally, the new parameter is either accepted or rejected according to the calculated acceptance probability[68].

When there is no specific information about the parameter setting x of the prior distribution, it is best to draw candidate points from a symmetrical distribution centered around the current point, $x^{(i-1)}$. This decision motivates exploration in all directions of the current point. This model is called the Random Walk Metropolis algorithm. Like all Markov chain systems, the choice of a future point depends only on the present setting, and not any previous states. If there is, however, information about the feasibility or likelihood of certain parameter values, a skewed distribution such as the log-normal or beta distribution should be used for exploration [68].

In order to converge towards the maximum likelihood parameters of the prior distribution, the acceptance function must motivate acceptance of values that produce high probability of the full joint density function. This is measured by the ratio of the joint densities at point $x^{(i)}$ and x^{cand}, $\frac{\pi((x^{cand})}{\pi((x^{(i-1)}}$. Simultaneously, the heuristic seeks to explore non-local points with some probability. This exploration is motivated by the ratio of the prior distributions $\frac{q(x^{(i-1)}|x^{cand})}{q(x^{cand}|x^{(i-1)})}$. The acceptance probability function comprised of these two components satisfies a condition called *detailed balance* which guarantees convergence towards the true posterior distribution [68, 63].

Algorithm 2: Pseudo-code of Metropolis-Hastings Algorithm [68]

```
Initialize $x^{(0)}$ $q(x)$
for i=1, 2..., do
    Propose: $x^{cand}$ $q(x^{(i)}|x^{(i-1)})$
    Acceptance Probability:
        $\alpha(x^{cand}|x^{()i-1} = min\{1, \frac{q(x^{(i-1)}|x^{cand})\pi(x^{cand})}{q(x^{cand}|x^{(i-1)})\pi(x^{i-1})}\}$
    $\mu \sim Uniform(\mu; 0, 1)$
    if $\mu < \alpha$ then
        Accept the proposal: $x^{(i)} \leftarrow X^{cand}$
    else
        Reject the proposal: $x^{(i)} \leftarrow X^{i-1}$
    predict recall
    record
```

Dirichlet Process.

Many data modelling tools use parametric statistics where the number of parameters is predetermined. Logistic regression may be an example. The trained model is therefore restricted in capacity to fit the data. Alternatively, Bayesian nonparametric statistics is a way to perform model selection among a form unrestricted in complexity. Therefore, underfitting is not an issue. Overfitting, too, is addressed by incorporating flexible prior beliefs of the posterior. Previous methods constrain the prior to a parametric distribution; the non-parametric approach allows the prior to follow any tractable distribution. The Dirichlet process (DP) is a popular model for defining a prior who's outputs are discrete distributions. The pattern modelled by DP follows the assumption that each observation of model parameters, θ_i, fall into a bin of previously observed theta, or fall into an unobserved category. The likelihood of θ_i being in a previously observed bin is a weighted balance of already observed data in the bin, and α, a pseudocount hyperparameter. The stick breaking example and the Chinese restaurant example are common educational analogies. The most popular applications of DP are Bayesian model validation, density estimation, and clustering via mixture models [69].

2.3 Meta-Learning

Meta-learning is an approach to machine learning that improves the learning of a base task as the base-learning algorithm gains experience. There are many strategies and applications of meta-learning. Several have potential applications in NIDS [2]. A selection these meta-learning technologies are outlined within.

Transfer Learning.

[3] describes transfer learning as the phenomena in which patterns learned for an initial task are beneficial for a future task. It is theorized that the when the two tasks contain the same important factors, transfer learning may be beneficial. This may be the case in computer vision where features such as a wheel, for example, may be important for separate classifiers that detect cars or bicycles. Domain adaptation is similar; it refers to an initial task that is learned in an initial setting, then the same task is redeployed to a new setting. Transfer learning and domain adaptation typically imply that the new task, or new domain process has a different distribution than the initial [3].

It has also been shown that representation impacts the effectiveness of machine learning tasks across domains. Analogously, a human asked to perform long division with Roman numerals will convert the numbers to Arabic numerals for a more intuitive process. Many algorithms perform tasks in various time complexities depending on the data format.

Representation may be key to why unsupervised learning often improves the performance of machine learning algorithms. When effectively pre-trained, new tasks can be learned with as few as one labelled example, an approach known as one-shot learning [3]. [70] provides a method of learning representations for tasks by selecting linear transformations of features. [71] provides a methodology of searching for transfer functions between tasks and optimizing the transfer policy. [72] Uses meta-learning to learn about various tasks, so new tasks can be learned with minimal training data. [73] describes the theoretical nature of learned features by representation learning systems. [54] provides some breakthrough applications of one shot learning in video frame prediction.

Local Interpretable Model Agnostic Explanations.

Black box machine learning models are used routinely for decisions of high consequence. In general, these models are assessed with classification performance metrics such as accuracy or recall. Humans who use the models for important decisions may be unwilling to blindly trust the model if the chance of incorrect classification is substantial, even if the model performs better than he human. Local Interpretable Model Agnostic Explanations (LIME) is a framework to develop trust for a machine learning decision that can be applied to any classifier. Trust, in this sense, encapsulates both trust for a good decision and trust of the model itself. Both aspects of trust are important for widespread adoption of machine learning beyond the research setting [74].

A reasonable approach for finding trust in a decision is to understand the rationale for the decision. To build trust of a model, multiple decisions are explored and mapped in an interpretable way [74]. LIME explores the decisions in a local region and provides an interpretable model of the decision space. The explanation may be qualitative in nature, but must be meaningful to the human user, especially one who is not an expert in machine learning. Further, the explanation should be accurate within a specified region of interest. While the important features may vary throughout the model, the decision explanation should be meaningful and faithful at any location in any complex model.

Let g be a meta-model that explains a predictive model and let G be the class of predictive model such as linear model or random forest. Noting that some explanative models of a particular class may be more complex than others, a measure of the complexity of g is the function $\Omega(g)$. The complexity is an indication of how well the model can be explained to a human. The classification model is denoted $f : \mathbb{R}^d \to \mathbb{R}$ and the probability of positive class is given by $f(x)$. The distance between point x and an exploratory point z is given by the function $\pi_x(z)$. Now, to measure the unfaithfulness of predictive model f in the region of x, we use the function $\mathcal{L}(f, g, \pi_x)$. A model is very trustworthy if both unfaithfulness, $\mathcal{L}(f, g, \pi_x)$, and complexity, $\Omega(g)$ are low. The LIME explanation is given as

$$\xi(x) = \underset{g \in G}{\operatorname{argmin}} \; \mathcal{L}(f, g, \pi_x) + \Omega(g) \tag{30}$$

The authors offer LIME as a strategy to improve interpretability of any machine learning algorithm and support informed decision making. Their report suggests future research brings LIME to new domains, optimizes the system for parallel processing, and strategically *picks* the most relevant examples to explain [74].

2.4 Data Sets

KDD CUP 99.

The KDD CUP 99 data set is regarded as the reference data set for anomaly detection analysis for IDS. The data was generated by Stolfo et al [9] in support of a DARPA study to use machine learning for IDS [9]. The cyber network traffic data was collected in the "tcdump" binary format. The training data set, which includes five million connections, was recorded during a sevn week study. An additional two million connections were recorded over two weeks and sequestered as test data. The data sets include 41 categorical and continuous features as well as a label that denotes normal or attack, and a label that specifies one of four attack types [9]. *Denial of service attacks* occur when an attacker barrages the network with frequent requests that surpass the victim's computing resources. *User to Root* attacks are where the attacker enters the system as a standard user but illegally gains a higher level of access. *Remote to Local* is a class of attack where the attacker does not have full access to a network but is still able to transmit data packets to manipulate a machine. The *probing attack* is when a user collects information about the security measures in order to manipulate vulnerabilities in the future [8].

According to [75], the KDD-CUP dataset was the best available large scale data set of cyber traffic at publication time, but qualified that it had many serious flaws. Of the 24 specific attack types present in the training data, only 14 are present in the test data. Therefore it seems that the two data sets are not of the same distribution. Although it is good practice to test a model on data from a different distribution, it is concerning that there is such a large discrepancy in attacks represented [8].

The training data contains an unrealistically high proportion of malicious examples, which biases any derivative model with higher false negatives and false positives when deployed [76]. An alarming 78% of training examples and 75% of test examples are duplicated, further biasing any trained model [8].

There is also a concern that some examples are experimentally much more difficult to classify than others. Further, if examples in the test data set are rated by difficulty to classify, we find that 86% of the training set and 98% of the test set are extremely easy for standard algorithms to classify. This imbalance makes it impossible to compare IDS for false positives and false negatives.

Additionally, [77] notes that the feature *time to live* follows as a different distribution in the attack and normal examples. This difference may be indicative of a recording error and may be easily exploited by classifiers.

NSL-KDD Data Set.

NSL-KDD is an updated version of the flawed KDD CUP 99 data set provided by [8]. Their revisions include removal of redundant records, reducing the training set by 78% and the test set by 75%. Additionally, the proportion examples that are difficult to classify was increased in the training and test set. The increased difficulty leads to more varied classification performance across different NIDS models. Despite these improvements, the new NSL-KDD data set does not provide modernized attack types since the 1999 predecessor. Further, the data set does not maintain temporal integrity of the original network audit [8].

UNSW-NB15 Data Set.

A 2015 effort by [77] acknowledges that KDD CUP 99 and NSL-KDD fail to represent the modern cyber threat environment. The critique specifically cites a lack of representation of *low footprint* attacks. Unlike the prior two data sets which are derivatives of collected cyber traffic, the UNSW-NB15 data set is a hybrid that also includes traffic from a simulated network. The simulation was implemented on the

Perfect Storm IXIA and the resulting data contained 49 engineered features derived from packet level data in the *Pcap* format [77]. The data set contains more than 175,000 training examples and more than 82,000 test examples. 25% of the data reflects nine modern low footprint attacks; the rest reflects modern normal traffic [11]. Despite these efforts, the UNSW-NB15 data set is aging and new attack types must be addressed. Future data sets should specifically reflect the four emerging strategies of adversarial attacks described by [18].

Recent Research.

The imbalance of traffic type in cyber data is a well known deficiency of KDD CUP 99. The imbalanced data biases classifier algorithms with inflated false positives and false negatives. This problem is addressed by [76], who uses GANs to generate attack records and rectify the imbalance. They show that it is feasible to generate realistic cyber data and that the augmented data sets improve the performance of IDS classifiers. Unfortunately, the method is only demonstrated for the binary decision of attack verses normal and does not attempt multi-class classification. Despite the benefit of augmenting the KDD CUP 99, we must note that cyber attack techniques have evolved in the past 20 years and KDD CUP 99 may not be a realistic representation of today's cyber landscape [76].

Rossow et al offers four common pitfalls of experiments that study malware including *correct data sets*, *transparency*, *realism*, and *safety*. Six guidelines apply to correctness of a dataset:

- Control the balance of normal examples and malicious examples. Including normal examples helps measure false positive rate but may distort measurements of false negative rate.

- Control the distribution of malware types. When specific varieties dominate training or test data, the detector quality may suffer and results may be uninformative.

- Decide whether unseen malware types should be included in the test data or if the test data should reflect the distribution of the training data.
- Ensure that malware does not interfere with accuracy of the monitoring module. If applicable, reports should discuss precautions taken to ensure monitoring systems are not afflicted by the malware, and whether these efforts are successful.
- Discuss aspect of the experiment that contribute to unrealistic or biased results. These aspects may include measurement issues, assumptions, and limitations in the test.
- If malicious examples are blended into a data set with normal examples, ensure that the combination reflects real world traffic [78].

From a holistic perspective, these recommendations seem to agree with the Chandola's principles of anomaly detection [29] discussed in 2.1.

Garcia et al addresses the issue of comparing the performance of botnet detectors that have been introduced in previous works but tested in with incomparable methods. Most IDS research employ proprietary data sets, in-house pre-processing, and poorly documented procedures[79]. Garcia uses best practices provided by [78] and details the steps taken to generate new data sets; the repository is hyperlinked to their manuscript.

2.5 Distributionally Robust Optimization

Distributionally robust optimization (DRO) accounts for a deeper level of uncertainty when the true distribution is not known. This larger problem seeks to optimize parameters such that the worst case performance across all sub-problems is a minimum. These sub-problems are differentiated by the probability distribution of random variables in the objective function, where the distributions are similar, but unique, and belong to the same *ambiguity set* [80]. Solutions to the DRO problem tend to be more robust than previous optimization techniques [81]. Distinguished DRO researcher Daniel Kuhn claims that DRO have the following benefits:

1. High fidelity models that account for estimation error
2. Performance is underestimated in the model, so real world trials do tend not to disappoint
3. Most problems simplify to a convex, tractable model
4. Protection against out of sample data
5. Protection against low probability events
6. Regularization effect
7. Justifiable by elegant axiomatic theory

Consider an inventory problem where we wish to maximize profit for an unknown demand. Previous methods have solved this problem for a known distribution, $\Phi(\xi)$ as described in Equation 31.

$$-cy + r\int_0^\infty min(y,\xi)d\Phi(\xi) \tag{31}$$

One approach of constructing an ambiguity set, $\mathcal{U}$, is to accept the two statistics from the available demand data, namely the mean and the standard deviation [80]. Then the new optimization problem is a mini-max, given in Equation 32

$$\inf_{f} \sup_{\mathbb{Q} \in \mathcal{U}} \mathbb{E}_{x \sim \mathbb{Q}}[\ell_f(x)] \tag{32}$$

where we denote the loss function under prediction $f(x)$ as $\ell_f(x)$. The key to effective DRO is to select an ambiguity $\mathcal{U}$ set such that the distributions in $\mathcal{U}$ diverge only slightly from the original estimate, but that $\mathcal{U}$ is also large enough that it contains the true distribution with probability $1-\delta$. $\mathcal{U}$ is often described as a ball whose radius is selected according to risk threshold. Wasserstein distance is commonly used to identify the ball for DRO in machine learning applications. This knowledge leads to one approach of robustly training machine learning models that are hardened against adversarial examples generated with Fast Gradient Sign Method [82].

Adversarial training with DRO has been theorized to prevent IDS evasion via adversarial examples. It exhibits increased robustness against adversarial joint perturbations committed on the entire training set, as apposed to just point perturbation. That is, protection is given for such adversarial examples that follow a joint distribution within ϵ-ball$\leq \delta$ in reference to the true distribution of data. Unfortunately, it is yet to be shown how to estimate the ϵ-ball that contains the worst case perturbed joint distribution [82]. The theoretical work in DRO Wasserstein training for machine learning algorithms supports the empirical evidence that it provides a regularizing affect against overfitting a distribution [83].

Table 1. Quick reference table of notable manuscripts that cover key topics

	Machine Learning			Computational Bayesian Statistics	Optimization	Cybersecurity			Policy
Source	Discriminative ML	Generative Methods	Meta-learning	MCMC	Robust Optimization	Adversarial Attack	NIDS	Cyber Data	Defense
Alhajjar 2020	■				■	■	■	■	
Anderson 1972							■		■
Anderson 1980							■		■
Applegate 2013							■		■
Axelsson 2020							■	■	
Bejtlich 2013							■	■	
Biggio 2013	■					■			
Chale 2021	■	■		■				■	
Chandola 2009		■	■				■		
Chernikova 2019	■					■			
Cui 2014	■		■						
Di Mattia 2019		■							
Fahlman 1983		■							
Finn 2017	■		■						
Gao 2017					■				
Gelman 2013				■					
Goodfellow 2014		■							
Goodfellow 2017		■							
Hinton 2006		■							
Jebara 2012	■	■							
Li 2020	■	■				■	■	■	
Liu 2018						■	■	■	
M. Tavallaee 2009							■	■	
Maxwell 2019	■						■	■	
McHugh 1999								■	
Moustafa 2015							■	■	
NDS 2018									■
Rosenberg 2021						■	■	■	
Rui 2017					■				
Scarf 1958					■				
Staib 2017					■	■			
Stolfo 2000								■	
Szegedy 2013	■					■			
Woods 2020			■						
Xu 2019		■							

III. Generate Realistic Synthetic Cyber Data

This chapter has been published in *Expert Systems with Applications Volume 207* [1]. It is included below with permission from the publisher according to their author rights agreement.

3.1 Introduction

Cyberspace has characteristics that differ from the air, land, maritime, and space domains. These characteristics affect how the military operates and defends cyberspace infrastructure, information, information systems, and data. Military forces have integrated and synchronized cyberspace capabilities along with the authorities to conduct effective cyberspace operations as part of an overall combined arms strategy in support of multi-domain operations and joint all-domain command control. Cyberspace operations provide the capability to process and manage operationally relevant actions, allowing simultaneous and linked maneuver, in, through, and across multiple domains and the information environment (IE), while engaging adversaries and populations directly across time, space, and scale. The IE is the aggregate of individuals, organizations, and systems that collect, process, disseminate, or act on information [84]. To manage the complexity of the cyberspace domain, the military has divided it into separate layers, to include physical, logical, and cyber-persona [85]. Connections between the layers of cyberspace generate a portion of the IE that is divided into three dimensions – physical, informational, and cognitive; each dimension is associated with a specific layer of cyberspace for which the latest artificial intelligence (AI) and machine learning (ML) technology can be integrated.

Cyberspace operations, in conjunction with AI and ML enhanced cyberspace infrastructure, make it possible to connect sensors directly to shooters independent of human control. Current military networks can best be described as sensor-to-human-to-shooter with this construct used to keep a human decision-maker as the final arbiter regarding the use of lethal force. Cyberspace infrastructure is network agnostic as it supports all users. From intelligence to fire control to aviation to mission command, all networks of purpose defined by the information to be exchanged can simultaneously coexist within the framework of cyberspace infrastructure. In a future with more prevalent semi-independent and independent information-centric technologies that leverage AI and ML, competitive advantage is gained since network connections will be ad hoc and information exchange and interconnectivity will fluctuate at speeds beyond human abilities to manage and control.

Increased digitalization, a proliferation of new sensors, new communication modes, the internet-of-things and virtualization of social communication have contributed significantly to formation of the military's Internet of Battlefield Things (IoBT). These AI and ML technologies serve as the pivot around which IoBT big data will be turned into actionable insight and knowledge and, ultimately, an information advantage for the military. As a component of the IoBT, intrusion detection systems (IDS), and specifically IDS embedded inside a network (NIDS), must detect, evaluate, and respond before a human operator may understand and react. These technologies enable the collection of synchronized, real-time capabilities to discover, define, analyze, and mitigate cyber threats and vulnerabilities without direct human intervention [86]. Quality data is needed to develop an ML-based NIDS that learns and manages network topologies, identifies and manages trusted users, detects network anomalies, identifies threats, and undertakes mitigation and response action. However, availability of and access to labeled cyber data is extremely limited, highlighting the need to leverage novel generative methods for deriving realistic cyber data for training and evaluating ML-based NIDS.

3.2 Literature Review

In this section, a broader cybersecurity context is provided, to include issues and progress in cyber data. Related works in data augmentation, data sampling, and generative methods are discussed.

Cybersecurity.

The military considers cyberspace to be a domain of operations which is susceptible to military targeting much like other domains of operations [87]. Cyber technologies permeate through today's battlefield to facilitate competencies such as command and control, communication, and navigation [86]. These technologies are undergoing a transformation as the IoBT emerges as a paradigm of war fighting [88]. The IoBT is the connected system of cyber-physical sensors, computers, and actuators used to carry out warfare [89]. This strategy is poised to shorten latency in intelligence collection, decision making, and effect delivery by adding autonomy throughout information pipelines. Smart devices increase human productivity and improve implementation of the observe–orient–decide–act loop [89]; autonomous devices replace humans from *in the loop*, instead allowing humans to supervise from *on the loop* or observe from *out of the loop* [90]. The IoBT carries with it a new cyber exposure that is only starting to be understood. Countless connected devices with sensors and actuators are vulnerable targets when not properly secured and monitored [91].

Cybersecurity is a discipline associated with promoting "integrity, availability, and confidentiality" of information on digital systems [89]. Data integrity ensures that data is modified only by authorized users and system integrity ensures that system functionality is unimpaired. Availability describes a system that is accessible to authorized users upon request. Systems exhibit confidentiality when they correctly deny illegal attempts to access private information [92]. Early worms sought to exploit computer resources across networks and the first virus was an academic exercise that demonstrated intentional damage to infected computers. Later examples of malware were developed to criminally manipulate, steal, or destroy data and to degrade systems [93].

Early surveys of cyber threats [19, 20] were sponsored by United States military organizations and concluded that cyber systems were vulnerable to attack and required preemptive security functionality such as auditing behavior, and enforcing tiered privileges. The framework for automating intrusion detection is developed in [94, 95]. [22] and [30] provide comprehensive reviews of intrusion detection technology. The United States Air Force deployed the first known NIDS in 1988. Today, NIDS are a primary component of the broader practices in network security monitoring. NIDS may be signature based, meaning they seek to identify patterns associated with known threats, or anomaly based meaning they detect abnormal behavior.

Issues and Progress in Cyber Data.

NIDS data is collected though combination of physical taps and specialized switches installed in critical network locations. Additionally, data collection can occur within host devices such as clients, servers, and workstations, however, these software based strategies are less reliable in the long term. Cyber data *collection applications* include Argus, Netsniff-ng, PRADS, Suricata, and Bro (recently re-branded as Zeek). The specific data being recorded may be statistical meta features of cyber packet headers or, in some cases, the entire packet headers and payloads. Zeek also belongs to the class of applications called *presentation tools* which presents live and recorded cyber data to operators and generates alerts. Best practice is for a human analyst to interpret all alerts and oversee a response [28]. This review process produces modern, relevant cyber data which is labelled as either legitimate or malicious. High quality labelled cyber data of this type is a scarce commodity in the cyber research community [96].

There have been many attempts to generate cyber data sets for IDS research and development. [8] and [75] provide constructive critiques of the ubiquitous KDD-CUP cyber data set [97]. Notable issues of the data set include spurious repetitions and biased ML training. Both KDD-CUP and the improved NSL-KDD data set [8] are still used in modern NIDS research [98, 2]. Many other data sets are reviewed by [30] and [99]. Each manuscript concludes that data sets become deprecated almost as soon as they are published because cyber strategies are constantly adapting to IDS. Further, many data sets lack the statistical information and realness required to train a NIDS. In particular, modern malware oriented at the IoBT does not express its malicious behavior until deployed in a realistic setting [99]. This implies that data sets should be built by collecting data in a real life or a realistic simulated setting. The most skilled hackers understand the behavior that triggers alerts from widely distributed NIDS [92]. This implies that NIDS require regular updates.

According to [30], NIDS training data sets should be expandable over time and easily accessible to the research community. Data should include realistic background traffic. Larger training sets provide a degree of robustness against hackers attempting to defeat NIDS. The study found approximately 11% of NIDS in their review were trained with simulated data [30]. The task of feature selection for cyber data is complicated by the broad range of data types utilized for IDS. [92] recounts *Network Security Monitor*, the first IDS to use network flow data. This practice is continued today with network security applications such as SNORT and Zeek [92].

[39] collect real, labelled cyber data and report the performance of NIDS ML classifiers when they are trained on a variety of engineered features. The U.S. Army Engineer Research and Development Center (ERDC) maintains the Cybersecurity En-

vironment for Detection, Analysis, and Reporting (CEDAR) database of network traffic passing on the Defense Research and Engineering Network (DREN). [39] queried CEDAR and selected a data set of 250,000 cyber alert records which have been labelled by analysts as either normal or malicious. 16 data fields from this set were judged to be potentially important to train ML models for NIDS. Another study introduced a novel technique called attribute ratio, which yielded better classification accuracy than existing statistical approaches to feature engineering [100].

Data Augmentation.

In the image domain for computer vision, the lack of diverse training examples may yield high variance machine learning models. That is, validation testing demonstrates an undesirable dependency on the specific training examples used and the final model. Models that suffer from high variance are unable to generalize well to unseen test examples. Techniques such as dropout, batch normalization, transfer learning, pre-training, meta-learning, and data augmentation have been shown to ameliorate variance in models [101]. Data augmentation, in particular, provides additional data examples with similar, but modified information compared to the baseline examples. Typically, models perform better when they are provided additional high quality data points from the target distribution. The augmented data sets allow machine learning models to generalize patterns more effectively. Basic data augmentation techniques in the image domain include kernel filters, geometric transformations, random erasing, color space transformations, and image mixing. Additionally, deep learning augmentation methods such as adversarial training, neural style transfer, and generative adversarial network perturbations have been used for data augmentation [101]. Both basic and deep methods for data augmentation have performed well in empirical studies [101, 102].

Data Sampling.

Data is often procured with suboptimal class distributions for training machine learning models [103]. Data oversampling methods are used to reduce model variance and they have been applied to a variety of domains. Oversampling is especially useful in the case that minority class is not well represented in the training data and in applications where misclassifying the minority classes (false positive & false negative) are more damaging than misclassification of the common classes [101, 103]. [104] investigates oversampling the minority class, undersampling the majority class, and a focused oversampling of minority class among points that lie near the decision boundary. While both undersampling and oversampling were demonstrated to improve classification accuracy, there was no additional benefit for the focused oversampling method. Recognition is another approach sought to perform positive class recognition on the majority class if the training set has a class imbalance. The recognition approach is only useful, however, for the majority class; nothing is learned on the minority classes [104]. Relevant background on single class recognition and anomaly detection can be found in [30, 105, 32]. [103] demonstrate how Synthetic Minority Over-sampling Technique (SMOTE), reference [106], and its contemporary variants use k-nearest neighbors to generate synthetic labelled points that improve the decision boundaries of ML classifiers. The synthetic examples generated by SMOTE are convex combinations of training data and, therefore, cannot extrapolate beyond the convex hull of neighboring points. SMOTE also does not enforce any conditional relationship for categorical variables. [103] expand on these methods by incorporating a stack ensemble to learn the best combination of classifier-sampler pairing for improved classification performance on highly imbalanced data sets. They find that the stack ensemble does provide good classification on a clean, labelled data set but it is yet to be shown why certain pairings perform well or how they will perform outside the laboratory setting. [107] builds on the concepts of SMOTE oversampling but performs the sampling within the latent space of an autoencoder. These sampled points in the latent space are then filtered to return only those within a specified distance to the decision boundary. Still within the latent space, the k-nearest neighbors to the sampled points are returned. The neighbors are decoded to the $\mathcal{X}$-space where they are recorded as samples. Generative models go beyond sampling techniques to model the entire joint distribution of data so realistic examples can be synthetically generated from the complex distribution.

Generative Methods.

Although previous academic studies [2, 30, 11, 108, 8] have demonstrated the great potential of using ML techniques for network intrusion detection, they typically use low quality, outdated data. Adapting models from the laboratory to the real world is especially challenging due to the lack of real world labelled training data [96]. The conventional approach to machine learning is to train a model that models a conditional relationship of class y given an input vector $\mathbf{x}$, but it does not model the overarching joint distribution of a population. These types of models are known as discriminative models and have been utilized in the field of statistical learning for decades. A model that does capture the joint distribution of training data is necessary for synthetic data generation. These models are collectively known as generative methods. A significant volume of recent work in generative modelling is in the field of machine learning [51, 102], though other generative methods are rooted computational Bayesian statistics [64, 63].

Generative models are appealing in order to achieve high classification performance [51, 32] and further, to generate desperately needed training data for discriminative models, particularly deep models that require large data sets [109]. The computational Bayesian approaches of generative methods have been a concentrated area of research since the early contributions of [110, 64]. Markov Chain Monte Carlo (MCMC) generative methods explore the model's parameter space $\boldsymbol{\theta}$ for values that maximize the correctness of the posterior distribution $p(\boldsymbol{\theta}|y)$ with respect to a target distribution. With each sampling of $\boldsymbol{\theta}$, the correctness of the current posterior is assessed and the step is either accepted or rejected. This process is repeated until a stopping criteria is satisfied [63, 68]. MCMC techniques possess the Markovian property which states convergence to a stationary distribution is theoretically attainable regardless of the initialized values $\boldsymbol{\theta}_0$. Gibbs Sampling is the most basic form of MCMC while Metropolis-Hastings MCMC is a more advanced variant that uses probabilistic acceptance of steps. This strategy of probabilistic acceptance provides the Metropolis-Hastings algorithm the property of *detailed balance* [65]. With detailed balance satisfied, the algorithm is guaranteed to obtain a posterior distribution matching the target distribution in a finite number of steps. Research suggests the Metropolis-Hastings algorithm is preferred for many systems because it is less likely to get stuck in local minima [68].

[109] find that in practice, the Metropolis-Hastings algorithm is not computationally tractable due to the low chance of accepting each step when there are many parameters $\boldsymbol{\theta}$, such as when modelling high dimensional cyber data. [109] suggest further work in generative techniques for synthetic cyber data should focus on generative machine learning. The Boltzmann Machine is a fundamental achievement in generative machine learning used to approximate the distribution of observed variables by learning a model's latent variables [61]. For discrete distributions, the Boltzmann Machine is a universal approximator of data [102]. [111] drew inspiration from cognitive science to improve the structure of the Boltzmann Machine. Denoted as the *Restricted Boltzmann Machine*, this model is bipartite, with no connections between

nodes on the observed layer or between nodes in the latent layer. This new structure captures complex relationships between variables and can be stacked for additional capacity. The conditional distributions of variables can be derived and sampled from the trained Restricted Boltzmann Machine. Variational autoencoders are a generative deep learning technique that uses gradient methods to learn parameters [102, 112]. Parameters are trained by maximizing the variational lower bound of data in the training set. New data examples can be generated from a trained variational autoencoder by drawing a random sample $\mathbf{z}$ from the distribution $p_{model}(\mathbf{z})$ and then passing $\mathbf{z}$ through the layers of the autoencoder g. The generator produces output sample $\mathbf{x}$ taken from $p_{model}(\mathbf{x}; g(\mathbf{z})) = p_{model}(\mathbf{x}|\mathbf{z})$ [102]. Another deep generative method, generative adversarial networks (GANS), provide a breakthrough for synthetic data generation [53]. As the name suggests, the architecture is comprised of two neural networks competing in an iterative game. A generator network, g, generates synthetic data $\mathbf{x} = g(\mathbf{z}; \boldsymbol{\theta}^{(g)})$, where $\boldsymbol{\theta}^{(g)}$ are the learned parameters of the generator. The other network is discriminator, d, which learns to classify synthetic examples x as real or synthetic via the operation $d(\mathbf{x}; \boldsymbol{\theta}^{(d)})$, where $\boldsymbol{\theta}^{(d)}$ are the learned parameters of the discriminator. The competing networks are simultaneously trained via gradient backpropagation. Initially, the generated examples are poor quality and the discriminator easily detects synthetic examples. Through continued training, the generator produces higher quality synthetic examples with likeness to the real data. Even as the discriminator becomes a better critic of synthetic data, the well trained generator can produce synthetic data that is indistinguishable from the real data. The fully trained generator is then used to draw synthetic examples from the joint distribution of the real data.

[109] proposed using generative methods to expand on the limited supply of high quality cyber data. Markov Chain Monte Carlo generated realistic continuous features

but proved ineffective with high dimensional data sets [109]. The conditional tabular generative adversarial network (CTGAN) and tabular variational autoencoder (TVAE) demonstrated capability to synthesize continuous data in [109]. CTGAN and TVAE demonstrated capability to generate synthetic data of both continuous *and* discrete high dimensional data in [4]; however, these studies used the obsolete KDD-CUP and NSL-KDD data sets which are not suitable to train a modern NIDS. Our manuscript explores whether high quality, synthetic, data generated from generative models can serve as a proxy for real data when training ML-based NIDS.

3.3 Methodology

The methodology applied in this research addresses three key issues that plague current cyber data sets: data quality, data currency, and metric-based evaluation.

1) Data quality: The realness, relevancy, and recentness of baseline data used in this approach can be guaranteed because it is obtained from actual cyber traffic on the DREN. The baseline data contains large quantities of background data packets as well as alert data. The Zeek NIDS in DREN uses specialized scripts and algorithms to flag alerts and severity levels in potentially malicious traffic. A computer incident response team (CIRT) adjudicates the alert packets as normal or bad. Although the alert threshold is set conservatively low, it is impossible to make any conclusions about false negatives (i.e., malicious packets that were not flagged for human review). Therefore, the label assigned by the human analyst is considered truth. Synthetic data generation seeks to provide additional data examples of the same statistical distribution, and therefore same high quality as the real data.

2) Data currency: The cyber data queried from DREN can be updated and expanded iteratively. The network's CIRT provides continuous monitoring and new queries can produce new cyber records as they become available. Additional data

fields can be incorporated into updated data sets as well, assuming they can be captured by Zeek. Synthetic data generation is used to increase the quantity of labelled examples. This is especially important when zero day attacks have been detected, but there are not enough examples to train and evaluate a ML classifier.

3) Metrics: The realness of synthetic NIDS training data can be assessed with descriptive statistical metrics and by comparing the performance of standard classifiers on the generated data to the real labelled data [109, 4]. These approaches are prevalent in the literature and should become the expectation in related cybersecurity research. The inverted Kolmogorov-Smirnov (KS) D statistic [113] and classification recall [2] work especially well to convey an overall quality of synthetic data. The inverted D statistic is provided by the Synthetic Data Vault package [113]. It is calculated as the average of 1 minus the D statistic for each column in a pair of data sets. As noted in Equation 33, the D statistic is calculated as the greatest observed difference between two distributions across all x [114, 115]. The D statistic is limited however in that it does not capture covariance in multivariate distributions.

$$d_N^{(i)} = \max\{|F_0^{(i)}(x) - S_N^{(i)}(x)|\} \tag{33}$$

In Equation 33, $d^{(i)}$ denotes the Kolmogorov-Smirnov D statistic specifically for column i in the data set. $F_0^{(i)}(x)$ denotes the observed distribution of real baseline data in column i. $S_N^{(i)}(x)$ denotes the observed distribution of synthetic data in column i. N is the number of synthetic observations in the sample. Critical values of d, with N observations and significance α are given by [115].

Collect Baseline Cyber Data.

All DREN traffic is monitored with Zeek. Zeek reports low, medium, and high confidence alerts for suspicious traffic detected by its community sourced scripts. Packets flagged with alerts are summarized with 73 features and stored on the CEDAR database. These records are then audited by the CIRT who adjudicates them as normal or malicious. The ERDC maintains these labelled feature vectors on CEDAR and approved researchers can query the records using the High Performance Computing Architecture for Cyber Situational Awareness (HACSAW) API. ERDC prohibits HACSAW data from being removed from the DREN network, therefore, all data analysis must be performed on the DREN.

This research follows the data collection method of [39] in which HACSAW is queried for real, labelled, logs of HTTP cyber traffic on the DREN. For the first research question in this study, the 250,000 data examples from [39] were obtained and reused. For the second research question, however, a greater number of examples were required, so a new data set was queried and pre-processed.

Details on HTTP protocol and the contents of the Zeek HTTP logs are outlined by the Zeek documentation [116]. This study also references the feature engineering method by [39], which provides guidance on optimizing features for machine learning with cyber data. Table 3 outlines which engineered features were derived from the HTTP logs and used for this study.

Table 3. Data fields and feature engineering technique.

Data Field	**Example**	**Technique**
connection	open, close, none	Label encoded
id_orig_h_cc	US	Label encoded
id_resp_h_cc	US	Label encoded
severity	H, M, L	Label encoded
method	GET, POST	Label encoded
request_body_len	10	minimax
response_body_len	10	minimax
hp	0000000000000000	16 bit vector
id_orig_h_org	external	Label encoded
id_resp_h_org	internal	Label encoded
host	my.connection.edu	Label encoded
status	normal, bad	One hot encoded

Generate Synthetic Cyber Data.

Recent research demonstrates that generating synthetic, tabular data has unique challenges that do not exist in other domains such as synthetic image generation [4, 109]. Tabular data contains variates that are continuous, or discrete, and in some cases a particular variate may contain both types of observations. Although image pixels may follow a roughly Gaussian distribution, the joint distribution of data tables do not follow any known distribution. Data tables tend to be multimodal, and existing generative methods such as a GAN perform poorly in multimodal domains. Discrete variates are typically one hot encoded, which is practical to model for univariate data. It is unclear, however, how to model joint distributions with discrete and continuous variates. Further, violations to one hot sparsity in the generated data would be easily detectable with simple rules. Finally, the categorical variates may predominately express major classes, and only express a few instances of the minor classes. Generative models may fail to capture the non-zero probability of minor modes despite converging to an optimal model [4].

Previous results from [109] indicated CTGAN, TVAE, and Markov-Chain Monte-Carlo (MCMC) with Metropolis Hastings algorithm were potential choices for modeling the joint distribution of cyber data. Our follow up trials demonstrated that only CTGAN and TVAE could be scaled for high dimensional data sets. Exploratory trials of MCMC were implemented in pymc3 [117] using Python 3.8. We found that the multivariate Gaussian mixture model (MVGMM) fit with the Metropolis-Hastings MCMC algorithm in [109] could not be adapted for more than two continuous variables, regardless of posterior distribution and hyperparameter settings. Likewise, the MVGMM fit with the No-U-Turn MCMC algorithm in pymc3 did not converge for data sets greater that two continuous variables. Therefore, CTGAN and TVAE are selected as models to synthetically generate cyber data, with details of these two generative ML methods to follow.

Generative Adversarial Networks.

Generative adversarial networks (GANs) are a machine learning framework for building generative models introduced by [53] and updated in [57] with recent insight. GANs are built using an adversarial process. On one side, a multi-layer perceptron generates data, initially with random weights, while on the other side, a discriminative MLP detects whether or not the generated data matches the distribution of a target set. Initially, the discriminator can easily distinguish generated data examples from true examples, but the generator improves its model over time. Simultaneously, the discriminator learns on its mistakes and improves its model. Eventually, the generator produces examples that match the original distribution very well, and the discriminator cannot detect a difference in the generated examples [53]. Most GANs today are based on the deep conventional GAN architecture [52] and used for image generation [53]. As alluded to in Section 3.3, creating a GAN for tabular data is a more nuanced task.

A GAN capable of modelling tabular data has been constructed [4]. The model, CTGAN, is designed to address the shortcomings of fitting traditional GANs to tabular data. The model addresses mode collapse of minor discrete classes by conditioning the training on each discrete mode. A condition generator is used to sample discrete classes and represent as a one-hot condition vector. The condition vector representation of discrete conditions is input to the CTGAN generator. This input motivates the generator to learn relationships between the condition and a realistic synthetic output for all discrete categories. CTGAN also detects modes within continuous variables and determines mode membership for each example using a Guassian variational model. Mode specific normalization facilitates learning for data examples attributed to minor modes. This information is also input to the generator. Collectively, these features of GAN address the challenges of modelling tabular data [4].

CTGAN is represented visually in Figure 4. The network represents variables with a special encoding structure to capture the conditional relationships of variables and to prevent mode collapse. The row containing both discrete and continuous variables is represented by $\mathbf{r}_j$. $\mathbf{r}_j$ contains α_{ij}, a continuous value within a mode for variable i and data example j; β_{ij}, a one hot vector indicating mode membership for variable i; and $\mathbf{d}_{ij}$ a vector specifying the discrete levels active for variable i. Mask vectors $\mathbf{m}_i^{(k)}$ are formed to encode the active discrete settings for all combinations of variables and settings where $\mathbf{m}_i^{(k)} = 1$ if setting k is active for variable i. The concatenation of all mask vectors for a data example is known as the condition vector.

The input layer of the CTGAN generator includes the condition vector and a vector of Gaussian noise. Two hidden layers of 256 nodes are fully connected with batch normalization, droput and leakyrelu activation functions. The output layers are also fully connected. The continuous output are assigned with a tanh activation function while the mode indicator vector and discrete values are assigned with a gumbelsoftmax activation function. The discriminator is a MLP with two hidden layers, and leakyrelu activation functions, that discerns whether the generated row vectors are drawn from the same population as the sampled row vectors. The parameters in the generator and discriminator are simultaneously trained with Wasserstein loss via the Adam optimizer.

Autoencoders.

Autoencoders are a deep learning framework that learns a function that summarizes input data in a low dimensional latent space and reproduces the data in the original space with minimal error. This is done by constructing a neural network with

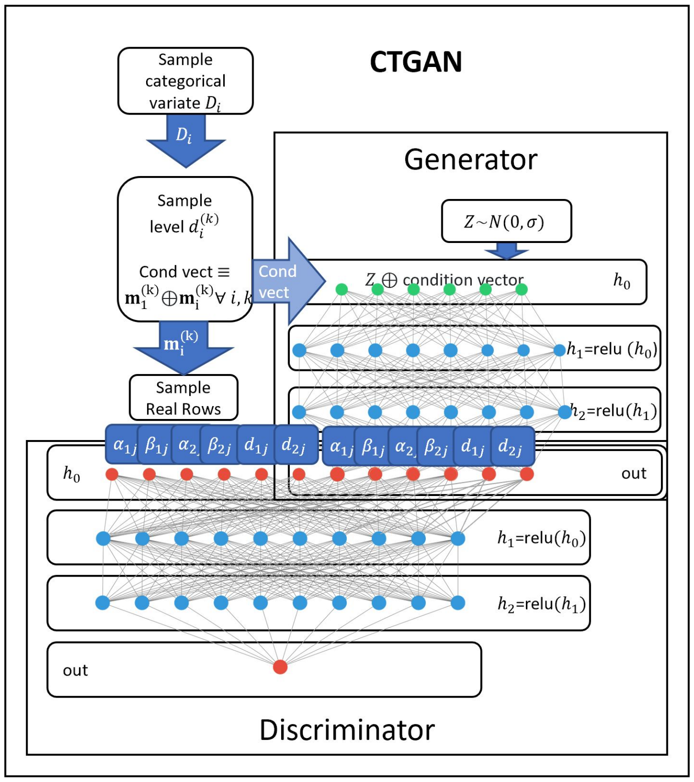

CTGAN
Sample categorical variate D_i
D_i
Sample level $d_i^{(k)}$
Cond vect ≡ $\mathbf{m}_1^{(k)} \oplus \mathbf{m}_i^{(k)} \forall i, k$
Cond vect
$\mathbf{m}_i^{(k)}$
Sample Real Rows
Generator
$Z \sim N(0, \sigma)$
$Z \oplus$ condition vector
h_0
h_1=relu (h_0)
h_2=relu(h_1)
out
α_{1j} β_{1j} α_{2j} β_{2j} d_{1j} d_{2j}
α_{1j} β_{1j} α_{2j} β_{2j} d_{1j} d_{2j}
h_0
h_1=relu(h_0)
h_2=relu(h_1)
out
Discriminator

Figure 4. CTGAN includes a generator network that is input to a discriminator network, as proposed by [4].

two sequence of layers. The first sequence of layers $\mathbf{h} = f(\mathbf{x})$ compresses the input to the latent space, and another sequence of layers $g(f(\mathbf{x})) = \mathbf{x}$ generates realistic data from the latent encoding [102].

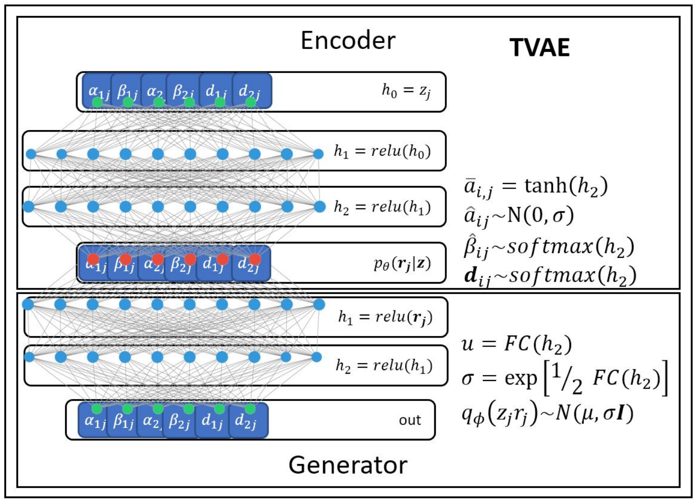
Variational autoencoders are more advanced than standard autoencoders because they are trained to maximize the variational lower bound, $\mathcal{L}(q)$, of training data [118]. This choice of loss function motivates the model to place high probability across many values of the latent variables $\mathbf{z}$ that could feasibly reconstruct $\mathbf{x}$ [102].

During training, we take the following perspective of the model. The approximate inference network $q(\boldsymbol{z}|\boldsymbol{x})$ is the encoder that generates $\boldsymbol{z}$; and the decoder is the distribution $p_{model}(\boldsymbol{x}|\boldsymbol{z})$ from which the reconstructed output is drawn [102]. The variational lower bound, $\mathcal{L}(q)$ for the sample $\mathbf{x}$ is maximized as shown in Equation 34.

$$\mathcal{L}(q) = \mathbb{E}_{\boldsymbol{z}q(\boldsymbol{z}|\boldsymbol{x})} \log p_{\text{model}}(\boldsymbol{z}, \boldsymbol{x}) + \mathcal{H}(q(\boldsymbol{z}|\boldsymbol{x})) \tag{34}$$

TVAE is a variational autoencoder designed specifically to model and generate tabular data. The autoencoder uses the same variable encoding as CTGAN to address mode collapse and capture conditional relationships. TVAE encoder contains two hidden layers with the relu activation function. An output layer contains tanh activation functions for continuous variables, α_{ij} and softmax activation functions for discrete variables, β_{ij} and d_{ij}.

The network produces one hot encoded values for discrete variables β_{ij} and $\mathbf{d_{ij}}$ in $\mathbf{r_j}$. For α_{ij}, the continuous variables, the network captures the mean and standard deviation. A generator model with two hidden layers then uses the learned distribution to generate normally distributed values for α, hence, fully defining $\mathbf{r}_j$.



Encoder
TVAE
$h_0 = z_j$
$h_1 = relu(h_0)$
$h_2 = relu(h_1)$
$p_\theta(\mathbf{r}_j|\mathbf{z})$
$\bar{a}_{i,j} = \tanh(h_2)$
$\hat{a}_{ij} \sim N(0, \sigma)$
$\hat{\beta}_{ij} \sim softmax(h_2)$
$\mathbf{d}_{ij} \sim softmax(h_2)$
$h_1 = relu(\mathbf{r}_j)$
$h_2 = relu(h_1)$
out
$u = FC(h_2)$
$\sigma = \exp\left[{}^1/_2\ FC(h_2)\right]$
$q_\phi(z_j r_j) \sim N(\mu, \sigma \boldsymbol{I})$
Generator

Figure 5. TVAE is a tabular variational autoencoder with an encoding network and a decoding/generator network, proposed by [4].

Machine Learning Classifiers.

Humans posses strong ability to interpret information and draw conclusions. In particular, humans can learn patterns allowing them to classify things into groups. Similarly, statistical models can be trained with large training sets to label unlabelled data examples. These models are machine learning classifiers, meaning the algorithm has learned the model from the data. A training data set includes vectors of observations $(\mathbf{x}_n, y_n)$ where $\mathbf{x}_n$ are the indicators and y_n is the label. An unlabelled data set includes only the indicator variables $\mathbf{x}_{n'}$, but a trained model can assign the label $y_{n'}$. Unlike regression models, which estimate a continuous response y, classification models assign a discrete class [119]. Most machine learning techniques require the data is prepared with several steps of pre-processing. Feature engineering is one such step [120]. In this research, we reference the successful engineered features for cyber data presented by [39]. There is a growing list of machine learning methods that perform classification. Several of the most pervasive methods have been selected for this study.

Logistic Regression.

Linear regression is a statistical model that captures the relationship between indicator variables and a continuous response variable [121]. The linear regression model takes the form of Equation 35 where $\mathbf{x}$ is a vector of predictor variables, $\hat{\beta}$ is the least squares estimator of the model parameters, and $\hat{y}$ is the prediction of the response variable.

$$\hat{y} = \mathbf{x}'\hat{\beta} \tag{35}$$

The model, however does not perform well for binary or categorical outputs. In the binary case, a trained model may output a response anywhere between 0 and 1, or well outside that range. These responses have no statistical interpretation, therefore, linear regression is ill suited for the binary classification problem [119]. [122] argues that the preferred form of regression for binary classification is given by Equation 36.

$$pr(Y_i = 1) = e^{\alpha+\beta_i X}/(1 + e^{\alpha+\beta_i X}) \tag{36}$$

with indicator variable x, parameters β_0 and β_1 [122]. The logistic regression formula, Equation 36 yields a response between 0 and 1 for all input.

By annotating $p(X) = pr(Y = 1|X)$ we solve for the odds function, Equation 37,

$$\frac{p(X)}{1 - P(X)} = e^{\beta_0+\beta_1 X} \tag{37}$$

which provides the ratio of the conditional probability $pr(Y = 1|X)$ and its compliment, given the estimated model.

The log-odds function, given in Equation 38

$$log\left(\frac{p(X)}{1 - P(X)}\right) = \beta_0 + \beta_1 X \tag{38}$$

has a linear relationship to X. That is, changing the parameter X by a unit, affects the log odds by β_1. β_0 and β_1 are now conveniently estimated by the method of maximum likelihood. Once the parameters are estimated, the logistic regression model is given by the probability distribution $p(X)$ shown in Equation 36 [119].

This research implements LogisticRegression in scikit-learn with the liblinear solver, which is preferred for smaller data sets. All other parameters are set to default. Notably, the scikit-learn function LogisticRegression uses L2 regularization with a weight of 10 to remove less important features from the model.

Support Vector Machine.

The support vector machine was initially presented by [123] and expanded for inseparable data sets in [124]. It works by using a kernel to project the training data into a higher dimension. In the higher dimensional representation, the algorithm searches for the hyperplane that optimally separates data points by class, and maintains a large margin between classes, and minimal violations. Equation 39 is the loss function for support vector machines which is used to find $\mathbf{w}$, the optimal separating hyperplane. The first term, $\frac{1}{2}\left\|\mathbf{w}\right\|^2$ is inverse proportional to the separating region between classes. The second term, $C\left(\sum_{i=1}^{N}\xi^{(i)}\right)$ sums the magnitude of $\xi^{(i)}$, the classification error for data examples $x^{(i)}$, which lie on the wrong side of the margin boundary. C is a user defined hyperparameter to balance the importance of the two terms. The constraints in equations 40 and 41 bind ξ to the distance of violation to the margin region. $y^{(i)}$ is the true class label and w_0 is the intercept term of the separating hyperplane [124].

$$\mathcal{L} = arg \min_{\mathbf{w}} : \frac{1}{2}\lVert\mathbf{w}\rVert^2 + C\Big(\sum_{i=1}^{N} \xi^{(i)}\Big) \tag{39}$$

$$S.T. \qquad \xi^{(i)} \geq \ 0 \ \forall i \tag{40}$$

$$y^{(i)}(w_0 + \mathbf{w}^\top \mathbf{x}^{(i)}) \ \geq 1 - \xi^{(i)} \ \forall \ i \tag{41}$$

The NuSVC function in scikit-learn uses a dual formulation to exploit computational efficiencies. By default, this function uses the radial basis function kernel, nu = 0.5, an adaptive formula for gamma, and a kernel cache of 200MB [125]. In this study, nu is set to 0.2 and the cache is increased to 500MB.

Multi-layer Perceptron.

A multi-layer perceptron (MLP) is a mathematical model translating an input vector to a response through one or more layers of intermediary functions. The model was developed over several decades with major advancements by [126], [127], and [128].

One layer of a fully connected MLP may perform the operation $\sigma(\mathbf{w}^\top \mathbf{x}) + b$ where $\mathbf{x}$ is an input vector, $\mathbf{w}$ is a vector of learned weights, b is a learned bias parameter, and σ is an activation function selected to improve training. The parameters of each intermediary functions are learned though a training process called back-propagation. Back propagation uses a gradient based optimizer to reduce the classification error, typically cross entropy. Training via back propagation is performed for many iterations until the model seizes to improve each iteration. MLPs are very versatile. The architecture of the MLP is the choice of number of nodes in a layer, number of hidden layers, and connectedness between layers [102]. These choices affect the learning capacity of the model. The greater the number of layers in a MLP, the more abstract the representation is, allowing the model more freedom to learn complicated relationships. This additional capacity also risks generating an overfit model. The Scikit-learn Python package includes a function called MLPClassifier. By default, the MLPClassifier model contains 100 hidden layers, uses the relu activation function, the adam optimizer, an L2 penalty term of $\alpha = 0.0001$, a constant learning rate of 0.001, and a maximum learning of 200 training iterations [125].

Random Forest.

Decision trees are the precursor to random forest classifiers. They are machine learning models represented by a tree-like flow graph. A data example is input to a trained decision tree at the root node. At the root node, some mathematical criteria determines a subsequent node the example flows through. The process of obeying a criteria to determine the path through the nodes continues until the example arrives at a leaf node. The leaf node corresponds to an output, or decision. [129] and [130] are regarded as two of the most significant contributions in the development of decision trees. Both algorithms seek the optimal splitting criteria in each node and yield an effective decision tree for classification [131]. There have been several significant advancements from the primitive decision tree. Random Forest is one of the most notable.

Ensemble methods leverage information learned in multiple models to classify with better accuracy. A typical strategy involves a hierarchical model that outputs the majority vote from each base classifier. The decision boundaries of ensembles may better reflect the *true model.* Empirically, greater diversity of base models provides better ensemble results. Random forest is an ensemble method that achieves diversity by randomizing the attributes selected in the base level tree classifiers. Random forests ensembles demonstrate among the best accuracy of the ensemble strategies for decision trees and they often faster and more robust to outliers in training data [132]. This study uses the scikit-learn RandomForestClassifer function which, by default, creates 100 estimators with the gini criterion. This study increases the number of estimators to 300 and uses the entropy metric for learning branch points.

Calculate Metrics of Quality.

The KS inverted D statistic was calculated for each synthetic data set of 25,000. This provided a measure of similarity of the joint distribution in the synthetically generated cyber features to the baseline data set of 250,000 real examples. Random forest (RF), support vector machine (SVM), multi-layer perceptron (MLP), and logistic regression (LR) were used as the ML classifier algorithms. Each was implemented in the scikit-learn python library with default settings except for modifications noted in Section 3.3. First, each classifier algorithm was trained with the synthetic data sets (generated from 1,000-25,000 real examples). Then, each instance of the trained classifiers was tested using the baseline set of 250,000 real examples. When the classifier was trained with real examples, those examples used for training were removed from the test set. Likewise, when the classifier was trained using synthetic data, any examples used to fit the generator were removed from the test set. Classification recall was recorded, as this performance metric appropriately reports the NIDS classifier ability to avoid false negatives.

Computational Experiments.

Computational experiments are conducted to investigate two aspects of using synthetic data to train an ML-based NIDS. In both cases, real data is used to fit generative models. Then, synthetic data is produced from the generative models. Machine learning classifiers are trained with either real data, synthetic data, or a combination of real and synthetic data. These classifiers are tested with real labelled data.

Does quantity of real training data affect quality of synthetic data set?.

This research problem leveraged the same set of 250,000 labelled real data examples collected in [39] and maintained the same engineered features. The two methods used to generate synthetic data were CTGAN and TVAE. Each method was implemented within the Synthetic Data Vault library available for Python 3.8. Instances of the generative models were fit using random subsets of the truth data of sizes 1,000, 5,000, 10,000, 15,000, 20,000, and 25,000. These scenarios reflect a situation where the number of real training examples is the primary constraint.

One-at-a-time parameter tuning did not demonstrate any obvious improvements to the inverted KS D statistic, so default settings were adopted for CTGAN and TVAE models. The training process was repeated for a second repetition with each aforementioned model. 25,000 synthetic examples were then sampled from each trained generator model. The random subset of real samples used to train the generators of each trial were recorded in order to prevent biased ML classification at test time. That is, baseline examples used to fit the generators were removed from the classification test set.

The resulting data sets were used to train LR, MLP, RF, and SVM classifiers as described in Section 3.3. Machine learning classifiers were trained using the 25,000 examples synthetically generated from the generative models, varying the number of examples to fit the generative model as outlined above. Additionally, classifiers were trained using 1,000, 5,000, 10,000, 15,000, 20,000, and 25,000 real examples.

Finally, classifiers were trained using training sets of 1,000, 5,000, 10,000, 15,000, 20,000, and 25,000 real examples, augmented with an equivalent number of synthetic examples. The recall performance of each trained classifier is reported.

Does the ratio of real and synthetic data affect classification performance?.

This experiment considers whether the classification performance of machine learning models is dependent on the proportion of data that is synthetic. For this experiment, it was determined that the set used to fit the generators and the classifier training sets should be much larger than in the previous experiment in Section 3.3. By eliminating any practical constraint on available data, stronger conclusions can be made about the research question.

A new set of real cyber data was queried from HACSAW using the method outlined in Section 3.3. 250,000 real malicious examples and 250,000 real normal examples were saved as csv files. 20% of each (malicious and normal) data set was sampled and sequestered as a test set, totalling 100,000 examples. The remaining 400,000 real examples were used to fit a CTGAN and TVAE generative model. These models were used to generate an additional 400,000 synthetic data examples. Next LR, MLP, RF, and SVM classifiers were trained with training sets ranging from 0% synthetic data to 100% synthetic data. This was done by sampling the real and synthetic data sets, without replacement, for each of the specified ratios shown in Table 4. The intent was to train models from mixed data sets as large as 400,000 examples; however, it proved computationally intractable to train a SVM classifier from very large sets. Therefore, classification was performed and performance was reported for training sets of 25,000 and 100,000.

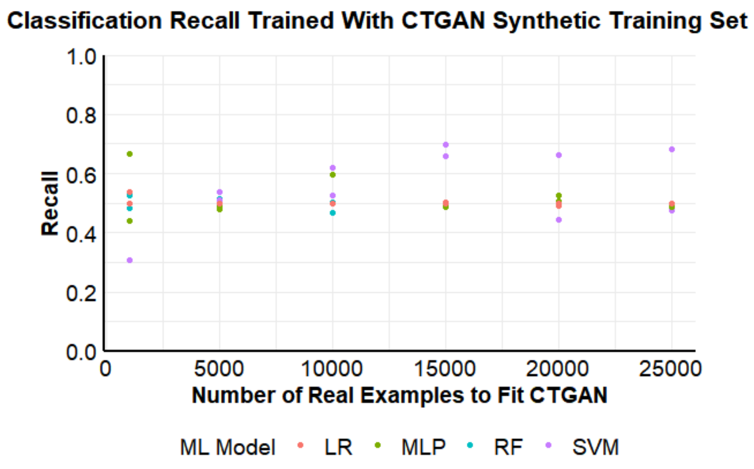
Table 4. Classification models are trained with data sets where the percent synthetic data is varied.

Trial	1	2	3	4	5	6	7	8	9	10	11
Real (%)	100	95	90	85	80	75	70	65	60	55	50
Synthetic (%)	0	5	10	15	20	25	30	35	40	45	50
Trial	12	13	14	15	16	17	18	19	20	21	-
Real (%)	45	40	35	30	25	20	15	10	5	0	-
Synthetic (%)	55	60	65	70	75	80	85	90	95	100	-

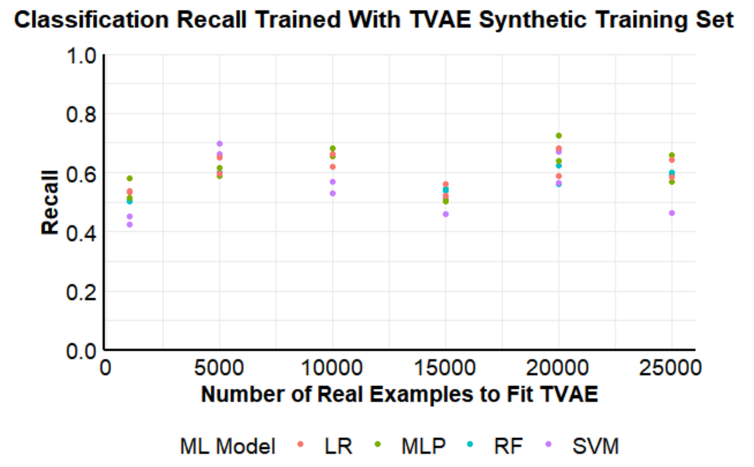
3.4 Experimental Results and Discussion

Does quantity of real training data affect quality of synthetic data set?.

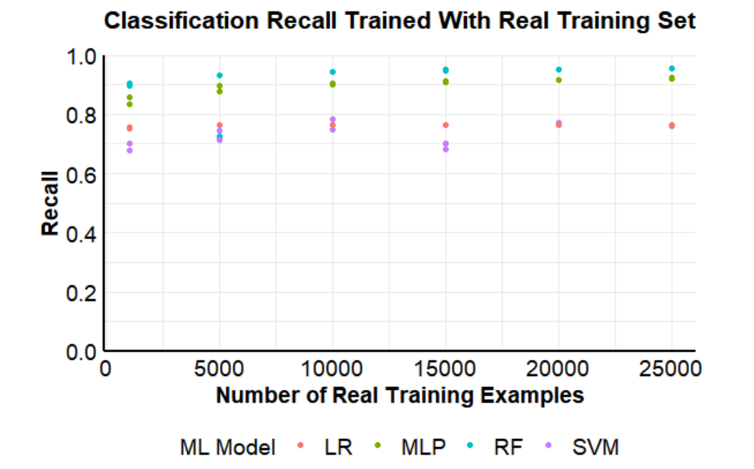
The classification recall for each ML model was tested with real labelled examples and plotted in Figure 6a for CTGAN synthetic data, Figure 6b for TVAE synthetic data, and Figure 6c for classification with real data features. Each plot contains 48 data points corresponding to four classifiers, six set sizes, and two repetitions. These results are provided in Tables 19, 20 and 21 in the Appendix. The recall of the classifiers trained on synthetic data is significantly inferior to the recall on the models trained with real examples. The effect of increasing number of training examples was investigated. The models trained with real examples demonstrate a subtle improvement in recall as the sample size increases, while the models trained on synthetic data appear to have a slight concave trend. A hypothesis test with $\alpha = 0.05$ did not conclude any significant linear trend as number of real training examples to fit the generators is increased. It is possible that the magnitude of by which number of training examples was varied was not large enough to yield a detectable difference. When real examples were used to train classifiers, the number of examples does demonstrate a significant effect on recall, with $\alpha = 0.05$.



(a) ML models trained with CTGAN generated cyber data



(b) ML models trained with TVAE generated cyber data



(c) ML models trained with real cyber data

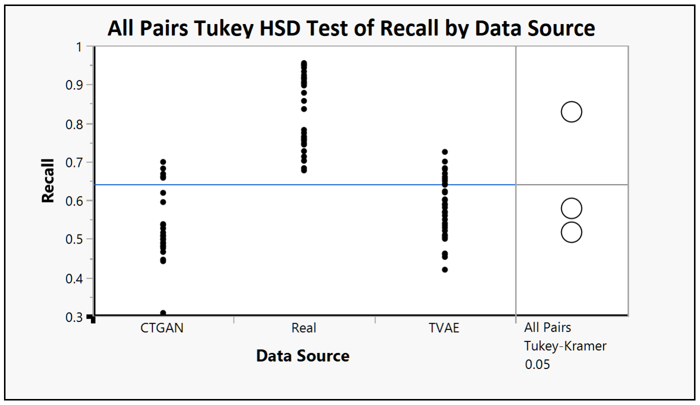
Figure 6. Classification recall of models trained with real or synthetic data.

The distributions of classification recall broken down by ML model and by data source are presented as box plots in Figure 8. Mean classification recall for NIDS classifiers trained with real data, TVAE and CTGAN were 0.829, 0.579, and 0.517, respectively. According to Tukey's all pairs HSD test, the mean recall for each training data source (CTGAN, TVAE, real) was statistically different from each other, with $\alpha = 0.05$. Figure 7a provides a graphical representation of the test. This finding somewhat contradicts the observation that CTGAN synthetic data provides a better fit to real data than TVAE synthetic data. It is therefore evident that the inverted KS D statistic is not a comprehensive metric for quality of synthetic data used to train machine learning IDS.

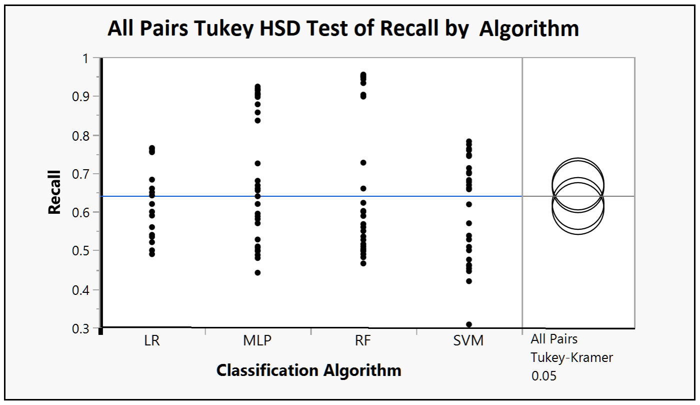
Mean classification recall for RF, SVM, MLP, and LR were 0.665, 0.608, 0.672, and 0.621, respectively, and Tukey's all pairs HSD test indicated insufficient evidence that treatment groups were different, with $\alpha = 0.05$. It is possible that the data sets used to fit generative models were too small to yield excellent classifiers with any of the selected algorithms. Figure 7b provides a graphical representation of the test.

The best observed recall among models trained on CTGAN data, reference Table 19 and Figure 8, was 0.699 produced by an SVM model. Interestingly, the worst observed recall, by far, was 0.308, also produced by an SVM model. The best observed recall on a model trained with TVAE, reference Table 20, was 0.725, produced by a MLP. Despite SVM producing the model with second best recall on the TVAE set, the five worst performing models were all SVM. By reasonable standards, no model trained on purely synthetic data demonstrated sufficient detection to be deployed as a NIDS.

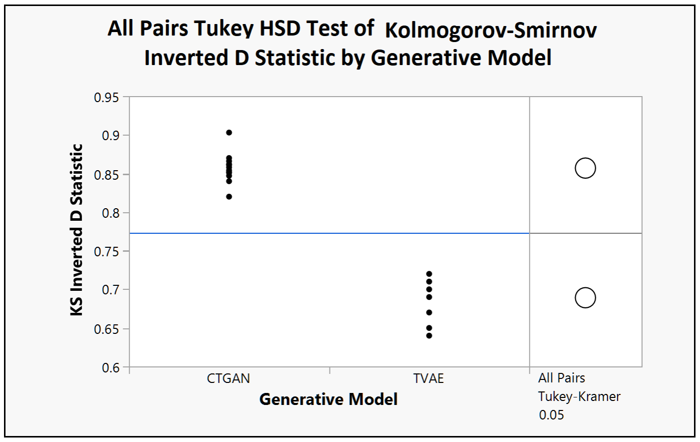
The best observed recall among models trained on real data, reference Table 21, was 0.955, the result of a RF classifier. RF models produced excellent recall, almost always above 0.9. Though slightly worse on average, MLP models also often exhibited



(a) Plot of recall of the real, TVAE, and CTGAN data on left, and connected groups with Tukey's HSD test of equivalent means on right.



(b) Plot of recall by machine learning classifier on the left, and connected groups with Tukey's HSD test of equivalent means on right.



(c) Plot of Kolmogorov-Smirnov inverted D statistic by generative model on left, and connected groups with Tukey's HSD test of equivalent means on right.

Figure 7. Significance of treatment groups on classification recall of models trained with real or synthetic data.

recall better than 0.9. SVM produced the five worst performing models with recall as low as 0.677.

Figure 9 presents the KS inverted D statistic as a function of training examples for each generative method employed and these results are tabulated in Table 22. Larger values of the inverted D statistic indicate likeness of paired variable's between two data sets (real and synthetic). The reported values are the average of the inverted D across all variables in the cyber data sets. The number of real examples used to fit TVAE and CTGAN generative models demonstrated a minor effect on data realness, as measured by the KS inverted D statistic for a comparison between synthetic and real data. Specifically, the real data benchmark included all records from the 250,000 baseline records except those used in fitting each particular generative model.

An effects test with the T distribution provides statistical evidence that increasing the number of real examples to fit TVAE also increases the KS inverted D statistic, where $\alpha = 0.05$. The results were inconclusive when performing the same effects test for the CTGAN results.

The mean inverted D statistic for data generated by CTGAN was 0.857, and it was 0.689 for data generated by TVAE. Tukey's all pairs HSD reports that there was a statistical difference in mean inverted D between generative methods. Although [109] previously reported an inverted D statistic for CTGAN, TVAE and Markov Chain Monte Carlo generative methods of 0.890, 0.723, and 0.875, respectively, these reflect a different cyber dataset containing only one discrete variable and two continuous variables while only training the generators with 10,000 examples. Therefore, this suggests it is significantly more difficult to achieve the same goodness of fit in generating synthetic cyber data when working with higher-dimensional data sets. The HSD Tukey test is shown graphically in Figure 7a.

The evaluation of recall was extended to classification models trained by a mix-

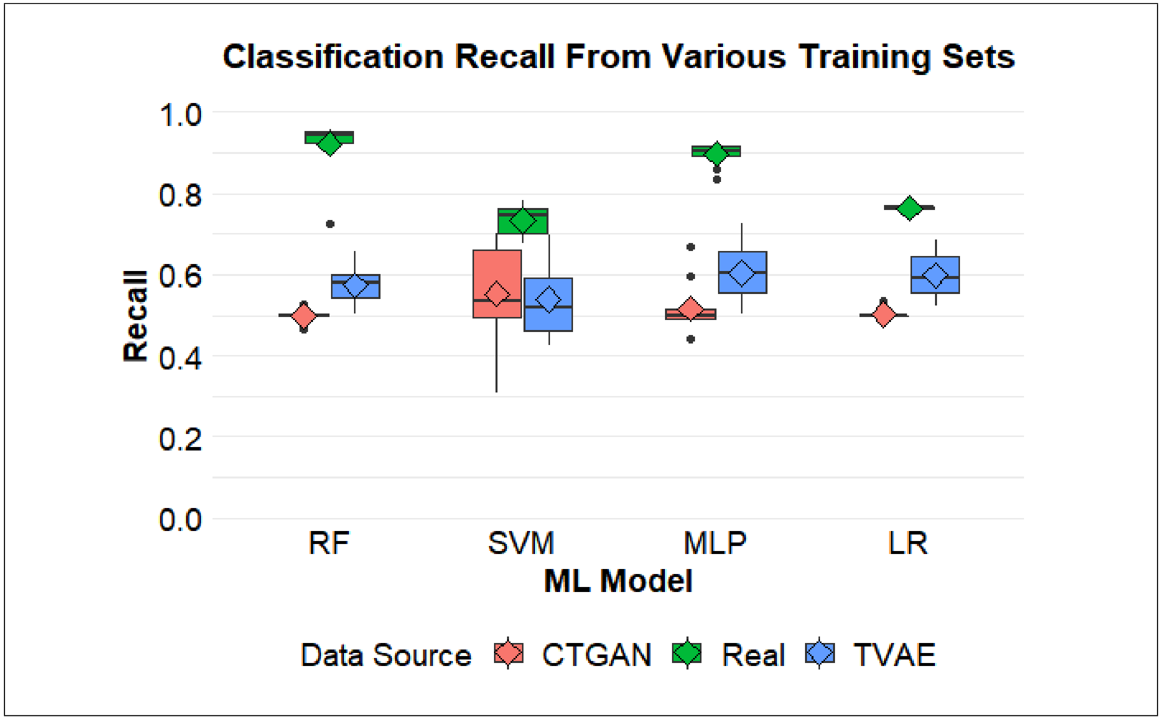


Figure 8. Classification recall by ML algorithm and data set.

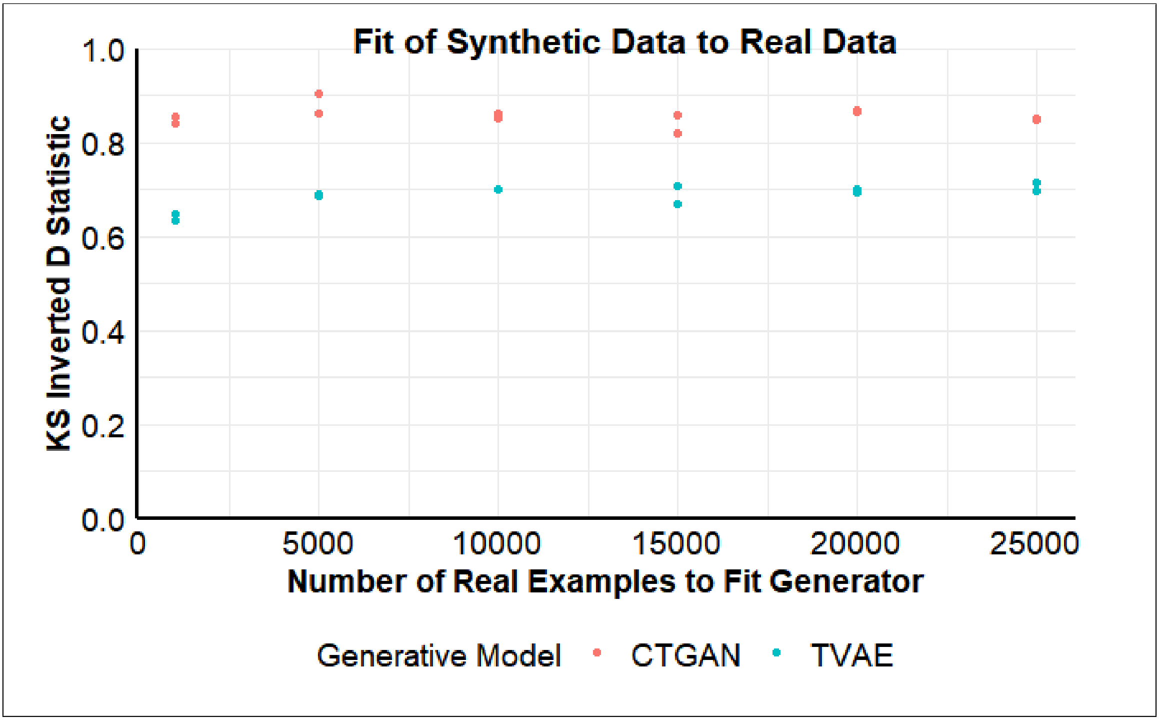


Figure 9. Similarity between real and synthetic data sets.

ture of half real and half generated data. This strategy depicted scenarios where 1,000, 5,000, 10,000, 15,000, 20,000, and 25,000 real examples were available. By incorporating an equal quantity of synthetic data, the models were trained with 2,000, 10,000, 20,000, 30,000, 40,000, and 50,000 total training examples. An effects test was performed using the T distribution and $\alpha = 0.05$. The test concluded that there is no statistical evidence that the number of real examples to fit the generator was important for the classification recall of machine learning models. This was true for the half real, half synthetic mixed datasets derived from both CTGAN and TVAE.

Figure 10a presents the recall for models trained with equal quantities of real examples and synthetic data generated from CTGAN and the results are tabulated in Table 23 in the appendix. The nine best performing models were all RF, each achieving a recall of at least 0.932, even when the number of real training examples were as few as 5,000. The nine worst performing models were SVM.

Figure 10b presents the recall for models trained with equal quantities of real examples and CTGAN synthetic examples. These results are tabulated in Table 24. The top 10 performing models were all RF, each achieving a recall of at least 0.929. No RF model had a recall below 0.896. Only one ML and one LR model demonstrated recall surpassing 0.9. SVM was by far the worst performing classifier for data sets combining equal proportions of real and TVAE generated data.

Tukey's all pairs HSD test with $\alpha = 0.05$ was used to test statistical difference between the mean recall of classifiers trained with real data, CTGAN data, TVAE data, real data augmented with CTGAN data, and real data augmented with TVAE data. The test indicated no evidence that the classifiers trained with mixed data sets performed differently than the models trained with real data. The classifiers trained with only synthetic data had significantly worse recall. Table 5 presents the connection table of mean recall for all groups under test and Figure 11 provides a graphical representation of the test.

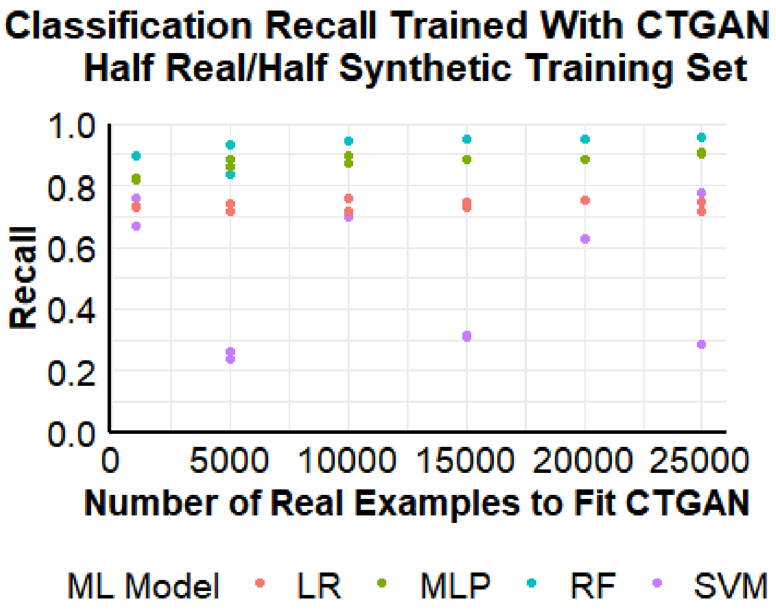
Does the ratio of real and synthetic data affect classification performance?.

Classification recall is presented in Figure 12 for classifier models where the training data is a mixture of real and CTGAN generated data and the percent synthetic is varied. Similarly, the recall for models trained with a mixture of real and TVAE generated data is presented in Figure 13. The experiment was performed once with training sets containing 25,000 examples, Figures 12a and **??**, and again with training sets containing 100,000 examples, Figures 13b and 13a. Each plot contains 84 data points corresponding to four classifiers, crossed with 21 settings of percent synthetic. The full results for these trials are presented in Tables 25-28 in the appendix.

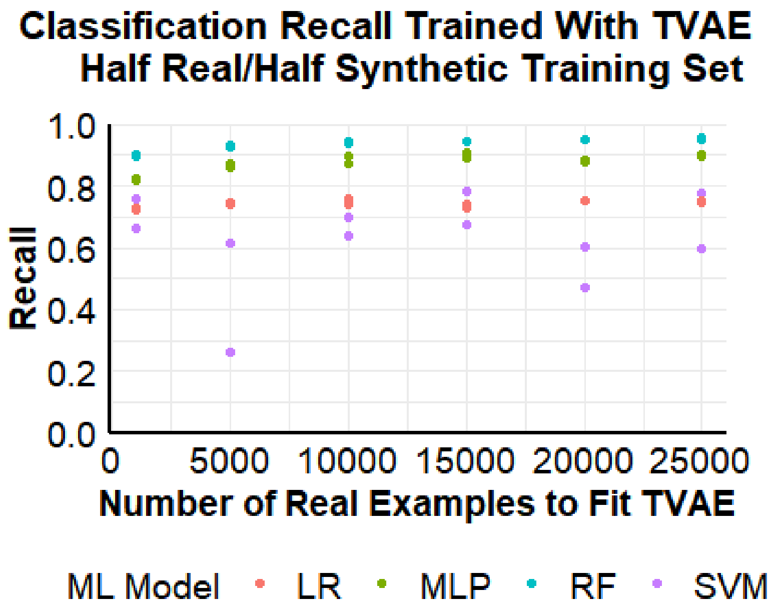
An effects test with $\alpha = 0.05$ determines that *percent synthetic* is an important factor for estimating recall. Further analysis indicates that *percent synthetic* may also have a quadratic effect, however the quadratic term is not conclusively significant on all data sets, with $\alpha = 0.05$, and the model adequacy checks suggest that residuals may not be independent. The results indicate that varying the training set size between 25,000 and 100,000 does not have a significant effect on recall. The mean recall of models trained on 25,000 examples (Table 25 and Table 27) is 0.715 and the mean recall of models trained on 100,000 examples (Table 26 and Table 28) is

Table 5. Connection table of recall by data source using Tukey HSD All Pairs test with a confidence of 0.95.

Level	Group		Mean Recall
Real	A		0.82887500
TVAE Mixed	A		0.79562500
CTGAN Mixed	A		0.76743750
TVAE		B	0.57883333
CTGAN		B	0.51697917



(a) ML models trained with CTGAN generated cyber data



(b) ML models trained with TVAE generated cyber data

Figure 10. Classification recall of models trained with a mixture of real and synthetic data.

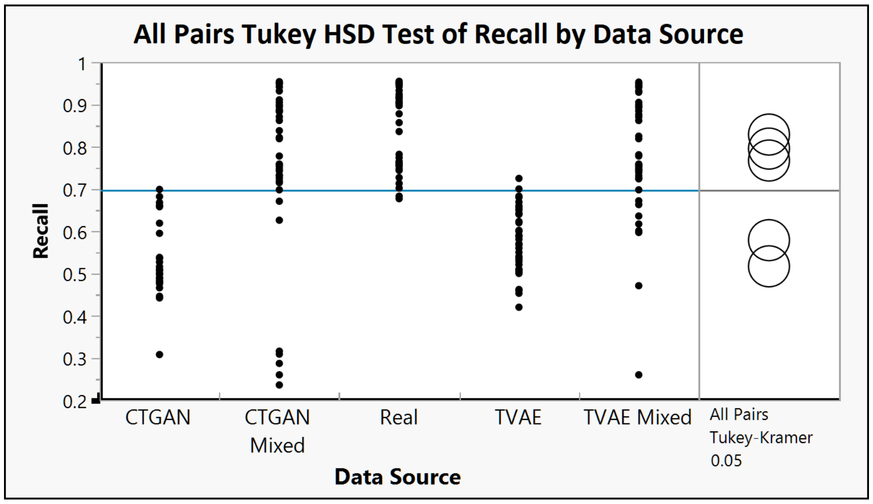


Figure 11. Plot of recall of the real, synthetic, and 50/50 synthetic and real data on the left, and connected groups with Tukey's HSD test of equivalent means on right.

0.730. When these two treatments are considered categorical, Tukey's HSD test fails to conclude the mean recall is different, with $\alpha = 0.05$. The low influence of training set size may be because the ML classifiers have a finite learning capacity.

Overall, classification recall is greatest with training sets containing no synthetic data and the recall decreases slightly as the percent synthetic increases. Figures 12-13 show an inflection point around 85% synthetic, at which point the recall decreases more rapidly. Following the standard practice of right sizing data sets in principal component analysis, this inflection point indicates that 15% real examples should be retained in training sets for machine learning based intrusion detection systems. An effects test with $\alpha = 0.05$ also determines that the machine learning classifier is an important factor for estimating recall. For every set in this experiment (25,000 CTGAN, Table 25; 100,000 CTGAN, Table 26; 25,000 TVAE, Table 27; 100,000 TVAE, Table 28), RF performed the best, and SVM performed the worst. The best observed recall in this experiment was 0.876 and the 20 best performing models were all RF.

The recall observed in this experiment for RF trained with 25,000 real examples is

0.861 (observed in both Table 25 and Table 27). This is inferior to the recall of 0.955 observed in Section 3.3 (see Table 21) where the RF model was also trained on 25,000 real examples. The lower recall in this experiment may be due to the increased size of the test set or random events during RF training.

The performance of RF had only decremented slightly upon offsetting 85% of the data with synthetic data. The recall for the RF models trained on 85% synthetic data were 0.820, 0.852, 0.821, and 0.853 with the 25,000 CTGAN, 100,000 CTGAN, 25,000 TVAE and 100,000 TVAE training sets, respectively.

Across the four sets, the mean recall for RF, SVM, MLP, and LR is 0.839, 0.535, 0.789, and 0.729, respectively. The standard deviation of recall for RF, SVM, MLP, and LR is 0.069, 0.069, 0.076, and 0.138, respectively. Evidently, the spread of recall for trials using the SVM classifier is much greater than for other classifiers.

Figure 12 and Figure 13 show that despite an overall trend of decreasing recall with increasing synthetic data, the relationship between recall and percent synthetic data is more nuanced for the SVM classifier. The SVM classifier used in this study employs the radial basis function kernel, which struggles to converge with large, diverse training sets. SVM may also be the least compatible classifier with the feature engineering scheme, shown in Table 3. This may be because SVM cannot perceive a difference between numerically similar, but distinct, label encodings. It is unclear why SVM classification improves slightly around 50% synthetic and again at 100% synthetic data.

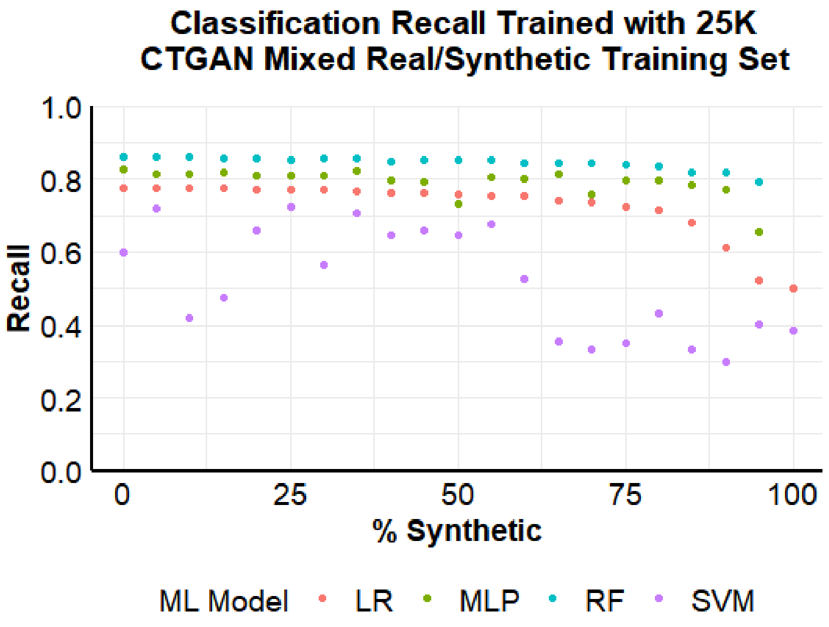
The results in this work show that some machine learning classifiers perform well when trained on sets of partially synthetic training data. This finding may help alleviate the lack of quality labeled cyber data for NIDS research. Deep learning classifiers have higher capacity than the conventional ML methods used in this study. Deep models excel with large, complex feature sets but they also demand a massive quantity of training data. The discovery that synthetic data can be used to offset real data lends itself to the possibility that deep methods can be applied with even greater classification performance.

3.5 Conclusion

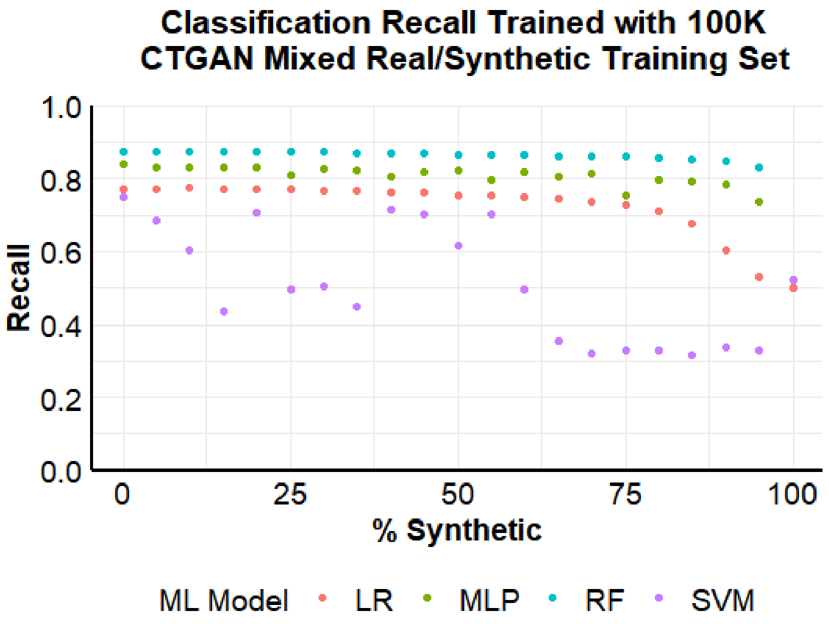
It is difficult to overstate the urgency of leveraging AI and ML technologies for cybersecurity to help protect military networks from attack. If vulnerabilities are not monitored, adversaries can suppress friendly systems from operating or potentially execute kinetic effects. NIDS, in conjunction with continuous monitoring, is an excellent way to detect, analyze, and react to threats.

As demonstrated in this work, the CTGAN and TVAE generative methods can each generate synthetic cyber data reasonably well. However, ML models trained with purely synthetic data resulted in underwhelming classification recall. These models yielded an unacceptable rate of false negatives. Classifiers trained with half synthetic data and half real data performed statistically equivalently to the classifiers trained with only real data. Further, it is shown that there is a linear relationship between percent synthetic data and classifier performance. NIDS classifiers performed well when trained with at least 25,000 examples and 15% being real. The engineered features used in this study did not use minimax scaling or one hot encoding for all variables, and this choice may have led to decremented classification performance with some algorithms.

Generative methods, including machine learning, evolutionary computation and Bayesian approaches, are an area of ongoing growth. The fast rate of advancement in this field is motivated by the broad applications of synthetic data. Synthetic cyber data, in particular, promises to address three well-documented issues with NIDS data in advanced cybersecurity research: lack of relevant records, expandability as

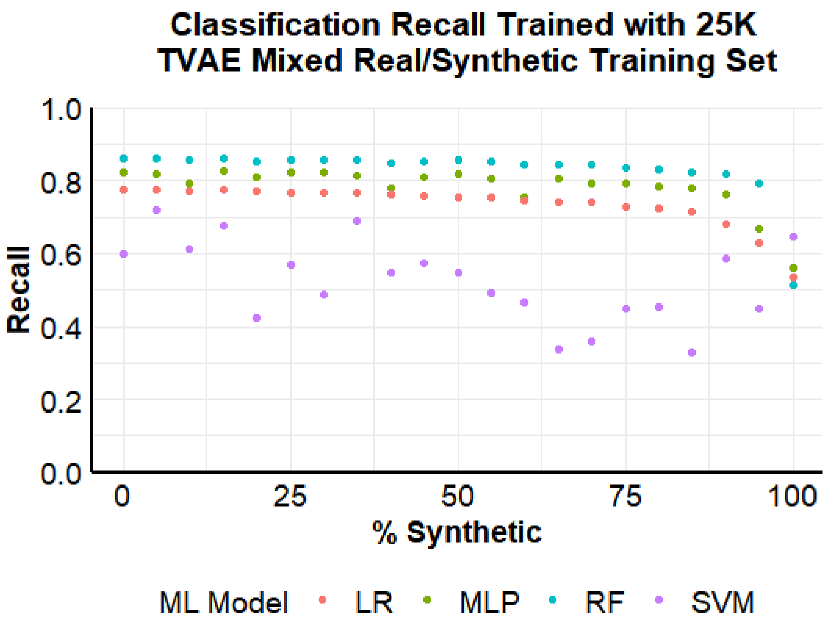


(a) ML models trained with 25,000 examples of real and CTGAN synthetic data

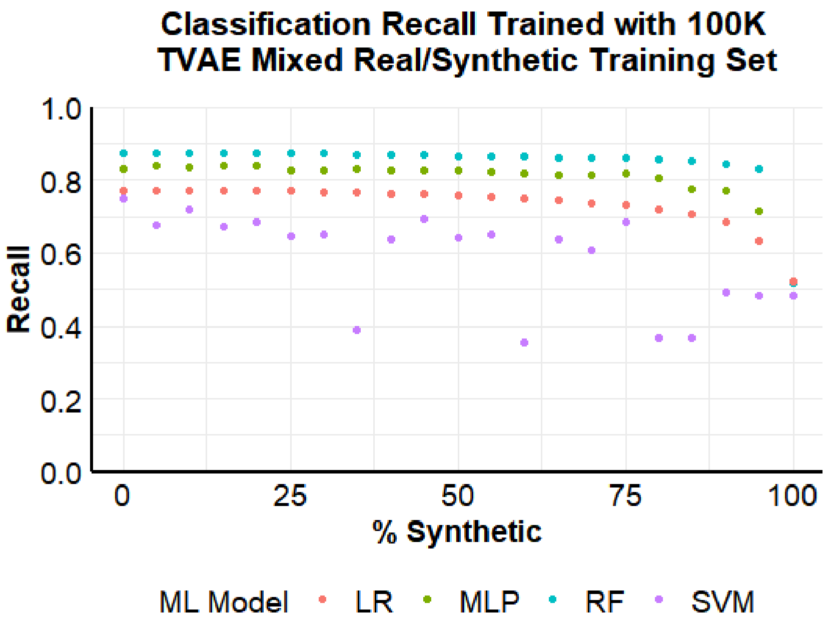


(b) ML models trained with 100,000 examples of real and CTGAN synthetic data

Figure 12. Classification recall of models trained with a mixture of real and synthetic data.



(a) ML models trained with 25,000 examples of real and CTGAN synthetic data



(b) ML models trained with 100,000 examples of real and CTGAN synthetic data

Figure 13. Classification recall of models where percent TVAE synthetic data is varied.

new information arises, and statistically rigorous testing and evaluation. Despite the efficacy of using synthetic data to train ML-based NIDS, as demonstrated in this research, there are some drawbacks. [109] critique that the joint distribution of synthetic cyber data produced by CTGAN and TVAE lacks likeness to the real data from the multivariate perspective. Therefore, non-conventional metrics such as Energy Statistics, proposed by [133], may provide insight unto the likeness of complex synthetic data sets to the baseline distribution. There is also a theoretical limitation on the utility of high quality synthetic data. Synthetic data from generative models does not introduce additional predictive information the system, but does improve statistical learning with the information already contained in real data. In the absence of a complex physical model or simulation, this limitation will likely not be breached.

We look forward to exploring improvements to generative methods as it may be applied to cyber data generation and evaluation, particularly for adversarial sample generation. These advancements will enable research on vulnerabilities and failure modes in AI-based NIDS. Future research should conduct cyber subject-matter-expert analysis of ML classifier false positives to determine if the classifiers can actually discover attacks that were mislabeled in the test set. These studies may expand beyond real cyber data onto high fidelity simulated data from cyber test beds such as LARIAT [134] and CyberVAN [135]. Future work will build upon the previous work of [11, 108, 136, 137, 138] to investigate adversarial sample generation in the constrained domain, as well as the use of meta-learning for improving the robustness of AI-based NIDS. Further extensions will explore evolutionary machine learning by developing cyber data generative methods through the open-world recognition context to generate not-yet-seen, but realistic and functional cyber data that is dynamic to potential future adversarial behavior.

IV. Generate Adversarial Examples

4.1 Introduction

Digital communications are integrated into all aspects of modern society to include business, home life, and military operations. Cyberspace is the system of networks and protocols that pass data across physical and virtual places. The rapid growth of digital communication through cyberspace has created new opportunities for commerce, social experiences, and education. Cyberspace has also proven a catalyst for the transformation society from industry 3.0, characterized by decentralized computing, to industry 4.0 where cyber-physical systems connect, humans, sensors actuators, and machines in order to massively automate decision making and response times. Ultimately, this trend towards automation will greatly reduce the requirement for a human in the loop for tasks.

This Internet of Things is expected to drive massive economic growth and productivity in the 21st century. Analogously, the Internet of Battlefield Things is drastically transforming military operations to remove humans from dangerous roles and to shorten the time between stimulus, detection, decision, and kinetic response. These architectures include human-in-the-loop, human-on-the-loop, and human-out-of-the-loop. Whether in the civil or military setting, the rate of advancement in cyber communications has outpaced any form of ensured cybersecurity. Despite the growing body of knowledge in cybersecurity, we often discover cyber vulnerabilities after systems are deployed. Attackers leverage their knowledge on cyber vulnerabilities to compromise the cyber triad: confidentiality, integrity, and availability of systems [92]. Security policies often account for a balance of system cost, utility, and security. Proper implementation of a good policy will maximize prevention of attack success and maximize attack detection[139]. This paper is concerned with network intrusion detection systems (NIDS) in the presence of adversarial attacks.

Adversarial machine learning has taken on a spotlight in artificial intelligence (AI) research as it highlights serious security vulnerabilities for AI models. Early work by [36] discovered that deep neural networks, which classify normal images with great accuracy, are easily fooled when images are strategically perturbed. Convolutional neural networks are the most prevalent model architecture in adversarial machine learning research. Numerous algorithms have been demonstrated to generate imperceptible perturbations in images that cause a target model to misclassify the image. Each of these algorithms exploits the following characteristic of image data: every pixel across all channels of image data can take on any value. There are no disallowed combinations of pixel values that result in invalid images. Perturbations can be made incrementally across a continuous domain. The only constraint relevant to these algorithms is the magnitude of permissible perturbation. This constraint is necessary to prevent humans, or anomaly detectors, from recognizing the visual artifacts of perturbations.

To the contrary, data in the cyber domain is strictly structured according to the internet protocol (IP) suite [140]. Packets are constructed with real data and utilize specific encoded commands to produce specific functional results. Small naive changes to the payload or meta-data of a packet could drastically affect the impact of the packet and in many cases could yield a corrupt or null packet. Even replacing a valid command in a packet with a different valid command could yield a sequence of code that fails to run. Therefore, any attempt at generating adversarial examples in the cyber domain must account for the rigid structure of IP packets and domain specific dependencies between coded elements. To date, research on adversarial machine learning in the cyber domain bypasses this issue of domain constraints entirely. Most previous efforts perturb packet flow features and not the actual packet payload. We assert that this approach is not as useful as an approach that perturbs packet payloads because a functional payload is necessary to implement the end-to-end cyber attack.

The NIDS of prior research learns from tangible features, or transformations of tangible features, so it is possible to draw conclusions about the importance of certain aspects of packet flow [11, 2, 138]. In practice, the packets used to implement these attacks can be modified to have any value for certain header fields such as IP address, port, and can be modified for other traits like payload length. For this reason, [45] demonstrates malicious packet detection on raw payloads, stripped of their spoofable headers. Deep learning methods such as convolutional neural networks inherently learn patterns in malicious packet payloads and can detect more rapidly than an experienced human analyst. It is not yet shown, however, if the machine learning models are leveraging the same tangible patterns as human analysts use. The CIC-IDS2017 (aka CICIDS) cyber data used in this study is publicly available and contains a variety of attack types in various protocols [141]. Our work implemented attacks using a variety of tools such as Slowloris, Metasploit, Hulk, and Patator. Based on recent findings of [45], it is our belief that raw packet payload learning is superior for feature learning for attack classification, and this holds true across a variety of attack types and protocols.

Our research is the first known solution to the problem of constrained optimization applied to raw packets to generate adversarial examples for NIDS [35]. Previous studies demonstrate adversarial attacks on features of cyber data and often ignore domain constraints. Perturbing features is not sufficient to engage in an end-to-end attack. By perturbing actual packets and respecting domain constraints, we expose a vulnerability of NIDS against an end-to-end adversarial attack. This manuscript reframes the problem and provides a mathematical formulation for optimally perturbing raw payload packets without affecting the packet function. A biologically inspired meta-heuristic, similar to genetic algorithm, is provided to solve the formulated constrained optimization problem. This work utilizes a designed experiment to find optimal combinations of hyperparameters for the meta-heuristic to generate adversarial examples. Then, additional experiments are conducted to compare the fool rate of adversarial examples when transferred to three machine learning classifiers. Just as the discovery of adversarial examples in the image domain has led to adversarial training as a robust defense, the vulnerabilities we uncover in this research open the door to a new generation of robust NIDS for the cyber domain.

The remainder of this work is structured as follows. Section 4.2 provides a review of relevant research. Section 4.3 defines the problem of constrained optimization for adversarial example generation, provides the use case for NIDS, and gives the formulation and experiment methodology. Results are presented and analyzed in Section 4.4, and conclusions are provided in Section 4.5.

4.2 Literature Review

Cybersecurity.

Cyberspace, and the tools used to build cyberspace, were initially intended to connect and empower people. However, the difference between a tool and a weapon is often based on how it is used. Over time, malicious actors have learned to use capabilities of cyberspace to steal confidential data, degrade the integrity of data and make systems unavailable [92, 142, 139]. The threat of cyber attacks permeates beyond the hardware and software of the cyber systems. If data becomes unavailable or incorrect, humans may succumb to rash, uninformed decisions and autonomous systems may become unstable. This leads to the possibility that cyber methods are used for espionage, sabotage, and cyber-kinetic attacks by state and non-state actors [86, 142]. The full impact of cyber attacks are still being researched, and it is apparent that security for existing infrastructures will not be sufficient for future environments [7].

As the military increased its reliance on cyber systems for managing and communicating information, for example, it found that their platforms had security deficiencies that could be exploited by outside attackers and insider threats. Cybersecurity was now seen as a crucial component of national security. Recommendations, such as introduction of tiered credential systems, aimed to deter attackers by increasing the cost and difficulty of the attempt, and to minimize the impact if successful [19]. A subsequent study expanded on the scope of known vulnerabilities and emphasized the need to log users, events, and traffic to learn about proper and illegal network usage [20].

Intrusion Detection System.

Intrusion Detection Systems (IDS) are a crucial component of cybersecurity because they alert and react to breaches in cyber policy automatically [30]. IDS may be embedded in host machines (HIDS), or more commonly integrated into networks (NIDS) with dedicated hardware and software applications [28]. [95] provides one of the earliest contributions to a NIDS. Denning's solution specified how to measure and log statistics on network events, resources, and traffic. A variety of expert systems and statistical methods were proposed to detect anomalies in the logs, indicative of a cyber attack. Denning also recognized that well informed attackers could adapt their attack to evade a NIDS either by slowly adapting their behavior until malicious behavior no longer appeared anomalous, or by exploiting a known vulnerability in the network security policy.

These systems may be designed to collect data and make detection decisions at host systems or centrally within a network [22]. Although host level systems offer the computer incident response team (CIRT) a high level of detail on specific machines, this strategy does consume compute resources on the host and it does not provide the CIRT with a holistic view of network behavior [28]. Host data is typically logged by the operating system, user applications [22] or security applications [28]. The preferred method of network data collection is through dedicated tap devices located strategically throughout a network. The main limitation of this method is that wireless communication of networked devices could at times communicate directly, and bypass the wired network. See [28] for guidance on tap placement in specific networks.

NIDS vary widely in their form. The detector is the most predominant area of research. The two approaches for detecting malicious traffic are signature based and anomaly based. Signature based detectors use previous knowledge to identify known threats. The anomaly detection strategy detects packets that are out of distribution from known packets. Signature based methods are more accurate and informative in general; however, only anomaly detection provides protection from unseen zero-day attacks.

Detector.

The detector is the component of the IDS that classifies network traffic as normal or as an attack. Modern IDS use either signature recognition or anomaly detection or both methods to identify potential attacks [92, 143, 144]. Signature based detectors may utilize methods such as expert systems [95] or machine learning [2, 109, 39]. Anomaly detection may be statistical, knowledge based, or machine learning based [144, 30]. A comprehensive survey on anomaly detection is given by [105] and provide a reference of anomaly detection approaches. Although anomaly based detectors are well suited to detect unseen "zero day" attacks, there is typically a higher risk of false positive alerts overall [143]. Machine learning, and deep learning in particular, provides high performance models for systems that are either too complex or too dynamic to be modelled with traditional methods [120]. 97.25% of IDS manuscripts reviewed by [30] incorporated machine learning techniques, and artificial neural networks were by far the most popular. [143] and [145] offer a review of machine learning and data mining techniques used for intrusion detection.

Cyber Data.

NIDS can be trained and deployed on packet data or aggregated network flow data. Most commonly, network flow data is used because the aggregated features provide context to the model [30]. Network flow datasets typical represents a bidirectional connection between nodes as defined by source and destination IP address, source and destination port number, and protocol ID. Raw flow level features may include quantity and rate of data transfer, length of connection, and protocol [45]. While denial of service (DoS) and distributed DoS (DDoS) cyber attacks are detectable through patterns in these features, many other attacks such as trojans, ransomware, and worms only express themselves through the content of the packet payloads. Further, any IDS that utilizes features of packet information or network flow are subordinate to the correctness of such features [45]. Various evasion techniques exploit this assumption by altering packet or network flow features [11, 138]. In fact, tools such as Scapy allow packet headers and payloads to be altered to any feasible value [146]. Therefore, the features used by NIDS, whether derived from individual packets or network flow data, could be completely fraudulent. Recent work contends that there there is a severe limitation on publicly available NIDS data for research [30, 109, 45]. Although it is possible to log traffic in networks, it is difficult to accurately label the traffic as normal or malicious and package the data in a standard form. [30] and [141] provide analysis on a number of commonly cited and publicly available cyber data sets.

Recent studies by [46] investigated transfer learning among several cyber data sets and machine learning architectures. The work found that models pre-trained with the CICIDS data set, published by [141], were accurate and also generalize well for use with other data sets. The CICIDS data set compiles raw packet capture (pcap) data collected on a real network over the course of a week. Only benign traffic was generated for the first day of collection; however on all other days, seven attack types were conducted using publicly available cyber tools. These attacks were implemented according to a pre-planned schedule so the resulting packet data could be accurately labelled [141]. We are concerned with the attacks that are most prevalent in the pcap data. Slowloris DoS attacks open many connections with a target server but never complete the transaction, draining the server of resources. Infiltration attacks, deployed via Metasploit, bypass security by embedding malware in downloaded files such as pdf. The infiltration attack then uses this access as a backdoor to conduct an IP sweep, portscan, and other exploitations using Nmap. Another DoS attack is conducted using the Hulk tool. Patador is used to conduct a brute force network attack on SSH credentials to gain illegal access [141]. [46] details a technique of labelling and preprocessing the resulting pcap data for raw payload learning.

Preprocessing Data.

There is also the matter of data preprocessing and feature engineering that plays a major role in the performance of machine learning classifiers. The particulars of effective pre-processing tend to be domain specific and algorithm specific. Imputation, aggregation, augmentation, vectorization, normalization, and kernel methods are other examples of preprocessing that, when performed properly, improve the performance of machine learning models [120]. Dimensionality reduction techniques such as principle component analysis may improve the tractability of running certain algorithms on large datasets, but does so at the cost of losing some predictive information.

Raw features in the cyber domain are logged from time series data, packet headers, packet payloads, or statistical metrics of network flow [147]. Literature discusses three actions for feature learning with NIDS: construction, extraction, and selection. Feature construction, or feature engineering, is the transformation of raw data into new, more expressive features for input to machine learning models [148, 30]. Feature engineering is extremely domain specific and relies on the judgement and experience of the analyst performing the study [120]. Feature engineering should use well understood relationships to present data in a form more amenable to learning. However, deep learning models, particularly convolutional neural networks, have the capacity to learn these relationships as well as more abstract relationships that are too complicated for an analyst to encode. It is still beneficial to perform feature engineering with deep models. It may reduce the amount of data and computational resources required to yield a satisfactory model; however, deep models are robust enough to adapt when that is not possible.

[149] studies the effectiveness of data representation techniques for nominal variables in the KDD Cup 1999 cyber data set. This study does not, however, use expert knowledge, but rather rules of thumb for representation. Other works claim that the literature overemphasizes feature *selection* criteria but does not properly pursue expert feature engineering in the cyber domain [39]. When feature engineering is used, it is based on legacy practices and anecdotal knowledge. In the study, [39] uses domain specific knowledge to generate feature sets on 10 features from network flow data collected from the U.S. Department of Defense Research and Engineering Network [39]. [45] expresses concern that the choice of raw feature extractor software, such as Zeek and Security Onion, provides an extra layer of obscurity to the IDS detector since various extractors use proprietary logging and aggregation techniques.

Feature extraction is an algorithmic search for optimal feature transformations with respect to a performance metric. Recent research involves feature extraction with deep learning methods, though principal component analysis also may be considered feature extraction [148]. Notably, however, feature extraction through principal component analysis optimizes information retained, not a metric on classification performance [119].

Feature selection is different than dimensionality reduction. Feature selection removes the least informative features to promote efficient learning. Many machine learning algorithms suffer from the curse of dimensional, and feature selection increases tractability [119]. [100] notes that the NSL-KDD data sets of [8] contains 43 features, many of which are noisy or otherwise non-informative. They therefore compare statistical metrics for feature selection. Unlike dimensionality reduction techniques, the strategy of [100] is to select the most useful raw variables, not to aggregate them. Models using natural variables may be more transparent to an analyst than models using transformed variables. [150] proposes that the laborious task of cyber feature selection should be algorithmically learned to remove human bias and improve learning on the base task. They generated 16 expert and arbitrary features from raw HTTP payloads. Their experiment shows that even abstract engineered features may be useful for base learning. The automatic selection among features yields good results while reducing analyst labor.

Despite the benefits of automating feature selection, [45] claims that proprietary feature logging software such as Zeek and Security Onion add an extra layer of obfuscation to feature engineering and selection. In practice, the data format and availability of features varies widely across factions of the cyber domain such as wifi, NIDS, HIDS, virtual private networks, and IoT. [151] reports that cyber data sets typically aggregate the information within raw packets into features of network flows. The resulting features emphasize packet header information and not the content and effect of the payload. [150] found that models trained on packet header data can detect only a subset of attack types. Many modern attacks are best detected from application protocol commands found in the packet payload.

[30] provides analysis on 85 manuscripts and 30 prominent data sets for NIDS research. According to the study, 50.5% of IDS manuscripts reviewed utilized the KDD-99 data set [30], which was recorded over 2 decades ago and has well documented flaws [8]. Another 17.2% of studies employed the NSL-KDD dataset. Although NSL-KDD corrects many statistical flaws of KDD-99, it is still derived from the outmoded KDD-99 data logs. [30] cautions an overall lack of compatibility among cyber data sets, and in particular an inability for modular sharing and expandable of data sets. The community should emphasize a common format and feature set.

Performance Metrics.

The most costly event for a NIDS is a malicious packet that is not detected. The malicious packet will enter the network and inflict harm undetected. In this case, the NIDS falsely labels the malicious packet as belonging to the negative class (FN). The proper classification of a malicious packet is a true positive prediction (TP). The erroneous classification as malicious is a false positive label (FP). The correct classification as negative is a true negative classification (TN) [109]. The accuracy of a NIDS is shown in Equation 42, as quantity of true positive plus true negative divided by the total classifications [131].

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{42}$$

Recall, also known as the true positive rate, is shown in Equation 43 [131].

$$Recall = \frac{TP}{TP + FN} \tag{43}$$

These two metrics provide good representation of detector performance on both malicious and non-malicious packets. There is additionally a need to quantify model robustness under adversarial attack. One approach is to measure the minimum perturbation η of correctly classified example $\mathbf{x}$ such that $f(x + \eta) \neq f(\mathbf{x})$ despite each each input's true classification being the same. η is the minimal adversarial perturbation if $\nexists \xi$ S.T. $\|\xi\| < \|\eta\|$ for any adversarial perturbation ξ [152]. The robustness of a model is therefore characterized by Equation 44, and detailed in Equation 45. Whereas $\triangle_{adv}(\mathbf{x}; f)$ is the minimum distance required to change the model prediction from correct to incorrect on one sample, $\rho_{adv}(f)$ is the expectation of the minimum distance cross the entire feature space. μ represents the probability metric in $\mathbb{R}^d$ [152].

$$\rho_{adv}(f) = \mathrm{E}_{x \sim \mu}[\triangle_{adv}(\mathbf{x}; f)] \tag{44}$$

and

$$\triangle_{adv}(\mathbf{x}; f) = \min_{\eta \in \mathbb{R}^d} \{\|\eta\| \mid f(x + \eta) \neq f(x)\} \tag{45}$$

Further analysis of metrics on model robustness is provided by [138].

Machine learning algorithms learn model parameters by minimizing a loss function which reflects how well the model performs its task. The most popular loss function for machine learning classification is *cross-entropy*. Cross-entropy has emerged over time as a derivative of early metrics in information theory. *Self-information*, expresses how much information is learned by reporting the specific outcome x of a probabilistic event following distribution P. Self information is defined in Equation 46.

$$I(x) = -\log P(x) \tag{46}$$

Entropy, as shown in Equation 47, is the expectation of information learned from a new observation. It is a measure of uncertainty across the entirety of distribution P[153, 154]. Intuitively, entropy estimates the lower bound on length of code to inform the next observation. The base of the logarithm defines the units of information and in machine learning it is conventional to use natural logarithm and information units called *nats*. Alternatively, a base-2 system yields units of bits [102].

$$H(x) = \mathbb{E}_{x\sim P}\big[I(x)\big] = -\mathbb{E}_{x\sim P}\big[\log P(x)\big]. \tag{47}$$

Kullback-Leibler divergence, Equation 48, [153] extends the concept of information to discriminate between a hypothesized distribution P and observed distribution Q [102].

$$D_{\mathrm{KL}}(P||Q) = \mathbb{E}_{x\sim P}\big[\log P(x) - \log Q(x)\big]. \tag{48}$$

Cross-entropy, shown in Equation 50, is used to identify model parameters in classification problems by minimizing the difference of hypothesised distribution P and known distribution Q. Although optimizing cross-entropy is equivalent to optimizing Kullback-Leibler divergence, cross-entropy is preferred because P is not evaluated in the formula [102]. Cross-entropy serves as the loss function for deep models in this paper.

$$\begin{aligned} H(P, Q) &= H(P) + D_{\mathrm{KL}}(Q||P) && (49)\\ &= -\mathbb{E}_{x\sim P}\log Q(x) && (50) \end{aligned}$$

Adversarial Machine Learning.

While machine learning models should be robust enough to train and test well in the presence of outliers, this objective becomes much more difficult in the cyber domain. The cyber domain is inherently adversarial [11, 155, 138]. Cybersecurity systems are designed specifically to detect unauthorized activity. A motivated attacker, however, will spend great resources to fool the model. An attempt to fool a security system such as an IDS or spam detector at test time is known as an evasion attack [156].

A naive evasion attack is not necessarily an adversarial attack. For example, an exploratory attack may naively search a feasible variables space for *blind spots* where a classifier does not perform correctly. These attacks partially learn the topography of the decision space but are limited by the number of queries performed and do not learn anything about unqueried regions. Efforts seeking to learn the geometry of convex sets via random sampling are too difficult to be practical for evasion attacks [157].

Even as the accuracy of machine learning models improves with normal training and test data, evasion attacks can actually succeed consistently [11]. Traditional metrics of classifier performance only convey a partial picture of classifier success. Classifiers must also be robust for out of distribution examples at test time [11, 155]. The statistical distribution of cyber data is non-stationary, meaning the observed behavior evolves over time and classifiers must adapt. Part of this changing distribution is attributed to increasingly clever evasion techniques invented by attackers.

Methods that utilize a known, or approximately known, loss function of the classifier are considered adversarial machine learning. Although extraction, poisoning, and inference are other forms of adversarial machine learning [11, 138], this paper is a study at evasion attacks of the classifiers at test time. Extraction attacks seek to learn the geometry of parameterization of surrogate models by querying target models [158, 159]. Poisoning attacks corrupt classifier performance at test time by purposely altering training data. The poisoning approach requires an attacker to have access to the database of training examples, an assumption that is often not realistic [11].

In a simple unconstrained case, the adversary modifies data, restricted within some closed ball, to evade the classifier. Early experiments demonstrated adversarial attacks to evade differential classifiers at test time using gradient descent attacks [155]. Gradient attacks, however, are not suitable for NIDS data which is highly constrained with discrete variables and other feasibility requirements. Later results proved it is possible to evade non-differentiable classifiers such as decision trees, K-nearest neighbors on cyber data. Particle swarm optimization demonstrated evasion rates of 99.99% on the UNSW-NB15 feature set [11]. In practice, true evasion rates will vary based on resources, knowledge of the classifier, and feasibility constraints.

The literature provides some discussion as to why adversarial regions exist in various machine learning models. Initial theories relied on the belief that non-linearity and non-convexity of high dimensional models, and in particular artificial neural networks, led to the existence of mis-classified regions. Experimentally, [160] finds adversarial examples also exist in other classes of machine learning models such as logistic regression and neural networks with near linear tendencies. In fact, in the unconstrained image domain, adversarial examples can be efficiently solved even if the model has modest capacity and is properly fit [160]. This finding then begs the question if machine learning models are learning the correct underlying patterns for classification. This is especially concerning for points that are close in Euclidean distance to high probability points in training data, but are themselves not represented in the training data [160]. Another theory explaining the existence of adversarial examples states that neural networks are highly non-linear in some regions but linear in others. Therefore, the model may behave unnexpectedly for certain test points [152]. The theory of evolutionary stalling notes that an ideal model should behave smoothly around any test point, however, during gradient descent training, the importance of training examples is greatly reduced once a particular point is correctly classified [161]. The model is incentivized to evolve in ways that do not change correct predictions. Therefore, as true positive predictions are achieved, effective learning stalls in the vicinity of the correctly classified training examples leaving decision boundaries precariously close to high likelihood regions of the training data.

When adversarial examples are identified, these points can be transferred to different machine learning models and fool these models at inference time with a high rate of success as well which implies a variety of algorithms will yield similar models. [160] provides experimental evidence that the transferability of adversarial examples is largely due to the near linearity of the models in the vicinity of training points. Adversarial training, which augments the training set with correctly labelled adversarial examples, is shown to improve resistance to adversarial attacks [108, 35, 160, 138]. It is also theorized that generative methods can be used to create a *rubbish* class of training points to improve training of models for real classes [160]. Creating ensembles of models is shown to increase robustness of models against adversarial attack, however, these ensembles have been shown slightly less accurate against non-adversarial test examples. Not all regularization techniques are effective defense against adversarial attack [160]; however, [162] penalizes the squared norm of the Jacobian for model parameters. This penalty promotes smoothness of the loss function and empirically contributes to model robustness. It is therefore hypothesized that it is the nature of the loss function and not the model architecture that drives the robustness of deep neural networks.

4.3 Methodology

Problem Overview.

Adversarial examples are data examples that have been altered such that a machine learning model changes its prediction from the correct class to an incorrect class [48]. In the image domain, color intensity of pixels can be perturbed without restriction, and there are many techniques for adversarial example generation, including projected gradient descent, Carlini & Wagner, and JSMA [35]. In other domains, however, it is generally not possible to make small perturbations to data to motivate misclassification. Naive perturbations to code will likely result in an example that fails to perform the original function. A new formulation is necessary to generate adversarial examples in constrained domains. This problem type can be referenced as the *The Constrained Adversarial Example Generation Problem*, which is formulated later.

Solutions to the constrained adversarial example generation problem are primarily used for evasion attacks against credit card fraud detectors, spam detection, NIDS, malware detection, and in constrained instances of image classification. Additional applications arise as attackers seek to fool classifiers in any domain where specific, known constraints bind the solution space. Alternatively, constrained perturbations could be used for a poisoning attack to degrade a machine learning model at training time. We motivate this problem class by highlighting the case of evading NIDS classifiers. NIDS classifiers are typically trained with features of cyber packets derived from network security monitor such as Zeek [39]. Previous work by [11] and [138] demonstrated that cyber feature data, such as NSL-KDD, can be perturbed to generate adversarial examples [35]. Unfortunately for the attacker, the feature data does not contain enough information to reconstruct a complete cyber packet used during a live attack. [45] demonstrates that 1-D convolutional neural networks (CNN) can be trained with 98.82 accuracy on the raw payload of cyber packets. This deep learning approach mitigates any necessity for feature engineering as the CNN model intrinsically learns features from labelled raw data. Just as the classifiers of [45] are trained directly from raw data, and not feature data, we seek to generate constrained adversarial examples from raw cyber data.

Generating adversarial examples from raw packets in the cyber domain is a constrained optimization problem. Computer code can be partitioned into functional units, where each unit performs a specific computational task. If the code used for a specific unit is replaced with a different, yet functionally equivalent code, the overall code remains functionally equivalent, satisfying the feasibility requirement of the problem. The solution is an entire set of code that fools the surrogate model, and the decision is to select exactly one alternative for each functional position. Our approach is fully dependent on an assumption that certain IP protocols such as HTTP are agnostic to capitalization of code. We leverage this assumption for perturbing capitalization of characters in payloads of various attack types. This choice of equivalent substitutions is a proof of concept for other choices. For example, an alternate corpus could be generated to establish functionally equivalent substitutions of variable names, commands, or entire lines of code. Unlike the approach in [43], there is no need to check feasibility at each step of the proposed meta-heuristic. All substitutions taken from the corpus are feasible. Further, the proposed method can be implemented with gradient and non-gradient surrogate models. Because the new method does not depend on gradient steps, it can be applied on systems with discrete and categorical variables.

Lemma 1. An unlimited number of functionally equivalent substitutions of partitions of code will produce a code that is functionally equivalent to the original.

The partitions of $\mathcal{A}$ can be repeatedly replaced with functionally equivalent partitions to yield a functionally equivalent code $\mathcal{A}'$.
Define code $\mathcal{A}$ and $\mathcal{B}$ as functionally equivalent iff for every input x, including a null input, $\mathcal{A}(x) = \mathcal{B}(x)$.
Given, set $\mathcal{A}$ and set $\mathcal{B}$ are partitioned into units $i...n$
$\forall i \in 1, ...n, \ \mathcal{A}_i = \mathcal{B}_i$.
Now, for any i replace $\mathcal{A}_i$ with $\mathcal{B}_i$ yielding $\mathcal{A}'_i$ which is functionally equivalent to $\mathcal{A}_i$.
WLOG, given $\mathcal{A}$, the partitions of $\mathcal{A}$ can be repeatedly replaced with functionally equivalent partitions to yield a functionally equivalent $\mathcal{A}'$.

The first challenge in solving this constrained adversarial example generation problem is to identify all known functionally equivalent units that yield the correct, equivalent function for each position. The set of equivalent feasible alternatives for each units i is denoted as the corpus $\mathcal{C}_i$ and the gene x_{ij} is the decision variable. Next, there is the challenge of identifying a loss function to optimize. Projected gradient descent [160] and its progeny algorithms use the training loss function of the classifier to generate perturbations. In particular, these algorithms use gradient of a training example with respect to constant model parameters. Each small step of this type results in an increase of the loss value [138].

In the constrained problem, we also seek to increase the loss function by changing the units in the example. However, the gradient information cannot be used for perturbations, as the small steps would yield an infeasible, nonfunctional solution. Therefore, substitutions must be made from the corpus until the loss ceases to increase. Feasible solutions require that exactly one alternative is selected per functional unit. The selection of each gene is a binary integer program. More specifically, it is a boolean satisfiability problem, where the goal is to find a solution that fools the surrogate classifier. The boolean satisfiability problem is an NP-hard problem and is very difficult to solve [163, 164]. In our case, we do have the benefit of knowing the model loss function but polynomial time discrete optimization techniques cannot be used because the loss function is non-convex. The problem is further complicated because a solution's success in fooling the NIDS cannot be verified until test time.

This research investigates the fool rate of near-optimal adversarial examples generated from raw packets of four attack types. The most frequent attack in the training data is DoS performed with Slowloris followed by infiltration attacks, DoS performed with Hulk, and finally the SSH attack conducted in Patador. First, the packets are split by attack type.

Figure 14 shows the primary steps of the methodology to create transferable adversarial examples. The first step involves labeling, pre-processing and splitting. The second step is to train a surrogate classifier as well as three NIDS classifiers. Next, a full factorial design of experiments is performed to identify the packet characteristics and meta-heuristic hyperparameters that maximize the cross-entropy loss of adversarial examples on the surrogate model. The optimal settings for each attack type are carried over to step four, where constrained adversarial examples are generated for each attack type. The fifth step is transferring and testing the adversarial examples on three NIDS and reporting the rate at which the examples fool the classifier.

Data Preprocessing.

Following the strategy of [45] and [46], the training and test data is derived from pcap data on a network under cyber attack. Our research utilizes the CICIDS dataset rather than the UNSW-NB15 dataset used by [45] because preliminary results by [46] indicate the models trained from CICIDS are more generalizable than models trained with UNSW-NB15. CICIDS packets from Monday through Friday are selected for this research in order to support diversity of both attack and benign packets in the

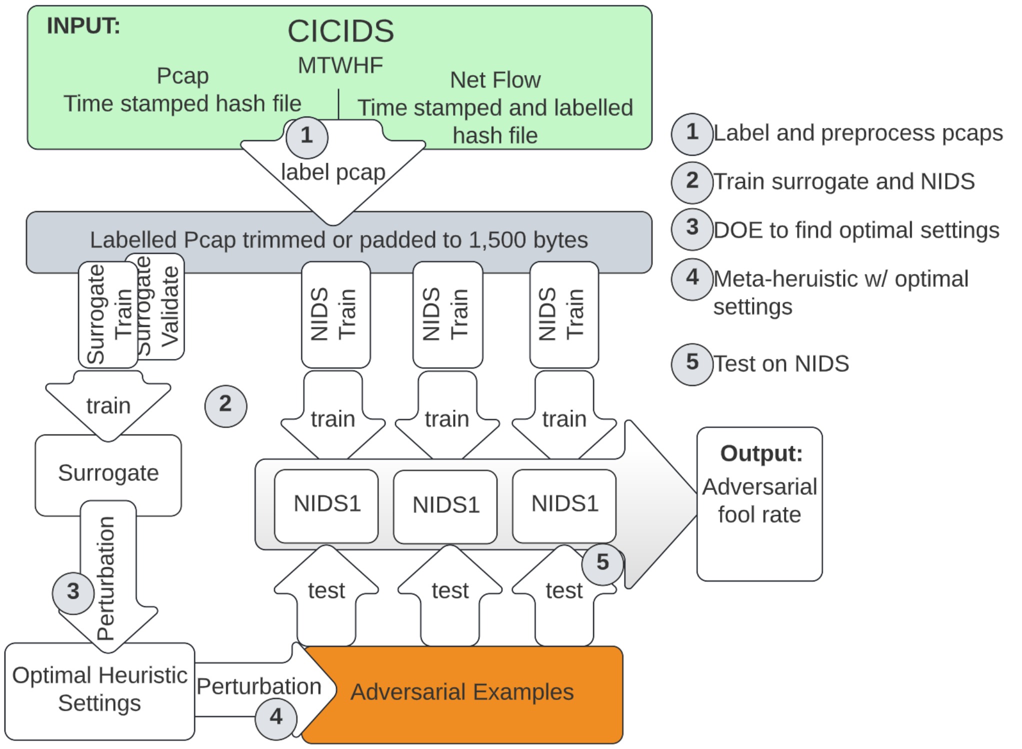


Figure 14. The methodology requires data preprocessing, training a surrogate model and three NIDS models, generating adversarial examples, and testing the adversarial examples on the NIDS.

training and test sets. Only UDP and TCP packets were retained for analysis because alternative communication protocols were either too scarce for supervised learning or otherwise do not contain meaningful information in the payloads [46]. Data processing for this experiment is greatly simplified by omitting feature engineering; however there are several important steps taken to decompose raw packet capture data to structures that can be input to machine learning models.

Although the pcap data was not labelled from the publishers [141], they provided enough meta-data to do so post hoc. We repeated the labelling process detailed by [46]. Timestamped and labeled hash files of network flow data were used in conjunction with timestamped pcap files to assign the proper label to each pcap file. The reduced database includes malicious and benign http, https, SSH, FTP and email protocol packets but https were dropped due to a lack of attack examples. The hexadecimal encoded payloads of these packets were converted from binary octets to integer representation [0, 255]. Next, the size of the payloads were standardized to 1,500 bytes by trimming large payloads or padding shorter payloads with zeros. The resulting payloads were minimax normalized [0, 1] for efficient learning. This produced 331,868 labelled pcap payloads. As depicted in Figure 15, 45% of these payloads were randomly sampled without replacement to train a surrogate model and 45% to train the NIDS models, respectively. 5% of the payloads were randomly set aside to validate the surrogate model, and the remaining 5% allocated to validate the NIDS models.

Train Models.

We consider a grey box adversarial threat scenario that assumes adversary knowledge of pcap preprocessing but no knowledge of NIDS model architecture and no input/output probing. For this reason, it is necessary to train a surrogate (aka proxy)

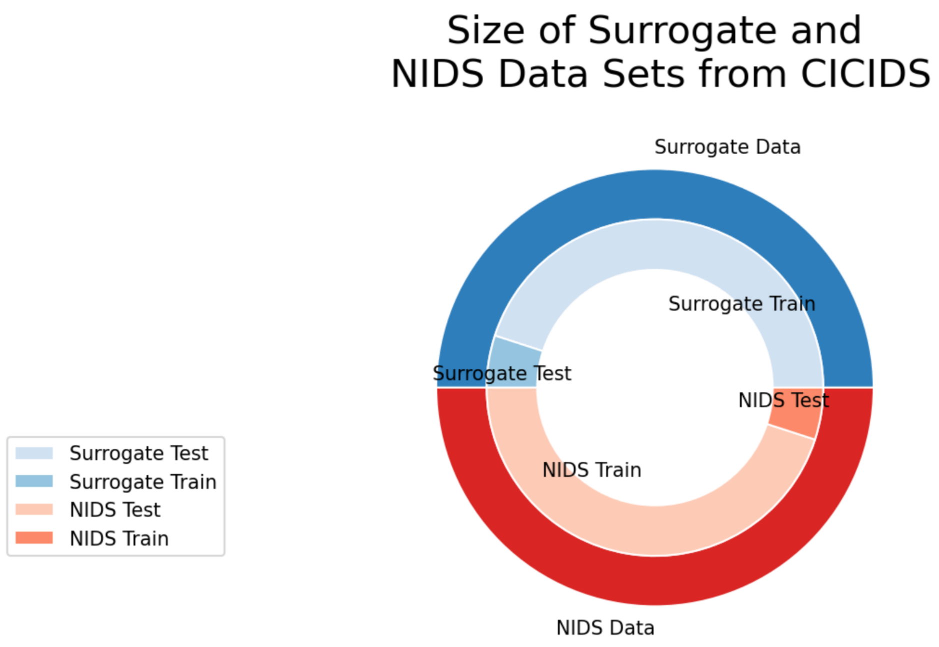


Figure 15. Distribution of CICIDS data used to train/test surrogate model and to train/test the NIDS models.

model for constrained adversarial example generation, while also training NIDS models to evaluate the fooling rate of the adversarial examples. Training these models is Step 2 of the methodology, shown in Figure 14. The three NIDS models include a CNN, a fully-connected neural network (FNN), and an Adaboost model.

CNN Surrogate and NIDS.

We utilize a 1-dimensional CNN that captures positional relationships between payload bytes. This is well suited for classifying raw packet data because of the verbal and programming languages contained within. The architecture of the CNN is shown in Figure 16, and the same architecture is used to produce the surrogate model and the first NIDS model. Adam is among the most effective optimizers for training deep learning models [165] and is flexible enough to use in various deep architectures. We use the Adam optimizer in all deep learning models trained, including the surrogate model used to generate adversarial examples and the NIDS models used to test the utility of the adversarial examples. The loss function used for all models, BinaryCrossEntropy, is shown in Equation 61.

FNN NIDS.

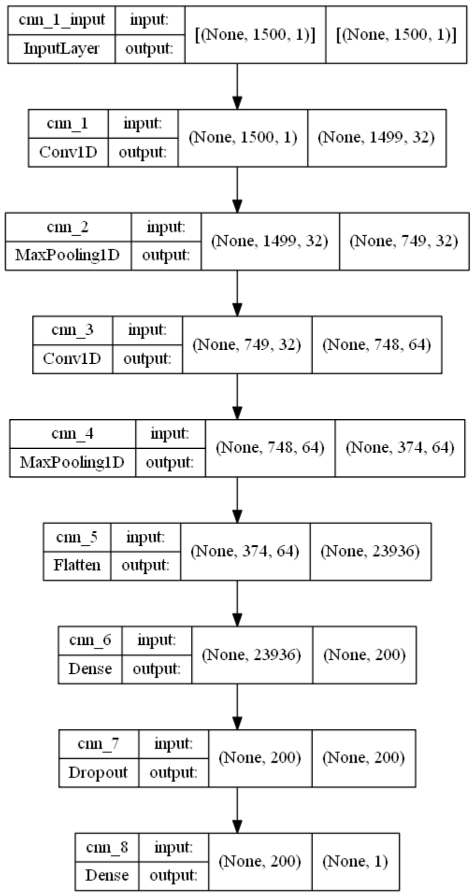
A FNN is presented for use as a NIDS model to test fool rate of the adversarial examples. Although simpler than the 1-dimensional CNN, the FNN is a deep architecture that can take on great capacity to model complex non-linear relationships. The architecture of the NIDS FNN is presented in Figure 17. Again, Adam is selected as the optimizer and binary cross-entropy is the loss function.

Adaboost NIDS.

An Adaboost classifier model was defined with 100 base estimators. The base estimators were designated as 1 node deep decision trees. The learning rate was 1 and the SAMME.R of [166] was employed for training.

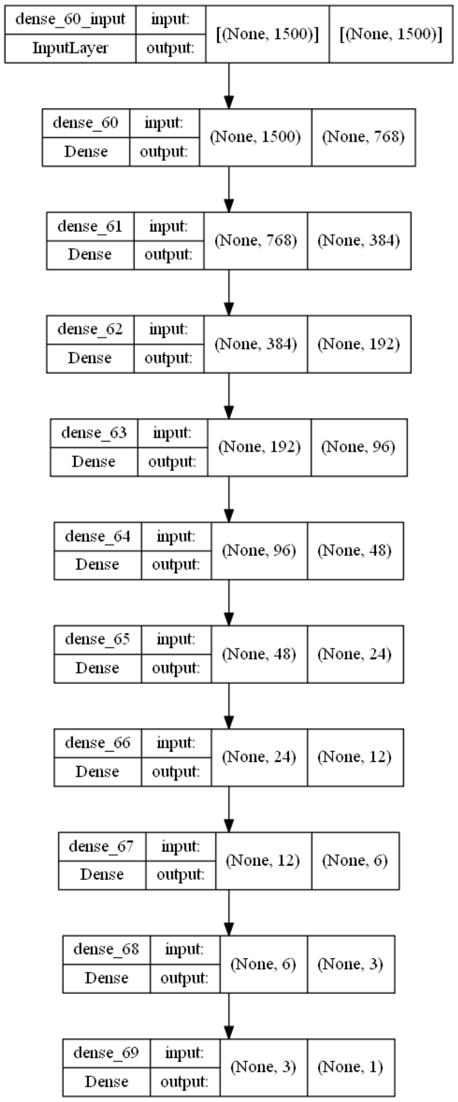
Meta-heuristic.

We develop a meta-heuristic as an engine to generate near-optimal constrained adversarial examples for raw packet NIDS models. The meta-heuristics goal is to perturb a malignant packet payload such that it remains a feasible packet payload which maximizes the classification loss with respect to a trained model. A perturbed packet with sufficiently large loss will fool the surrogate classifier with certainty, and has a strong chance of fooling other classifiers. Meta-heuristic solutions are constrained such that they must maintain the function of the original unperturbed packet. This constraint is fundamental for an end-to-end adversarial attack. The packet must not only bypass the IDS detector but its packet must perform its exact original purpose in the target application. We denote this property as being *functionally equivalent*.



cnn_1_input | input: | [(None, 1500, 1)] | [(None, 1500, 1)]
InputLayer | output:
cnn_1 | input: | (None, 1500, 1) | (None, 1499, 32)
Conv1D | output:
cnn_2 | input: | (None, 1499, 32) | (None, 749, 32)
MaxPooling1D | output:
cnn_3 | input: | (None, 749, 32) | (None, 748, 64)
Conv1D | output:
cnn_4 | input: | (None, 748, 64) | (None, 374, 64)
MaxPooling1D | output:
cnn_5 | input: | (None, 374, 64) | (None, 23936)
Flatten | output:
cnn_6 | input: | (None, 23936) | (None, 200)
Dense | output:
cnn_7 | input: | (None, 200) | (None, 200)
Dropout | output:
cnn_8 | input: | (None, 200) | (None, 1)
Dense | output:

Figure 16. The architecture of the convolutional neural network selected for the surrogate model and the first NIDS model.



dense_60_input | InputLayer | input: | output: | [(None, 1500)] | [(None, 1500)]
dense_60 | Dense | input: | output: | (None, 1500) | (None, 768)
dense_61 | Dense | input: | output: | (None, 768) | (None, 384)
dense_62 | Dense | input: | output: | (None, 384) | (None, 192)
dense_63 | Dense | input: | output: | (None, 192) | (None, 96)
dense_64 | Dense | input: | output: | (None, 96) | (None, 48)
dense_65 | Dense | input: | output: | (None, 48) | (None, 24)
dense_66 | Dense | input: | output: | (None, 24) | (None, 12)
dense_67 | Dense | input: | output: | (None, 12) | (None, 6)
dense_68 | Dense | input: | output: | (None, 6) | (None, 3)
dense_69 | Dense | input: | output: | (None, 3) | (None, 1)

Figure 17. The architecture of the fully-connected neural network used for the second NIDS model

Gradient methods have previously been used to perturb data and create adversarial examples in the unconstrained domain of computer vision. In the constrained cyber domain, finding feasible solutions that maximize loss is a non-convex assignment problem and cannot be solved with deterministic solvers. In the constrained domain, it may be possible to obtain the gradient of $\mathcal{L}$ for example X^k; however, the gradient of a deep model is highly non-convex and only reliable in very local regions. Therefore, a step in the direction of the gradient would in general not be feasible. On the other hand, a feasible step would likely not follow the local gradient. Fortunately, there are non-gradient search heuristics.

The proposed meta-heuristic is inspired by the genetic algorithm but is applied in a way that guarantees each step of the solution to feasible. A true genetic algorithm performs crossovers on known feasible solutions or arbitrary feasible solutions. Our approach, however, only implements functionally equivalent gene substitutions. Hence there is no need for an additional feasibility check.

Lemma 1 justifies that an unlimited number of crossovers can be made across feasible chromosomes and result in new, feasible, functionally equivalent chromosomes. We also note that this problem does not fit into the traditional class of the "assignment problem" as the assignment problem has a linear cost function written in the form $min \sum\limits_{i,j=1}^{n} c_{ij}x_{ij}$. Our problem has a nonlinear, non-convex cost function, taken from the trained CNN loss.

Equivalent substitutions (genes) are initially placed into a corpus organized by position (allele). A sample population of initial solutions (chromosomes) is created via random draws of genes from each allele (i.e. random combinations of functionally equivalent substitutions). This population is then used to seed the meta-heuristic. We find effective convergence is achieved by generating a large initial population and retaining only the best chromosomes in the initial solutions for the crossover phase. Each generation the 'genetic-algorithm' creates new 'child' chromosomes via crossover of selected 'parent' chromosomes. Specifically, a crossover point in the packet payload is drawn using a uniform distribution between 0 and the number of alleles in the solution. Notably, the function used to draw the crossover point does not consider 0-padding as variable alleles in the solution; most chromosomes contained less than 1,500 alleles and will always have a crossover position less than that number. The probability of selecting a chromosome as parent is defined by a stepwise linear probability distribution function defined in Equation 63 and visualized in Figure 18. This distribution assigns the greatest probability of selection to the most fit chromosome (number 20 in the figure), and the lowest probability to the least fit chromosome (number 1 in the figure).

$$p(\chi = x) = \frac{(n - x)}{\sum\limits_{x=\{1,2,\ldots n\}} x} \tag{51}$$

In addition to crossover driven evolution, child chromosomes are probabilistically subjected to random mutations. Unlike a standard genetic algorithm, mutation steps in this meta-heuristic are guaranteed to yield feasible solutions because each mutation is a substitution with a functionally equivalent gene. Mutations of child chromosomes promote exploration of the solution space and prevent premature convergence to local minima. In some cases, mutations yielded best known solutions. The meta-heuristic also has the advantage that it can be tuned for exploration or exploitation by adaptively updating the chance of mutation. It is further possible to tune the percent of genes mutated; in steps mutation does occur. The probability that mutation occurs on a child chromosome is shown in Equation 64. At baseline, the probability of mutation occurring is set at 5%; however it increases by 1% with each generation that fails to produce improvement. Mutations are guaranteed after 95 consecutive children fail to produce an improvement. This feature promotes exploration when

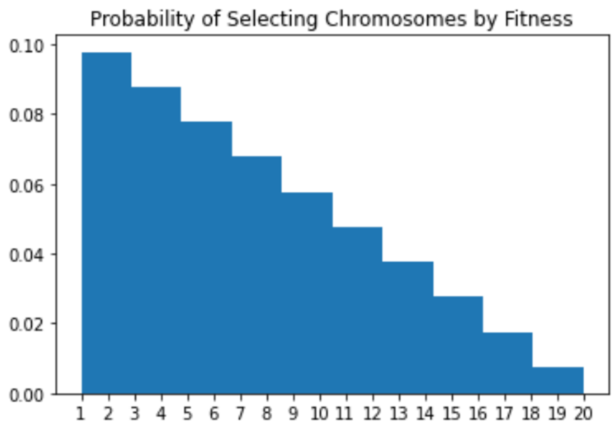


Figure 18. Chromosomes from the population are probabilistically chosen as parent chromosomes for crossover according to a stepwise linear distribution.

exploitation is not effective.

$$P(mutate) = 0.05 + (0.01)(generations\ since\ improvement) \tag{52}$$

What follows is the mathematical formulation of *The Constrained Adversarial Example Generation Problem.*

Objective Function.

The objective function $H(P,Q)$ is the cross-entropy of the model's hypothesized classification, distributed as P, and the true labels in the data, distributed as Q.

$$H(P,Q) = H(P) + D_{\mathrm{KL}}(Q||P) \tag{53}$$

$$= -\mathbb{E}_{x\sim P}\log Q(x) \tag{54}$$

Sets.

We define the following variables $\{i, j, k\}$ along with their corresponding sets as:

$i \equiv$ allele	$i \in \{1, 2, ...n\}$,
$j \equiv$ gene	$j \in \{1, 2, ...m\}$,
$k \equiv$ test example	$k \in \{1, 2, ...o\}$

Decision Variables.

$$x_{ij}^{(k)} = \begin{cases} = 1 \text{ if gene } j \text{ is selected for allele } i \text{ on example } k \\ = 0 \text{ otherwise} \end{cases} \tag{55}$$

Constraints.

$\sum_{j \in \mathcal{J}} x_{ij} = 1$	$\forall i \in \mathcal{I}$	One gene selected per allele
$x_{ij} \leq q_{ij}$	$\forall i, j$	Allowable genes are defined by 1's in a sparse matrix Q
$x_{ij} \in \{0, 1\}$		Assignments are binary

$\mathbb{Q}$	Sparse matrix of dimensionality $n \times m$. $q_{ij} = 1$ if allele i can be set to gene j
$\mathcal{L}(X\|\Theta)$	The loss function of a model with set parameters Θ is provided for any problem instance
$\mathbb{C}_i$	The set of feasible genes for each allele is referred to as a corpus. This information is used to generate Q

Data.

The corpus of feasible substitutions includes n lists, that is one list for each allele, or functional unit.

$$\mathcal{C} = \{C_1\ C_2 ... \ C_n\} = \begin{Bmatrix} x_{1\cdot 1} & x_{4\cdot 2} & \dots & x_{7\cdot n} \\ x_{2\cdot 1} & x_{5\cdot 2} & \dots & x_{8\cdot n} \\ x_{3\cdot 1} & x_{6\cdot 2} & \dots & x_{9\cdot n} \end{Bmatrix} \tag{56}$$

$$\begin{array}{c} \text{Genes} \\ \begin{array}{l} \text{Alleles} \\ \mathcal{Q}= \end{array} \begin{Bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 1 \end{Bmatrix} \end{array} \tag{57}$$

Optimize Meta-heuristic Settings.

Like many meta-heuristics, our perturbation engine has a number of tunable hyperparameters. While some of these hyperparameters appear to be robust across many values; others are sensitive and must be studied. A designed experiment (Figure 19) was conducted to identify hyperparameter settings that maximize cross -entropy between the class prediction and the true label. A secondary response of runtime was logged but not optimized. Potentially important factors including payload size, number of generations, initial population size, retained population size, and percent mutation were studied to find optimal levels.

A full factorial experimental design, shown in Table 6, was constructed with these five factors resulting in 32 design points. Each design point was tested four times in order to increase the test's expected power with respect to the noisy response. This experimental design is performed independently for the four most frequent attack types in the data. The payloads used in this experiment were drawn randomly with replacement from the surrogate train set. Replacement was necessary because there were in sufficient unique payloads in the surrogate model training set for the slowloris attack type. The payload size was held constant for the test points of the SSH attack type because the set only contained payloads of the larger size, over 600 bytes; there were no SSH payloads in the surrogate training set with small payloads, fewer than 400 bytes.

First order regression models were constructed with a significance of 0.05. Interactions were not considered because they had no tangible meaning for the system. The models were revised iteratively until they included only significant factors and

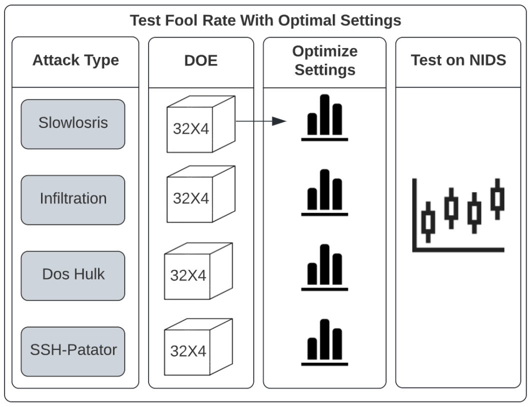


Figure 19. A designed experiment is used to select meta-heuristic settings for creating adversarial examples.

satisfied model adequacy tests. The regression models were then used to inform optimal meta-heuristic parameter settings under the assumption that payloads which demonstrated high loss on the surrogate model are more likely to fool the four NIDS models. All computational experiments were performed on an Intel Xeon Platinum 8260 and four cores.

Algorithm 3: Major tasks to generate adversarial examples

```
Input   : Set $\mathcal{X}$ of labelled binary payload packets
Output: Set $\mathcal{A}$ of adversarial examples; summary statistics
$\mathcal{X}$ = EncodePayload ($\mathcal{X}$)
split $\mathcal{X} \rightarrow$
surrogate train $\mathcal{S}_{train}$; Surrogate test $\mathcal{S}_{test}$; NIDS train $\mathcal{N}_{train}$; NIDS test $\mathcal{N}_{test}$;
TrainCNN($\mathcal{S}_{train}$)
for $j \leftarrow$ to num(payloads) do
    encoded packet $\leftarrow$GenerateTensor(payload)
    corpus $\leftarrow$CreateCorpus(encoded payload)
    random AEs$\leftarrow$GeneratePopulation(encoded payload)
    for $k \leftarrow$ to num(Iterations) do
        fitlist$\leftarrow$EvalFitness(AEs)
        AEs$\leftarrow$ReducePopulaton(AEs) parents$\leftarrow$SelectParents(AEs)
        children$\leftarrow$Crossover(parents)
        AEs$\leftarrow$AEs+children
    end
end
Output: List of adversarial examples
```

Fool Rate of Adversarial Examples.

Optimal parameters for generating constrained adversarial examples with high loss are obtained from the designed experiment. The parameters are assigned based

Table 6. A full factorial design of experiment is performed with 4 repetitions for each attack type.

Point	Payload Size	Generations	Population	Initial Population	Percent Mutate
1	Small	1000	10	50	5
2	Small	1000	10	500	5
3	Small	1000	50	50	5
4	Small	1000	50	500	5
5	Large	1000	10	50	5
6	Large	1000	10	500	5
7	Large	1000	50	50	5
8	Large	1000	50	500	5
9	Small	1000	10	50	10
10	Small	1000	10	500	10
11	Small	1000	50	50	10
12	Small	1000	50	500	10
13	Large	1000	10	50	10
14	Large	1000	10	500	10
15	Large	1000	50	50	10
16	Large	1000	50	500	10
17	Small	5000	10	50	5
18	Small	5000	10	500	5
19	Small	5000	50	50	5
20	Small	5000	50	500	5
21	Large	5000	10	50	5
22	Large	5000	10	500	5
23	Large	5000	50	50	5
24	Large	5000	50	500	5
25	Small	5000	10	50	10
26	Small	5000	10	500	10
27	Small	5000	50	50	10
28	Small	5000	50	500	10
29	Large	5000	10	50	10
30	Large	5000	10	500	10
31	Large	5000	50	50	10
32	Large	5000	50	500	10

on the results of Section 4.3 for infiltration, slowloris, hulk, and SSH attacks. Then, for each attack type, the meta-heuristic attempts to generate adversarial examples of 100 payloads randomly selected, with replacement. Each raw payload is first tested against the surrogate model and each of the three NIDS models. The meta-heuristic then generates 500 random perturbations of each raw packet or an initial population of chromosomes where the perturbations are the result of functionally equivalent substitutions taken from a corpus. In particular, the case of ASCII characters A-Z is randomly exchanged.

These randomly generated chromosomes are also tested against the surrogate and three NIDS models. The meta-heuristic then performs 5,000 generations of crossover and mutations on each payload. The meta-heuristic maintains a population of the 50 most fit unique chromosomes. Upon termination, each member of this population is tested against the surrogate model and each NIDS. If any one member of the population fools a particular NIDS, that payload is considered successful against the NIDS. This follows from a reasonable assumption that the attacker is capable of sending hundreds of packets, sometimes redundant packets, over the duration of an attack. Any single instance of the packet will perform its task when it penetrates the target NIDS. Any redundant packets that do not penetrate the NIDS would not reduce the efficacy of penetrating packets. We report the number of payloads for which at least one perturbation fools the NIDS. Algorithm 1 details the major tasks performed to generate the constrained adversarial examples.

4.4 Results and Discussion

Designed Experiment.

The designed experiment was performed to identify the optimal meta-heuristic settings to maximize cross-entropy of four types of attack packets: infiltration, slowloris, hulk, and SSH. While cross-entropy was the primary response, runtime to perturb each payload was also logged. Unfortunately, however, it was not possible to control for supercomputer resource availability. Therefore, analysis was not performed for runtime. The overall classification accuracy and recall of the CNN surrogate model on the surrogate test data is presented in Table 8. The accuracy and recall for the CNN NIDS, FNN NIDS, and Adaboost NIDS model on the NIDS test data is presented in Table 9. Although the CNN NIDS appears to capture the distribution of its respective test data better than the CNN surrogate, both CNN models are superior to FNN and Adaboost models according to accuracy and recall. Cross-entropy on the surrogate model was used as the response for this experiment.

Of the five potentially important factors affecting cross-entropy of perturbed payloads, only *number of generations* was important for the infiltration and slowloris attacks with $\alpha = 0.05$. *Payload size* and *number of generations* were statistically important for Hulk and SSH attacks attacks with $\alpha = 0.05$. Although significant factors were found for each attack type, the predictive capability of models varied. R^2 of the regression models ranged from 0.070 for SSH, to 0.83 for Hulk. The settings that maximize cross-entropy of the resulting payloads with respect to the surrogate model are provided in Table 7. It seems plausible that cross-entropy is maximized with large payloads because larger payloads provide the meta-heuristic greater freedom to perturb the most influential characters in combinations.

Figure 20 demonstrates that best observed loss is monotonically increasing with generations for every trial. In fact, the meta-heristic is designed to retain children chromosomes only if the child is strictly better than the best known solution in the meta-heuristic. This feature guarantees that any chromosome in the population is unique from all other chromosomes in the population and the claim of uniqueness has been validated experimentally. The incorporation of the uniqueness feature was useful to maintain diversity in population and to escape local maxima. In particular, Figure 20 shows that the parents chosen for each generation range in their fitness from cross-entropy nearly zero to over 2.0. Children, shown in blue triangles, are often much more fit than either of the parents, represented with green dots. The best known solution is plotted with a blue line. The suprema and infima of threshold cross-entropy to fool the surrogate are shown with a grey dotted line. Additional trials, not part of the designed experiment, demonstrated that the meta-heuristic continued to improve the best known solution well beyond 5,000 generations, with probability greater than 0. Although better solutions are always desirable, the improvements generally diminished after 5,000 generations and it was unpractical to dedicate additional runtime and memory for longer runs for the experimental design.

Resulting Fool Rates of Adversarial Examples.

Constrained adversarial examples were generated using optimal meta-heuristic settings and tested against the surrogate model and three NIDS models. 100 payloads were tested for each of four attack types, infiltration, slowloris, hulk, and SSH. Tables 10-13 are provided in order to convey the success of near optimal adversarial examples against each of the NIDS models. The first column reports the percent of raw packets that fool the surrogate and each NIDS, as well as the loss of the raw packet when tested on the surrogate model. The second column reports the percent of packets for which at least one random perturbation fooled the surrogate and each NIDS as

Table 7. Statistically significant factors for cross-entropy with significance of 0.05

Initial Population	Payload Size	Generations	Population Size	% Alleles Mutate
Infiltration	NA	5000	Large	NA
Slowloris	NA	5000	Large	NA
Hulk	NA	5000	NA	NA
SSH	NA	5000	NA	NA

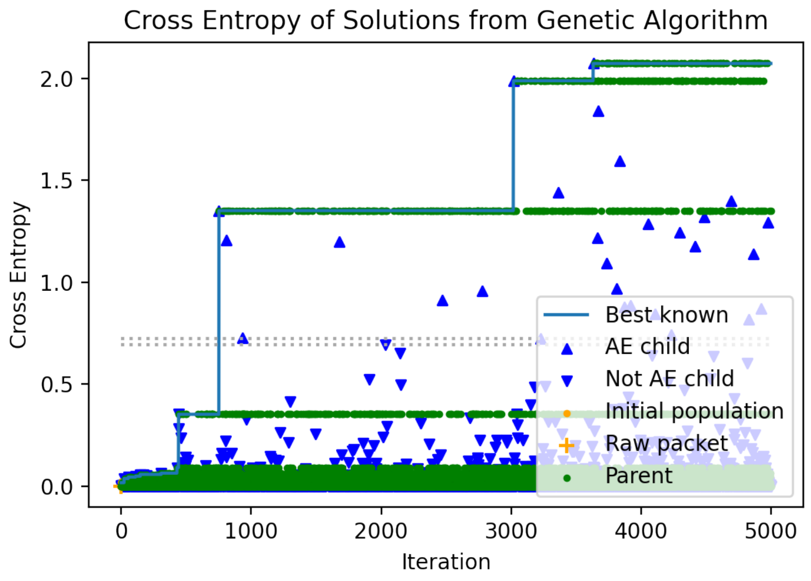


Figure 20. The meta-heuristic maximizes the cross-entropy of an SSH payload by performing crossovers and mutations of the best known solutions at each generation. The suprema and infima of threshold cross-entropy to fool the surrogate are shown with a grey dotted line.

Table 8. Test accuracy and recall is measured on the surrogate model using data sequestered prior to training.

	Test Accuracy	Test Recall
CNN Surrogate	0.995	0.995

Table 9. Test accuracy and recall is measured on three NIDS models using data sequestered prior to training.

	Test Accuracy	Test Recall
CNN NIDS	0.999	0.999
FNN NIDS	0.994	0.992
Adaboost NIDS	0.987	0.992

well as the average loss of randomly perturbed packets when tested on the surrogate model. Similarly, the third column reports the percent of packets for which at least one near-optimal perturbation fooled the surrogate and each NIDS, as well as the average loss of near-optimally perturbed packets when tested on the surrogate model. The fool rates are displayed in Figure 21. Within each attack type, the fool rate is plotted for raw packets, randomly perturbed packets, and near-optimally perturbed packets. This information is color coded by attack type. The initial hypothesis is that for any set attack and model, the values would increase as we move right from raw, to random, to near-optimal perturbations. The results convey that this hypothesis is a the general trend, but not the rule.

Only one raw infiltration packet fooled the surrogate model, and that packet also fooled the Adaboost NIDS. 6% of slowloris raw packets fooled the CNN surrogate model; however, none of those specific packets fooled the CNN NIDS. 43% of raw slowloris packets fooled the FNN classifier. All classifiers appeared accurate against raw hulk and slowloris attacks, with no raw payloads fooling any classifier. Raw slowloris packets demonstrated, by far, the greatest average loss when tested on the surrogate model and raw slowloris packets were misclassified at the highest rate against each classifier. This may be due to the great diversity of the ASCII content in slowloris payloads. Although slowloris payloads are well represented in the training data, the patterns could be more difficult to capture with the chosen model architectures. Packets that fooled the surrogate model did not always fool the NIDS. One raw slowloris packet which was misclassified by the CNN surrogate with a loss of 4.94 surprisingly did not fool the NIDS of the same architecture.

Functionally equivalent random perturbations were generated for 100 payloads of each attack type. Particularly, 500 randomly perturbed examples were generated to populate the initial population for each payload, and each of these was tested against the NIDS models. Figure 21 shows randomly perturbed packets fooled the surrogate and NIDS models at a much higher rate than unperturbed packets. Slowloris packets fooled classifier more than any other packet type. At least one of the 500 random perturbations fooled the surrogate model for 61% of the 100 slowloris packets. 56% of packets yielded at least one random solution that fooled the CNN NIDS and FNN NIDS while 55% of slowloris packets yielded at least one random solution that fooled the Adaboost NIDS. Fewer packets demonstrated fooling models with random perturbations for infiltration attacks and hulk attacks. Zero random perturbations of SSH attacks fooled any NIDS. Only one randomly perturbed SSH packet fooled the surrogate. The high classification accuracy on SSH packets may be because all SSH payloads share *nearly identical* payloads. Classifiers may learn the class manifold precisely and therefore detect perturbations at a high rate. The Adaboost NIDS was the most robust model to random perturbations for all attack types.

Near optimal perturbations were generated for 100 payloads of each attack type. The top 50 observed solutions, with respect to the surrogate model, were retained for each payload and tested against the NIDS models. These fool rates are reported in Tables 10-13 and plotted in Figure 21. The best known near optimal perturbation fooled the surrogate model with 81% of infiltration attacks, 100% of slowloris attacks, 100% if hulk attacks, and 37% of SSH attacks. Near optimal examples transferred to NIDS classifiers with varying degrees of success. Despite slowlowis and hulk perturbations fooling the surrogate model in all instances, only 69% and 37% of payloads produced any perturbation that fooled the NIDS of the same CNN architecture, respectively. Fool rates against the FNN and Adaboost NIDS were lower across the board. Only payloads of slowloris fooled all three NIDS with a high degree of success, with fool rates against CNN, FNN, and Adaboost at 69%, 53%, and 52%. Concomitant to the poor capability of randomly perturbed SSH payloads to fool NIDS, all near optimal

SSH payloads also failed to fool each NIDS. Many adversarial examples that appear very fit to the meta-heuristic transfer well against target NIDS classifiers. Among the optimally perturbed packets of slowloris, there is a 63% correlation between CNN surrogate loss and the rate at which perturbations of that packet fooled the CNN NIDS. The correlation between surrogate loss and rate to fool FNN is 77%, and the correlation between surrogate loss and rate to fool Adaboost is 54%. There may also be a connection between number of characters in the payload and the rate of fooling each NIDS. The correlation is 0.59, 0.77, and 0.32 for the CNN, FNN, and Adboost NIDS. Average payload lengths were similar across attack types. These findings confirm the results of the experimental design that perturbing larger payloads generates stronger constrained adversarial examples than perturbing smaller payloads.

The initial hypothesis of this work was that near-optimally perturbed payloads generated from a CNN surrogate would always fool any NIDS at a higher rate than random perturbations. The empirical results are more nuanced. The advantage of using the nearoptimal perturbations is best seen when the adversarial examples are tested against a NIDS of common architecture. It is evident by *comparing % packets that fool the surrogate* in Tables 10-13 that near-optimal perturbations are the clear choice if the target NIDS is known (or highly expected) to be a CNN and the surrogate is also a CNN. Random perturbations are effective against some NIDS because they are extremely inexpensive to generate and it is only necessary for one example to fool the NIDS. We characterize the random perturbations as a *brute force* strategy that provides value to the attacker and sometimes outperforms the near-optimal perturbations. In particular, we identify at least one randomly perturbed hulk payload to fool the FNN NIDS for 12% of payloads, but the near-optimal perturbations of hulk payloads are not observed to defeat the FNN NIDS. Among other classifiers and attack types, the near-optimal payload fools at rates that are similar or much better than random perturbations. Ocular inspection of the ASCII suggests that slowloris payloads are too varied to be well learned by any classifier. The meta-heuristic, therefore, stands on weak footing to provide a generalized constrained adversarial example that is "best" in all settings. Fool rates of near-optimal perturbations slightly underperform random perturbations for the highly varied slowloris attacks. On the other extreme, there is no variance among the ASCII content of hulk payloads and minimal variance for SSH. While the surrogate model learns the hulk and SSH attack patterns with high confidence, so do the NIDS models. Near -optimal perturbation of hulk attacks worked well against the CNN NIDS, but did not transfer with any success to the FNN and Adaboost models. Despite some success against the surrogate, no SSH packet fooled any NIDS.

What is responsible for the underwhelming transferability of fooling packets from the CNN surrogate to other models? Firstly, the meta-heuristic always accepts perturbations that increase loss with respect to the surrogate. This is shown by the monotonically increasing *best known* solution in Figure 20. It also follows that constrained adversarial examples which fool the CNN surrogate would often fool the CNN NIDS because they share a common architecture and joint distribution of training data. The CNN architecture is superior to the FNN and Adaboost models according to all metrics. Reference Tables 8 and 9 for overall accuracy and recall of each model, and Tables 10-13 for fool rates of raw payloads. Only the CNN captures semantic and temporal information in the payloads. The joint distribution of attacks learned by the CNN is possibly more nuanced than the other models. Perturbations generated with the CNN surrogate drive the example off the learned manifold, but not the learned manifold of the FNN and Adaboost models. The particulars of this analysis are difficult to understand because the surrogate and each NIDS are black box models and because raw packet detection is state of the art within the literature [45, 46].

This work represents the first known study of constrained perturbations of malicious cyber packets. Extensive efforts in formal methods and controlled experimentation could someday uncover the key to transferability of adversarial examples in the cyber domain.

4.5 Conclusion

This work is the first to formulate and solve the constrained optimization problem to generate an end-to-end adversarial attack in the cyber domain, specifically network intrusion detection. The solution leverages a biologically inspired metaheuristic that iteratively perturbs code within the payload of a cyber packet with functionally equivalent code. This strategy drastically improves the fool rate of malicious payloads compared to a brute force strategy and it guarantees the resulting payload is functionally equivalent to the original. The positive results are an important advancement in cybersecurity because previous research of adversarial machine learning in the cyber domain does not operate with functional packets, but rather aggregate features from network flow data. Adversarial examples generated from features cannot be reverse engineered into an end-to-end attack but perturbed packets can be. The results confirm prior beliefs that network intrusion detection systems are vulnerable to adversarial attacks. Our novel formulation extends beyond the cyber

Table 10. Classification performance for infiltration raw packets, best performing randomly perturbed packets, and best performing near-optimal packets against the surrogate and NIDS models.

Infiltration			
	Raw	**Random**	**Near Optimal**
% Packets Fool Surrogate	1%	4%	81%
Avg Loss Surrogate	3.88E-02	2.57E-01	3.73E+00
% Packets Fool CNN NIDS	0%	6%	11%
% Packets Fool FNN NIDS	0%	6%	2%
% Packets Fool Aaboost NIDS	1%	3%	3%

Table 11. Classification performance for slowloris raw packets, best performing randomly perturbed packets, and best performing near-optimal packets against the surrogate and NIDS models.

Slowloris			
	Raw	**Random**	**Near Optimal**
% Packets Fool Surrogate	6%	61%	100%
Avg Loss Surrogate	3.03E-01	4.93E+00	1.44E+01
% Packets Fool CNN NIDS	5%	56%	69%
% Packets Fool FNN NIDS	43%	56%	53%
% Packets Fool Aaboost NIDS	12%	55%	52%

Table 12. Classification performance for hulk raw packets, best performing randomly perturbed packets, and best performing near-optimal packets against the surrogate and NIDS models.

Hulk			
	Raw	**Random**	**Near Optimal**
% Packets Fool Surrogate	0%	13%	100%
Avg Loss Surrogate	4.91E-06	3.77E-01	8.34E+00
% Packets Fool CNN NIDS	0%	12%	37%
% Packets Fool FNN NIDS	0%	12%	0%
% Packets Fool Aaboost NIDS	0%	0%	0%

Table 13. Classification performance for SSH raw packets, best performing randomly perturbed packets, and best performing near-optimal packets against the surrogate and NIDS models.

SSH			
	Raw	**Random**	**Near Optimal**
% Packets Fool Surrogate	0%	1%	37%
Avg Loss Surrogate	1.68E-06	3.11E-02	1.38E+00
% Packets Fool CNN NIDS	0%	0%	0%
% Packets Fool FNN NIDS	0%	0%	0%
% Packets Fool Aaboost NIDS	0%	0%	0%

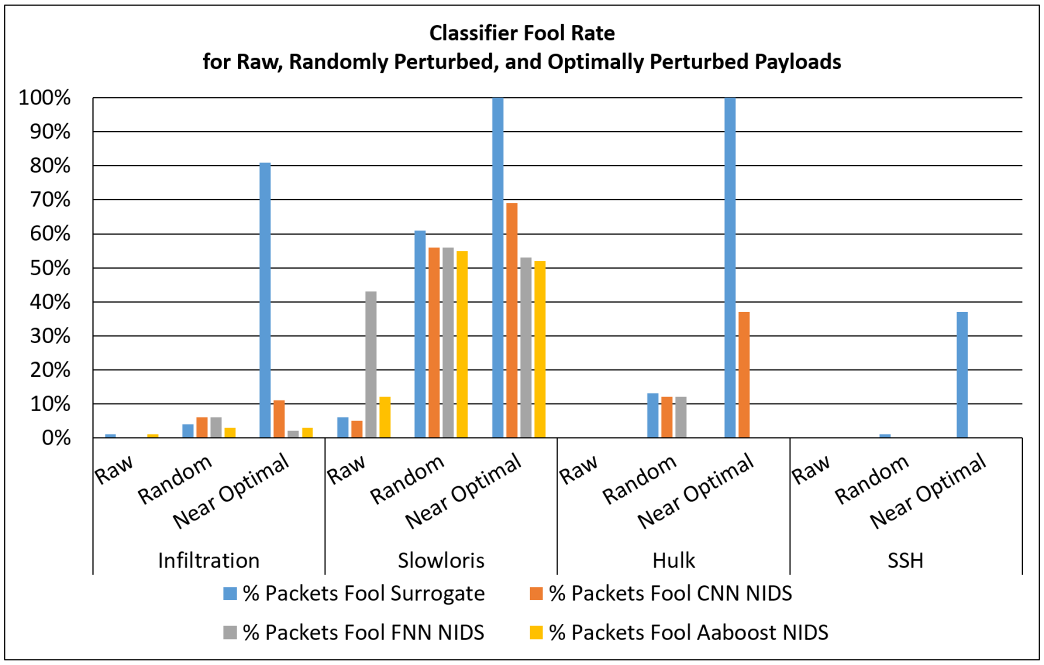


Figure 21. Classification performance is presented for raw packets, best performing randomly perturbed packets, and best performing near optimal packets against the surrogate and NIDS models.

use case to spam blockers, malware protection, and banking fraud.

Analysis of the controlled experiment determined that the *number of generations* and *size of payload* are important factors for generating strong constrained adversarial examples. With optimal settings established, a second experiment reveals that adversarial examples of slowloris packets transfer effectively when tested on a CNN, FNN, and Adaboost NIDS. Transferability of adversarial examples was less effective using payloads of infiltration and hulk attacks, and it was completely ineffective for SSH attacks, where machine learning models detect all attacks with high confidence.

The performance of constrained adversarial examples generated by substituting individual letters is promising, albeit imperfect. Many alternative classes of substitution are possible, however, assuming a corpus of equivalent units is available. Rather than substituting individual letters, future research strategies could replace entire commands, variable names, or more complex structures such as a line of code.

The meta-heuristic proposed herein will handle any choice although corpus generation will require domain expertise for advance cases. Further work must explore why constrained adversarial examples of network packet payloads do not always transfer well across every machine learning classifier type whereas unconstrained adversarial examples in the image domain do transfer well.

Cyber criminals, state sponsored terror groups, and adversary nations utilize the cyber domain to carry out crimes and as an extension of kinetic warfare to achieve strategic objectives and promote their political agendas. Adversarial machine learning attacks have been used to degrade computer vision systems in real world settings; however, it is much more difficult to conduct an adversarial attack on cyber systems because perturbations typically corrupt the payloads of cyber packets. This research helps solve the problem of constrained optimization to generate adversarial examples in the cyber domain and respects strict constraints ensuring the output payloads function precisely the same as the unperturbed payloads.

This work exposes a previously unpublished vulnerability of AI-based NIDS; however, it also motivates work towards a solution. We propose that meta-learning strategies that combine generative modeling with ensembling techniques may be fundamental to ensure that next generation NIDS are robust and resilient against conventional cyber attacks and adversarial machine learning attacks. Future work will investigate out-of-distribution (novelty) detection and generation for NIDS, as well as develop methods for self-sustaining NIDS leveraging incremental and self-supervised machine learning that evolve with adversarial behavior.

V. Demonstrate a Robust Intrusion Detection System

5.1 Introduction

The information age is loosely associated with the historical time period since the invention of the transistor in 1947. The transistor is fundamental to practically all advances in modern digital computing and communications. The explosion of research in this regime has led to widespread adoption of digital systems within industry. Digitization of information systems and rapid advancements in digital communication across networks has served as a major stimulus for growth of the world economy. This period of economic transformation is known to economists as the third industrial revolution. Cyber is a term describing the environment for communication between computers across networks. Today, cyber technologies have permeated across all areas of society including business, healthcare, government, and military.

The United States Department of Defense defines cyberspace as "a global domain within the information environment consisting of the interdependent networks of information technology infrastructures and resident data, including the Internet, telecommunications networks, computer systems, and embedded processors and controllers"[167]. This perspective of cyberspace portends a reality where digital communications are not only ancillary to warfighting across land, air, and sea, they are indispensable[168].

The internet of battlefield things (IoBT) embodies the shift to connected warfighting. It is a tactical model that connects sensors, computers, and humans to enhance rapid battlefield decisions and actions. By integrating new technology, the IoBT model accelerates the timeline of the OODA (observe, orient, decide, act) loop tactic.

In peacetime and in war, cyber systems operate in a contested information environment. Cyber attacks take on many forms ranging from degradation of capabilities to cyber-attacks with kinetic effects such as Stuxnet. The National Cyber Strategy of the United States underscores the importance of cybersecurity as a matter of national security. By bolstering the security of the United State's networked infrastructure, the strategy seeks to promote innovation and prosperity, deter aggression, and protect the internet as an open and reliable place for commerce. Our research helps answer this call to action.

The aim of this research is to provide a solution to the recently exposed vulnerability of adversarial attacks against network intrusion detection systems. Adversarial attacks are more difficult to conduct in the cyber domain than in the computer vision domain. The predominant challenge is maintaining cyber packet functionality as it is modified to bypass the detector. These attacks are therefore cutting edge and we provide the first known defense against them.

Our research contributions are to 1) provide a flexible meta-learning framework to combine imperfect base learners 2) demonstrate the trade-off between accuracy and adversarial examples detection rate for various base learners 3) demonstrate that an ensemble provides the most efficient trade-off of overall accuracy and adversarial example detection rate 4) test the amount of adversarial training required for robustness.

The proposed solution incorporates stacked ensembles and adversarial training, two technologies that have been utilized to improve classification in the domain of computer vision. We combine these technologies into an enhanced solution and use it to solve the network intrusion detection problem.

5.2 Literature Review

A comprehensive review of literature is provided to highlight developments in NIDS and the operations research methods used by cyber attackers and defenders.

References are provided for advanced readings. First, we define cyber and discuss the emergence of cyberspace. Next, we introduce cyber attacks, cybersecurity, and intrusion detection systems. Metrics are defined to assess intrusion detection performance. The section concludes with a discussion on adversarial machine learning and meta-learning.

Cyber.

The prefix cyber is believed to come from a Greek term “kybereo” meaning steer, control, or govern, and was adopted by [169] to describe his work in computer automation and communication. The field of cybernetics is an interdisciplinary study of devices with logic and control capabilities inspired by biological organisms and it was formalized in a book entitled “Cybernetics by ”[169]. Cybercommunications emerged as a derivative of cybernetics. This is a field concerned with digital communications between connected nodes, analogous to the communications between organisms in a population. Just as communities of humans are more productive than the sum of the individuals, cyber systems are enabled to make better and more complex decisions as digital communication becomes faster and more reliable [169]. Although there was always some degree of biological inspiration behind the cybernetics movement, the connotation changed over time to a discipline that studies the relationship between input and output of information systems and their autonomous behavior in their environment. Eventually, cyber became a word in its own right.

In our modern context, cyber is the environment supported by the internet and computer nodes. Other definitions emphasize the tangible infrastructure of digital communication or use cyber to describe anything digital [170]. The adjective “cyber” modifies a noun by indicating its association with the cyber domain, as in the phrase cyber space. Alternatively, cyber can be appended to the beginning of a noun to create a more specific compound word, as in cyberspace [170]. The word cyberspace seems to add emphasis on the non-tangible environment associated with cyber. The United States Department of Defense describes cyberspace as "a global domain within the information environment consisting of the interdependent networks of information technology infrastructures and resident data, including the Internet, telecommunications networks, computer systems, and embedded processors and controllers"[167]. Early computer networks allowed computers to pass limited data messages between each other if the computers adhered to strict requirements, such as shared operating system. The Advanced Research Projects Agency Network (ARPANET) was revolutionary in that it allowed compute nodes of various configurations to access a common communication network regardless of each node's hardware, operating system, or location. The computers at each end node, called hosts, used a common protocol to conduct their communication [171]. These protocols promote a secure two-way connection between hosts. Further advancements in protocol standards are given in efforts by [140] [41]. Even hosts adopting secure internet protocols proved to be vulnerable to accidental bugs and deliberate cyber attacks.

Cybersecurity.

The first known virus was deployed to ARPANET in 1969 as an informal proof of concept. The virus worked and it served as a wake-up call for the security vulnerabilities inherent in networked communications. Viruses with various purposes, typically malicious, emerged over the next several decades [172]. An industry handbook [173] from National Institute of Standards and Technology states that computer security is :

"*the protection afforded to an automated information system in order to attain the applicable objectives of preserving the integrity, availability, and confidentiality of* *information system resources (includes hardware, software, firmware, information/-data, and telecommunications).*"

We refer the interested reader to [30] for a thorough review known network threats and a discussion on NIDS approaches.

Since cyber technologies are highly integrated into all aspects of modern life, today's cyber threats can have a cyber-kinetic effect resulting in physical destruction and loss of life [7]. Most cyber-physical systems connect to the internet without any serious protection against potential exploitation. The United States Department of Homeland Security has experimentally validated a theory that cyber attacks can catastrophically disable a powerplant. Another study demonstrated that implantable cardio defibrillators could be remotely hijacked to manipulate a patient's heart function which would lead to death. Other credible vulnerabilities could lead to disabling water utilities, misdirecting trains, and even disabling automobile brakes. [7]. The Stuxnet cyber attack which degraded operations of a nuclear enrichment facility was one of the first successful cyberkinetic attacks [172] [7]. Although most cyber-physical devices in service are highly vulnerable with almost no security features, there are ways to harden new systems as they come online. Devices can be designed to only accept digitally signed instructions. Hardware can also be hardened to stay online in event of an attack. Finally, network intrusion detection systems (NIDS) should play a vital role in identifying malicious cyber packets before they are deployed to the client application.

Intrusion Detection Systems.

A 1972 study commissioned by the United States Air Force identified flaws in cyber infrastructure that allowed a dedicated attacker to penetrate and modify information systems [19]. Security breaches can inflict severe damage to the organization's materiel and information advantage. Redesigning the system was determined to greatly reduce but not eliminate the chance of a defeat. Therefore, the study called for a comprehensive effort to surveil, log, and audit activity in cyber networks [20]. The crucial audit step of network surveillance was intended to detect unauthorized users, especially those masquerading as credible users. Naturally, as the volume of cyber traffic increased, the logs became prohibitively large for reliable inspection by human analysts. By 1985, technological advances in compute power and expert systems software permitted the framework of intrusion detection systems [95]. Early NIDS leveraged a variety of rule-based expert systems and classical statistics [94]. Intrusion detection systems that operate on individual computers are known as host intrusion detection systems (HIDS). In practice, it is more efficient to monitor cyber traffic by tapping into the network at central locations. These are called network intrusion detection systems (NIDS). This research focuses on NIDS because of their widespread usage in enterprise networks, IoT and IoBT, however most methods in Section 5.3 could also be adapted for HIDS use.

There are two primary NIDS strategies. NIDS that model patterns and features of malicious cyber packets are known as *signature based NIDS* [30], [35]. NIDS that screen for out-of-distribution packets, or packets that don't fit the statistical model of normal, are denoted *anomaly based NIDS*. [105] provides a survey of anomaly detection methods. [24] details the metrics used specifically for anomaly detection in NIDS. Most NIDS research utilizes machine learning models and these are typically for the signature-based approaches [35], [30]. There are, however, some exceptions where machine learning technology is adapted for anomaly detection [32], [137]. IDS also varies widely by type of training data. [30] critiques recent NIDS research efforts and the data sets used to develop. Most NIDS use features derived from network flow data, which summarizes an internet protocol (IP) connection. Examples of these features are detailed in [108] [11]. Alternatively, IDS may derive features from individual packets [174]. [28] provides guidance on collecting and logging traffic from physical networks. [108], [39], and [31] investigate best practices in feature engineering.

Despite these advancements in feature engineering, [45] argues that most features of net flow and packet data can be spoofed with open source software. Tools such as Scapy allow users to manipulate any aspect of a packet, including IP header and payload information. [175]. Therefore, [45] advocates that NIDS should ignore the spoofable header information, and predict class based on the packet's raw, unprocessed payload. [45] and [31] demonstrate raw packet classification with accuracy as high as 99.9%.

Some scholars have postulated that packet encryption can be used to better evade NIDS. These claims are well founded because encryption obfuscates payload content from detectors. Research shows that Shamir's secret-sharing scheme [176] and randomized network proxies can be incorporated into a NIDS strategy on encrypted networks to mitigate the complications of detecting encrypted packets. Results demonstrate worst-case detection rates better than 99% on encrypted traffic [177].

Performance Metrics.

Our machine learning NIDS models are trained by iteratively adjusting parameters to minimize a loss function. Loss functions, however, cannot be interpreted as predictive power for classification problems. Therefore we calculate performance metrics on a test set. By convention, we define all correctly classified malicious packets as a true positive. A correctly classified benign packet is true negative. Conversely, an example that is misclassified as benign is a false negative. An example that is misclassified as malicious is a false positive. By testing a large set of examples, we can estimate the predictive power of the NIDS models.

Accuracy is a standard metric that reports the rate of correct classifications as a percentage. Accuracy is defined in Equation 58 as number of true positives plus true negatives divided by total examples in the test set.

Our study of NIDS is especially concerned with recall which is the true positive rate for classifying malicious packets. Equation 59 defines recall as the number of true positives divided by the quantity of true positives plus false negatives [131].

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{58}$$

Recall, also known as the true positive rate, is shown in Equation 59

$$Recall\ or\ Detection\ Rate = \frac{TP}{TP + FN} \tag{59}$$

Some test sets in this study include adversarial examples rather than unperturbed malicious examples. The detection rate of adversarial examples is calculated with Equation 59, the same formula as recall, but using a test set of only adversarial examples. To avoid confusion, we differentiate nomenclature between the two. Recall reports model performance for non-perturbed payloads and adversarial example detection rate reports classification performance of perturbed payloads.

Adversarial Machine Learning.

An adversarial attack is a deliberate attempt to fool machine learning models and inflict harm [155] [36]. One vector of adversarial attack is to alter a model's training data until the model performs poorly on unaltered test data. This is known as a poisoning attack. The other attack vector is to modify, or perturb, test examples such that they fool a classifier at test time. This is known as an evasion attack [155],

[174], [35], [31]. A survey of adversarial attack vectors is given by [178].

Deep models tend to be accurate in regions near training points but succumb to blind spots in regions further from training points. Adversarial regions occur in many subspaces of the model parameter space, meaning there are many directions that can lead to an adversarial region [36]. Since attack generation algorithms tend to be nonconvex, it is difficult to anticipate adversarial regions [178]. [179] reports that adversarial examples designed to target specific models can also fool other models, with varying evasion rates. Adding adversarial examples to training sets does harden models against adversarial attacks and diversity of adversarial examples is especially beneficial [155], [180]. [181] augments the training data of neural networks with a latent barrier class to reduce adversarial regions and promote adversarial example detection. Ensembling techniques also appear to increase robustness against adversarial attacks [108].

The majority of research in adversarial machine learning is concentrated in the domain of computer vision. Conducting adversarial attack in the domain of cyber traffic is drastically more difficult due to the risk of corrupting packet protocol or payload code during perturbation[35]. Notwisthstanding, the cyber domain is perpetually contested due to its importance in business and warfare. [11] and [35] report feature perturbation to attack feature-oriented NIDS, but the perturbed features have not been reverse engineered into functional packets. [179] perturbs actual IP packets capable of end-to-end attack. The unanswered question is whether the end-to-end adversarial attack against a NIDS can be thwarted by using existing or novel hardening techniques.

Meta-learning.

Meta-learning is machine learning task to learn about and autonomously optimize the base learning process [2]. Meta-learning processes leverage experience of prior learning to improve algorithm performance on future tasks. This strategy helps automate model selection, feature engineering, or data selection, tasks that otherwise become the burden of a fallible human analyst or inefficient grid search program [182].

Meta-features are a building block for meta-learning algorithms. The meta-features are quantifiable indicators that inform the meta-model the conditions of prior learning. In some cases, meta-features can be autonomously learned. An evaluation metric is used to quantify the efficacy of prior learning. The evaluation metric may be a model's validation loss or a performance metric such as accuracy.

[182]. There are many use cases of meta-learning that have proved beneficial in practice. Transfer learning is a method to leverage weights learned for one machine learning task unto a different, but related task. Transfer learning often provides the beneficiary model learned features, such as image edges, that are necessary for an intended task. It can drastically reduce the amount of training data or training iterations required to achieve a satisfactory model. Transfer learning that permits new tasks to train with just a few new training examples is known as few-shot learning [182], [183]. For example, a classification model may have learned weights that facilitate the detection of bicycle images. These weights can be leveraged to classify images of Segways with only minimal additional training [183]. [46] demonstrates transfer learner to reduce training required to detect network intrusions on edge devices.

Performance prediction is a broad class of meta-learning where the meta-learner anticipates the fitness of a base learner before the base learner is deployed for the specific task. [2] investigated the efficacy of meta-learning for algorithm selection of NIDS. As the NIDS received new sets of cyber data, the meta-learner selected a base algorithm that identified malicious packets with high true positive rate.

Ensemble machine learning is a meta-learning approach that combines the predictive power of multiple models to make better predictions [184]. Stacked ensemble is a framework where a meta-learner is trained to weight the predictions of base learners to so that the weighted predictions fit the training set. [103] uses a stacked ensemble strategy to improve model robustness against out-of-distribution examples of fraud and [108] uses ensembling to increase robustness of feature based NIDS. Results indicate a benefit to robustness in both cases, but for the NIDS application there appears to be a slight trade-off between robustness and accuracy of models.

5.3 Methodology

This work addresses a recently exposed vulnerability of NIDS. Raw packet NIDS achieve high performance by learning directly from the packet payload, not from engineered features. Raw packet NIDS are also favored over feature oriented NIDS because it is easy to spoof packet headers and their derivative features but it is more difficult to perturb a packet's actual payload. Despite these advantages, it is possible to conduct adversarial attack against raw packet NIDS.

The defense against this attack requires a rudimentary understanding of the attack vector. Section 5.3 outlines the meta-heuristic that generates functional adversarial examples from raw packet payloads.

Along with the meta-heuristic, the attacker uses a surrogate model to mimic the behavior of the target NIDS and generate adversarial examples.

Additional NIDS models are used to test the efficacy of adversarial attacks. Construction of the surrogate NIDS model and the baseline NIDS models are provided in Section 5.3. The strategy used to harden base models against adversarial attack is given in Section 5.3. The base learners are improved using adversarial training and meta-learning, which is outlined in Section 5.3.

Adversarial Example Generation.

Adversarial example generation is the technological key to conducting adversarial attack. This process requires high quality labeled cyber data, similar to the data of the target network. Figure 22 illustrates the four step process to generate and test adversarial examples. Step one is to remove the packet payloads from the complete packet and to assign truth labels to each payload. Details on data preparation are presented in Section 5.3. Step two is to train surrogate and NIDS models from labeled payloads. Figure 22 shows steps 2a and 2b as training the surrogate and NIDS models respectively and these are detailed in section 5.3.

Step 3 is packet payload perturbation to fool the surrogate. The perturbation process is formulated as a constrained maximization problem. The fitness function is the cross-entropy of the payload with respect to the trained surrogate model. Cross entropy is given in Equation 61 where *H(P)* denotes the hypothesized class distribution which is measured against the true class, distributed as Q. The heuristic performs steps much like a genetic algorithm. In this heuristic, however, the packet payload is viewed as a set of functional units of code. The initial population of 500 is generated by randomly substituting units of code with functionally equivalent units taken from an expertly procured corpus. By using the special corpus, the heuristic guarantees the functional equivalence of the payloads in the initial population. Each version of the payload in this initial population is evaluated by the surrogate. The 50 variants with the highest cross-entropy are retained in the population for iteration 1. In each iteration, the meta-heuristic probabilistically selects two parent payloads from the population for crossover. Highly fit payloads are preferred for selection according to the distribution in Equation 63. Crossover is performed at a randomly selected functional unit drawn from a uniform distribution. Since all versions of the payload have functionally equivalent code at any allele location, crossover always maintains the functional equivalence of the child payload. The child payloads are evaluated by the surrogate and are retained if better than the least fit payloads in the population.

The problem formulation is summarized within and complete detail is given in [31].

Objective Function

$$
\begin{aligned}
H(P,Q) &= H(P) + D_{\mathrm{KL}}(Q||P) && (60)\\
&= -\mathbb{E}_{x\sim P}\mathrm{log}Q(x) && (61)
\end{aligned}
$$

Sets

Indicators $\{i,j,k\}$ define the mechanics of the meta-heuristic:

$$
\begin{array}{ll}
i \equiv \text{allele} & i \in \{1,2,...n\},\\
j \equiv \text{gene} & j \in \{1,2,...m\},\\
k \equiv \text{test example} & k \in \{1,2,...o\}
\end{array}
$$

Decision Variables

$$
x_{ij}^{(k)} = \begin{cases} = 1 \text{ if gene } j \text{ is chosen } i \text{ for payload } k \\ = 0 \text{ otherwise} \end{cases} \tag{62}
$$

$$p(\chi = x) = \frac{(n-x)}{\sum\limits_{x=\{1,2,\ldots n\}} x} \tag{63}$$

The parent selection, crossover, and evaluation steps are repeated for 5,000 iterations. If there is an improvement to the best-known solution at iteration i, the child of iteration i+1 receives mutation with probability 0.05. Equation 64 shows the probability of mutation increases by 0.01 after each iteration without improvement. After 95 iterations without improvement, mutation is certain. After 5,000 iterations, the most fit payload in the population is reported as the near-optimal solution.

$$P(mutate) = 0.05 + (0.01)(generations\ since\ improvement) \tag{64}$$

The fourth and final step is to evaluate the proposed payload against fully trained NIDS to determine if it is indeed an adversarial example. True adversarial examples are sequestered for later use. [179] provides greater detail on adversarial example generation and analysis of adversarial example evasion performance.

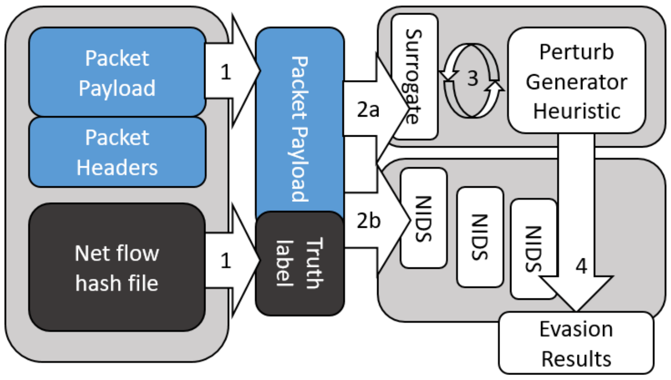


Figure 22. Generation and test of adversarial examples is presented as a 4 step process

Data Preparation.

[174] provided a pipeline for extracting and labeling the payloads of IP packets from pcap files. First, the pcap files are obtained from an existing data set or by using a feature extraction tool such as bro. Then net flow data must be obtained. Finally, a ground truth file must attribute a label to every net flow connection. Truth labels are often provided with cyber data sets but can also be inferred from meta-data such as source IP. Labels are then linked from truth data, to net flows, to pcap packets using Source IP address, Destination IP, address, Source Port number, Destination Port number, Protocol number, and time stamps. The dpkt Python tool is used to extract payloads from TCP and UDP packets. Packets of other protocols are not considered in this methodology because they lack the diversity for effective learning; packets are nearly all malicious or nearly all benign. Payloads are encoded as ASCII text integers between 0 and 255, but then normalized as a float 0 to 1. Most of these encoded feature vectors contain fewer than 1,500 characters, but feature vectors with additional units are truncated to the 1,500 limit. Feature vectors smaller than 1,500 elements are padded with zeros. [46] provides additional details on generating labelled, encoded, feature vectors from the Canadian Institute for Cybersecurity Intrusion Detection System (CICIDS) 2017 cyber data set. This study leverages the examples generated from CICIDS collection days of Monday though Friday and does not apply an additional feature engineering. Adversarial examples were generated from encoded slowlorris malicious packets. Table 14 shows the data sets used for training, validation, and testing of weak learners and ensemble models. Examples were rigorously randomized then sequestered to avoid conflation of training examples in validation and test sets.

Table 14. Raw packet feature vectors are partitioned into training, validation, and test sets. Data sets are augmented with adversarial examples. A hold-out set of adversarial examples is reserved for testing detection rate.

Names of Data Set Partitions	Number Examples	Explanation
NIDS-Train-0	74,670	Nearly balanced set of unperturbed vectors for training weak learners
NIDS-Train-1	74,671	Nearly balanced set of unperturbed vectors. Weak learners generate predictions which serve as meta-features for level 1 learning
NIDS-Val	8,296	Nearly balanced set of unperturbed vectors for model validation and selection
NIDS-Test	8,297	Nearly balanced set of unperturbed vectors to test weak learners and ensemble models
AE-Train-0	1,000	Adversarial examples generated from slowlorris payloads. Used to create NIDS-Train-0-Augmented set.
AE-Train-1	500	Adversarial examples generated from slowlorris payloads. Used to create NIDS-Train-1-Augmented set.
AE-Test	1,000	Adversarial examples generated from slowlorris payloads for testing AE detection of weak and ensemble models
NIDS-Train-0-Augmented	75,670	NIDS-Train-0 combined with AE-Train_0 for training robust weak learners
NIDS-Train-1-Augmented	75,171	NIDS-Train-1 combined with AE-Train_1. Used by weak learners to generate predictions which serve as meta-features for level 1 learning

Base Models.

Base models are trained to perform the NIDS task of discriminating malicious IP payloads from benign. Although powerful tools in their own right, the base learners are also utilized as the weak learners in the stacked ensemble robust framework. Base learners include a convolutional neural network (CNN), a fully connected neural network (FNN), and an Adaboost classifier. The surrogate model used for adversarial exampled generation is also a CNN of identical architecture but separately partitioned training data. All models have an input vector of 1,500 ASCII characters that have been numerically encoded. The CNN models have seven hidden layers including convolutional, max pooling, convolutional, max pooling, flatten, dense, and dropout. The FNN contains nine dense hidden layers. Both neural network architecture employ relu activation in hidden layers while sigmoid is used for the output layer. Within the

Tensorflow API, the adam optimizer is selected with a loss function of binary-cross entropy. Neural nets were programmed for 20 epochs and an early stop threshold 0.0005 nats with patience of 3 on the validation set. The Tensorflow dataset pipeline was used to implement minibatching with 128 examples, which was more efficient than a generator-based pipeline. Additional details of the neural nets are provided in [31]. Adaboost models were trained from the same training set using the Scikit-learn API. Models were trained with 50, 100, 150, and 200 1-layer decision trees. All other parameters were set to default. The model instances with 200 trees were selected based on validation accuracy.

Robust Framework.

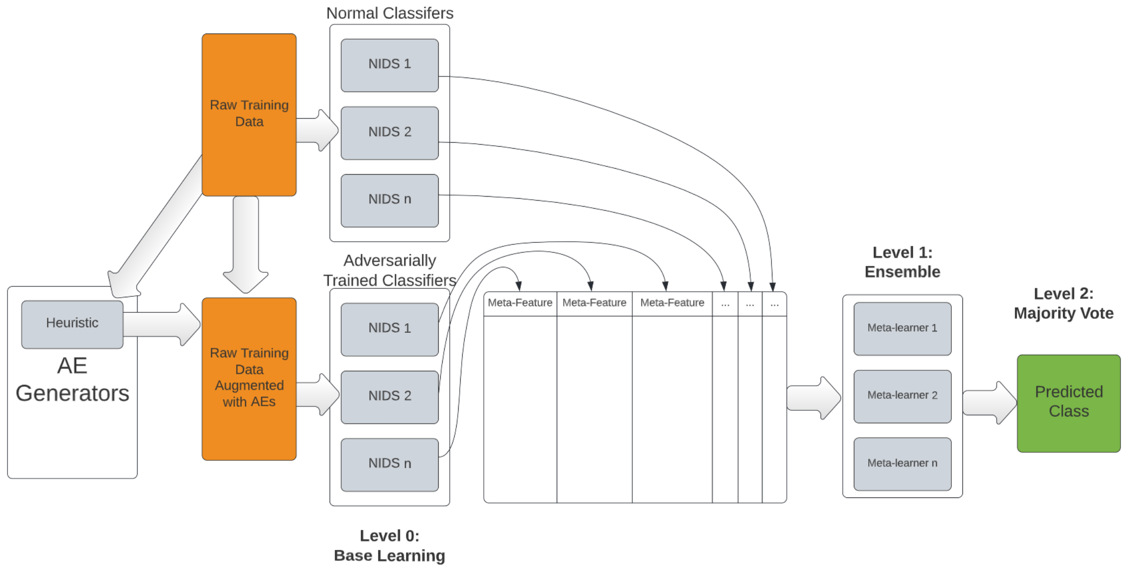


Figure 23. The meta-learning framework is hardened against adversarial attack by intelligently combining the predictive power of each base model. normally trained classifiers are trained with normal cyber traffic. The adversarially trained classifiers are trained from an augmented data set outlined in 5.3.

The primary contribution of this manuscript is an expandable framework for a NIDS that is highly accurate against cyber traffic but also robust against adversarial attack. The proposed framework includes weak learners of various categories that are incorporated in a stacked ensemble. First, and most fundamentally, a variety of weak learners are trained using the unperturbed NIDS-train-0 data set as outlined in Section 5.3. Weak learners are denoted as level 0 of the ensemble. These models are shown as the *normally trained Classifier* cluster in Figure 23. Next, models of the same architecture are adversarially trained with the NIDS-train-augmented data set. These comprise the *Adversrially trained* cluster of classifiers. This framework encourages the use of additional clusters focused on either adversarial examples or normal examples. Accuracy and recall of the level 0 models are reported with the NIDS-test dataset. Adversarial example detection rate is reported with AE-test-0 data set. Level 1 of the ensemble uses features of the weak models as inputs to predict true class. We propose training multiple level 1 meta-learners. This is demonstrated within random forest, XGBoost, and logistic regression classifiers. Each level 1 classier leverages the combined predictive power of the various weak learners. Level 1 models should yield high accuracy when trained and tested with quality cyber data. Majority vote is used as a level 2 meta-learner to resolve disagreements for final decision on each test example.

5.4 Results and Discussion

Six base learning models were trained with the NIDS-Train-0 data set and model selection was performed with NIDS-Val. The first three models include a CNN, a FNN and an Adaboost classifier trained on the NIDS-Train-0-dataset. These models are denoted as normally trained models in Table 15, which reports accuracy, recall, and aversarial example detection rate. Additional instances of CNN, FNN and Adaboost were trained with the NIDS-Train-0-Augmented data set. Models trained on the augmented data set are referred to as adversarially trained models in Table 15.

Figure 24 demonstrates the early stop strategy for training the neural nets. Although the number of epochs varied by model, all model instances demonstrated rapid improvements to fit early on followed by asymptotic improvements to training loss, validation loss, training accuracy, and validation accuracy.

Among the normally trained base models, the CNN provides the best accuracy and the CNN is tied with the FNN for best recall against normal packets. Despite slightly lower accuracy and recall, the Adaboost classifier demonstrates the best detection rate against the adversarial examples. This finding is similar to the findings of [31] which reports that CNN is most vulnerable to adversarial attacks generated from a CNN surrogate. These results also confirm previous observations that Adaboost has some natural robustness against that adversarial attack. It is also clear from these metrics that adversarial training provides significant hardness against adversarial evasion attacks, yielding a perfect detection rate in some cases. The CNN once again reports the lowest detection rate, although the disadvantage compared to FNN and Adaboost is small. While the CNN and FNN report slightly lower accuracy and recall from adversarial training, the Adaboost model maintains or slightly improves on these metrics. The accuracy reported in these models is slightly lower than the accuracy of comparable models in [31], perhaps due to the decreased number of examples in each training set caused by splitting the training set into two partitions.

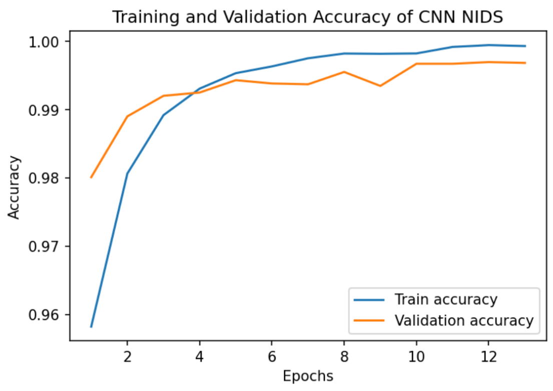
Three types of meta-models are used to create level 1 ensembles from the predictions of the 6 base models. The meta-learners are random forest, XGBoost, and logistic regression. They are trained from predictions of the weak learners on the NIDS-Train-1 data set as predictors, and the ground truth as the target. These three meta-learners are then retrained using the base predictions from the NIDS-Train-1-Augmented set. Although all the meta-learners incorporate both normally trained and adversarially trained weak learners, it is only the meta-learners that are themselves adversarially trained that learn to leverage the robust base models. The detection rates of each level-1 meta-model are plotted in Figure 29a as a function of the importance attributed to adversarially trained base models. The red points in Figure 29a represent the adversarially trained meta-learners; green are normally trained. It is clear from the figure, and confirmed by Table 16 that the adversarially trained random forest and the adversarially trained logistic regression place most of their importance on the three adversarially trained base classifiers. Further, these two meta-learners yield perfect detection rates. Interestingly, the adversarially trained XGBoost meta-learner places nearly all importance on the normally trained base models but manages to report a near-perfect detection rate as well.

The connection between the training approach of the meta-learner and the resulting accuracy is less clear. Figure 29b and Table 15 report that the normally trained logistic regression classifier is the most accurate level-1 meta-model. Although the least accurate level-1 meta-model is the adversarially trained random forest classifier, it still yields excellent accuracy. The training approach of level-1 meta-models has no practical impact on classification accuracy.

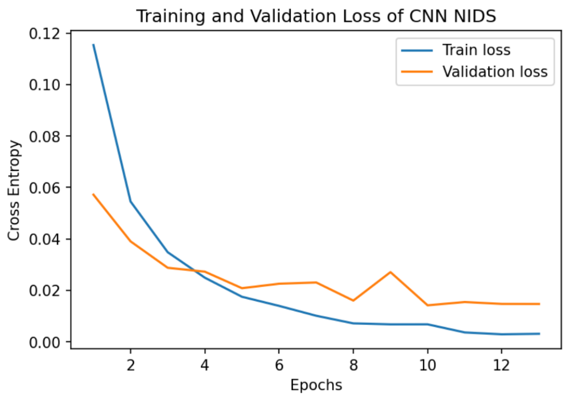
A level-2 majority vote node combines the three meta-learners for the stack using normally trained meta-learners and for the stack using adversarially trained meta-learners. In both cases, the majority vote node produces excellent accuracy and recall. Unsurprisingly, the majority vote predictions that leverage the fallible normally trained ensembles do not fare well against adversarial examples. On the contrary, the majority vote node on the adversarially trained stack detects 100% of adversarial examples.

A motivated hacker can select from a variety of cyber-attacks that cause grave damage to target networks. Fortunately, recent research reports that a well-trained machine learning model can detect almost all such attacks [2]. Adversarial evasion attacks pass the advantage back to the attacker by fooling these detection models [31]. A persistent attacker with a low, but non-zero evasion rate can still complete their attack after many evasion attempts. Cybersecurity analysts must therefore be vigilant to protect against conventional cyber attacks *and* adversarial attacks. It is difficult to argue the relative priority of detecting conventional and adversarial attacks; this manuscript reports both. It is important, however, to understand the tradeoff of these goals. We test and report accuracy and recall for unperturbed IP payloads and we report detection rate on a set of adversarial examples. No model in our test dominated all three metrics. Table 15 provides these metrics and their respective rank for each model. The logistic regression meta-learner from the normally trained stack provides the best accuracy and recall while the logistic regression model from the adversarially trained stack provides a perfect detection rate and is a close second for both accuracy and recall. Figures 26-28 shows Pareto frontiers for each pairwise plot of metrics. No model is Pareto efficient for all three frontiers, however, the augmented logistic regression is Pareto optimal for two of the three. Most decision-makers would value the high performance of the adversarially trained logistic regression ensemble against both conventional cyber attacks and adversarial attacks. The majority vote nodes never dominated performance compared to the level 0 and level 1 models in their respective stacks, however, they always outperformed the worst performing level 0 and level 1 models in their stacks. The majority vote node in the augmented stack was especially robust against adversarial examples with only a minuscule drop in performance against normal traffic. The majority vote strategy may therefore be a good choice for NIDS if the worst-case performance of lower-level models is unknown. We also recognize there is variance in model performance due to randomness in training. It is unproven that small differences in performance metrics are true effects- they could be the result of random events.

An additional experiment was conducted to investigate the effect that number of adversarial examples in the training set has on the hardness of the CNN against adversarial attacks. Trials were performed by increasing adversarial examples by one until there were 10, then by five examples for 10 through 50. Then the number was increased by 25 for 50 though 200. The resulting detection rates are plotted in Figure 25. Trials with fewer than 50 adversarial examples demonstrated a wide variance in detection rate but an overall upward trend. A logarithmic trendline is shown in Figure 25. The detection rate stabilized towards its limit value of 1.0 with just 150 adversarial examples. The impact of augmentation on test *accuracy* was also investigated and there is no evidence of a significant effect with $\alpha = 0.1$. The CNN is the most vulnerable to this specific adversarial attack among the models we investigate. It is promising to see the excellent hardness against adversarial attack with only a small number of adversarial examples. It seems that the patterns learned from conventional training permit the model to easily adapt for adversarial detection with only a small amount of adversarial training. [180] argues that diversity of adversarial examples in training sets provides additional hardness for classifiers in the image domain. This concept remains untested in the cyber domain and offers great opportunities for future work. Also untested is the concept of few-shot learning where training is performed incrementally on a previously trained model as new adversarial examples become available. We expect these general areas of transfer learning may play a role in improving NIDS hardness in the future.



(a) Validation accuracy improves during training



(b) Early stop occurs as loss fails to improve

Figure 24. The artificial neural nets were trained using an early stop callback with a patience of 3 and threshold of 0.0005 nats

Table 15. The classification performance is presented for 14 models. The first three models are base models trained on normal data. The next three models are adversarially trained from a training set augmented by adversarial examples. There are four meta-models trained with features derived from normal data. Finally, there are four meta-models trained with features derived from a data set augmented with adversarial examples. Each model is ranked by its accuracy, recall, and adversarial example detection rate.

	Metric:	**Accuracy**		**Recall**		**Detection Rate**	
	Reference Set:	**NIDS-test**		**NIDS-test**		**AE-test**	
Normally Trained Base Models	1D-CNN	9	0.996	6	0.997	14	0.311
	FNN	13	0.991	6	0.997	13	0.427
	AdaBoost	12	0.992	12	0.992	9	0.802
Adversarially Trained Base Models	1D-CNN	11	0.992	13	0.987	7	0.987
	FNN	14	0.990	14	0.983	1	1.000
	AdaBoost	10	0.992	11	0.994	1	1.000
Normally Trained Meta-models	Random Forest	6	0.997	10.0	0.997	11	0.650
	XGBoost	2	0.998	2	0.997	12	0.596
	Logistic Regression	1	0.998	1	0.998	8	0.819
	Majority Vote	2	0.998	5.0	0.997	10	0.697
Adversarially Trained Meta-models	Random Forest	7	0.997	6	0.997	1	1.000
	XGBoost	5	0.997	6	0.997	6	0.999
	Logistic Regression	2	0.998	2	0.997	1	1.000
	Majority Vote	8	0.997	4	0.997	1	1.000

Table 16. Feature importance is reported for random forest and XGboost level-1 meta-models. Regression coefficients are reported for the logistic regression level-1 meta-models. The rows are the meta-models and the columns are the base models used to generate predictive meta-features. The first three meta-models are trained with features derived from predictions on normal traffic. The bottom three meta-models are trained with features derived from a data set augmented with adversarial examples.

Feature Importance or Regression Coefficients								
	Meta-Model	**Normally Trained CNN**	**Normally Trained FNN**	**Normally Trained AdaBoost**	**Adv Trained CNN**	**Adv Trained FNN**	**Adv Trained AdaBoost**	**% Adv Trained Models**
Normal Trained Ensembles Trained with NIDS-Train-0	**Random Forest**	0.282	0.090	0.219	0.249	0.021	0.139	41.0%
	XGBoost	0.934	0.002	0.022	0.003	0.006	0.034	4.2%
	Logistic Regression	5.045	1.304	1.945	0.244	2.811	3.243	43.2%
Adversarially Trained Ensembles Trained with NIDS-Train-1	**Random Forest**	0.102	0.002	0.090	0.348	0.181	0.277	80.6%
	XGBoost	0.934	0.002	0.022	0.003	0.006	0.034	4.2%
	Logistic Regression	4.073	0.276	1.201	1.311	4.088	3.985	62.8%

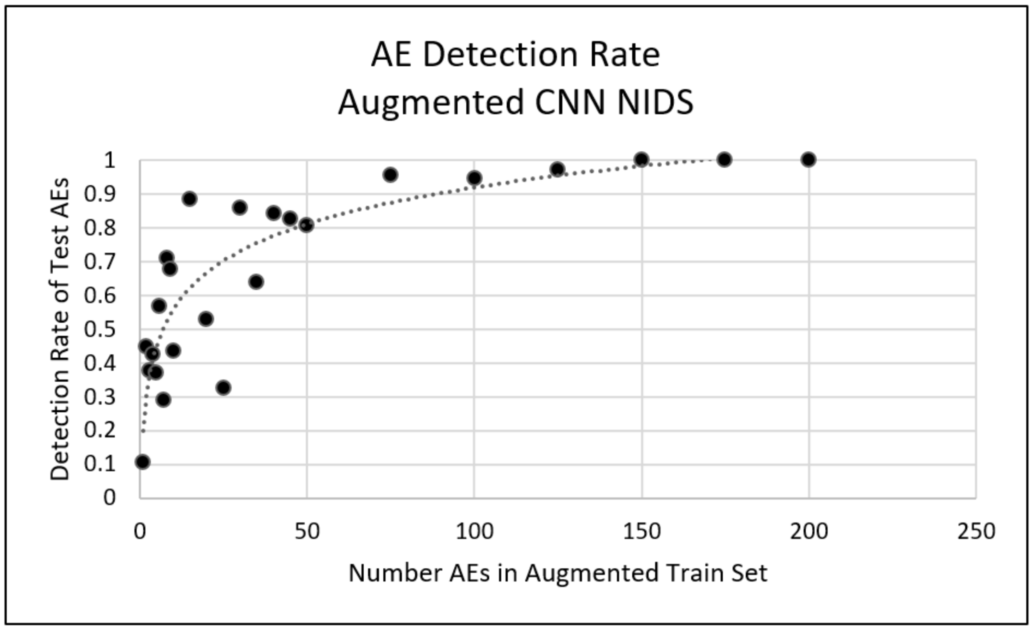


Figure 25. Detection rate of adversarial examples is plotted as AE count during adversarial training is increased

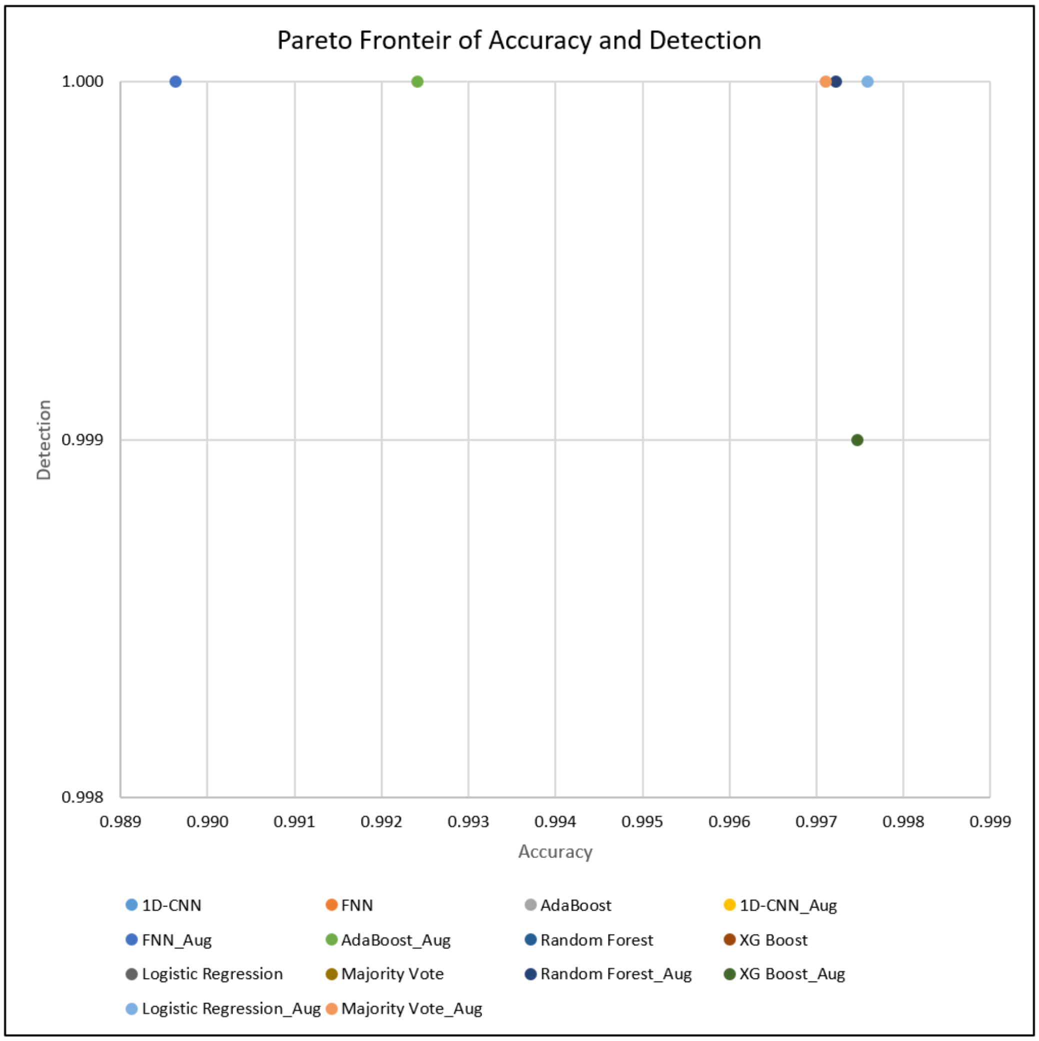


Figure 26. Pareto frontier of accuracy and detection rate

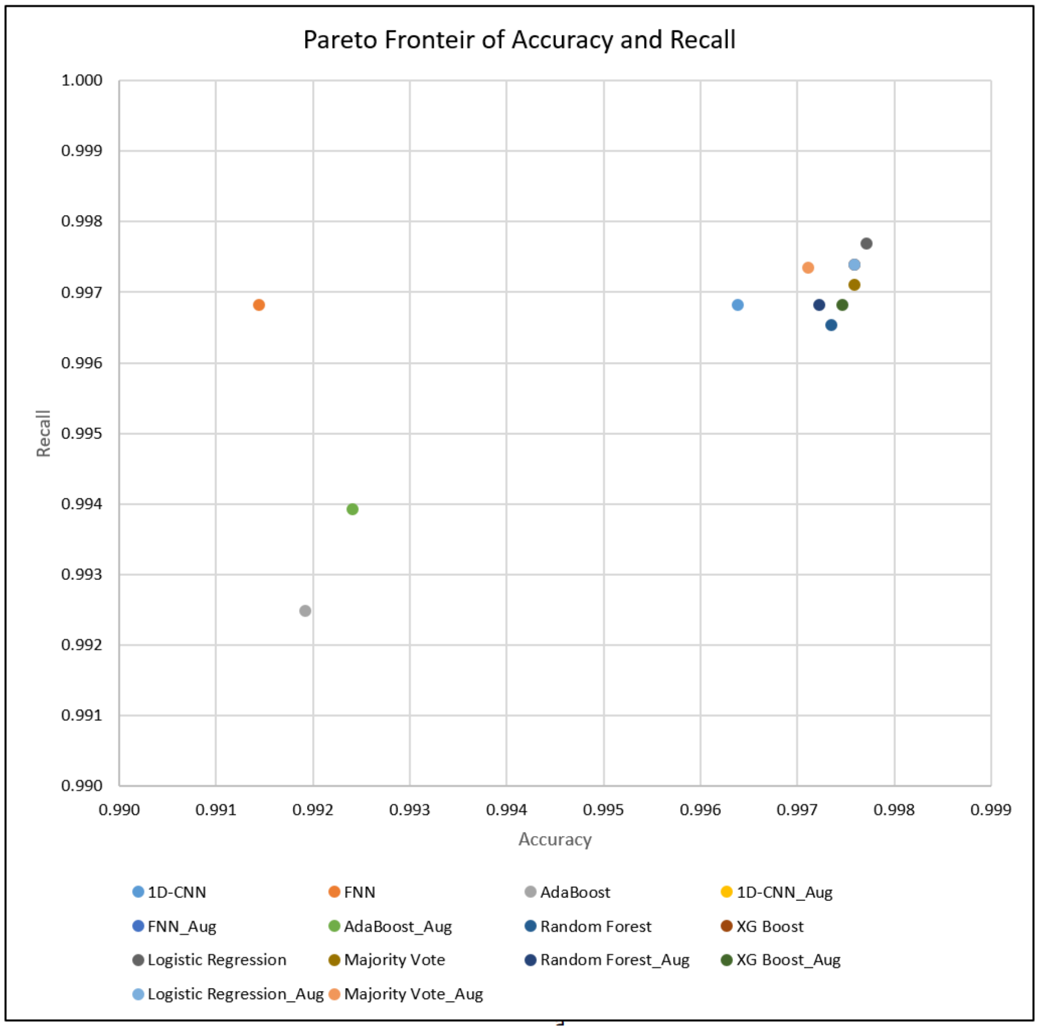


Figure 27. Pareto frontier of accuracy and recall

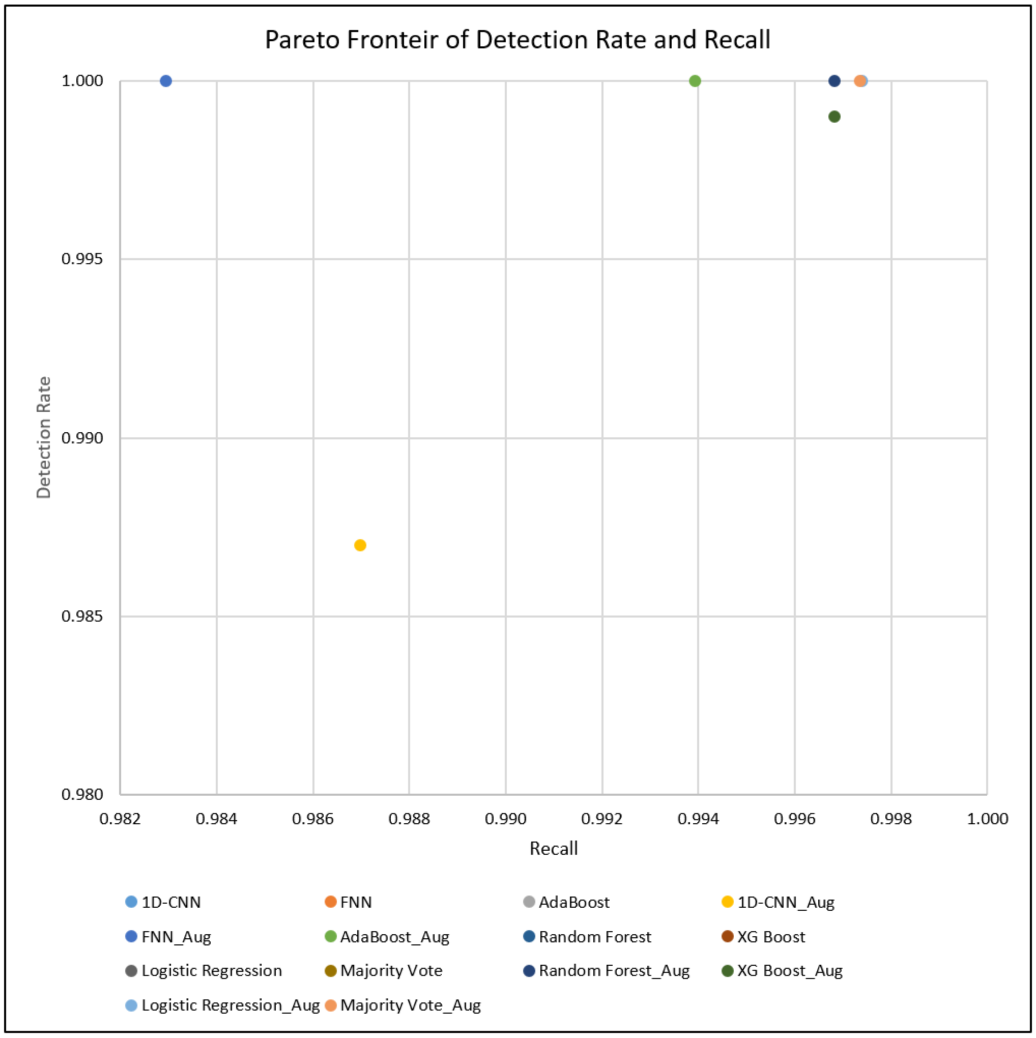
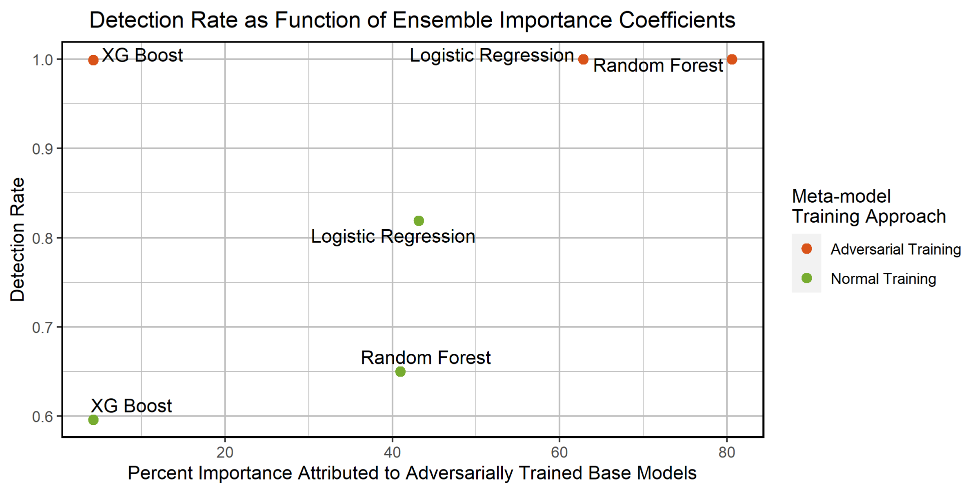
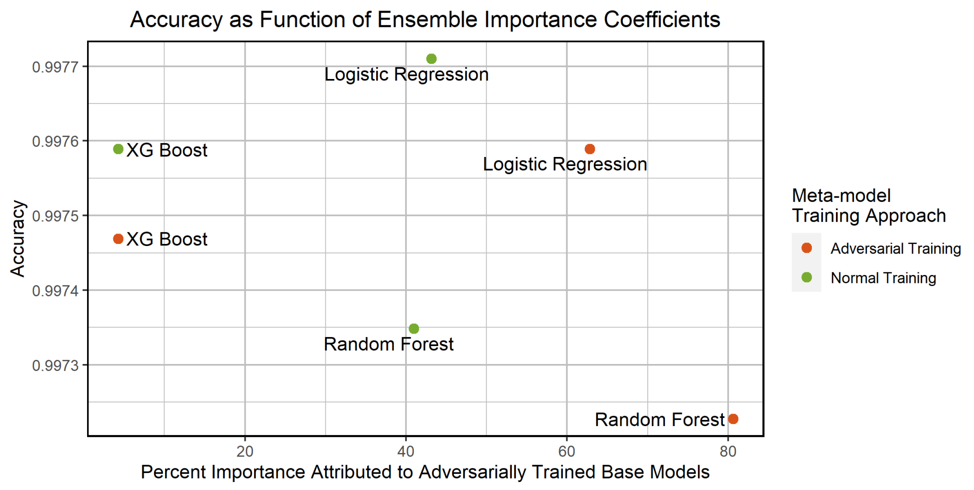


Figure 28. Pareto frontier of detection rate and recall



(a) The meta-models with the highest adversarial example detection rate tend to be adversarially trained models who's training features are derived from the augmented data set



(b) The meta-models with the highest accuracy tend to be normally trained models trained with features derived from the normal data set

Figure 29. Performance trends among meta-models that are normally trained and adversarially trained

5.5 Conclusion

This work studies NIDS hardening techniques against an adversarial attack in which slowlorris attack packets are perturbed to bypass the NIDS. Without hardening, the target NIDS, a CNN, detects only 31.1% of adversarial examples at test time. Other base models demonstrate better but imperfect detection rates. Adversarilly trained models demonstrate nearly equivalent performance against normal (benign and malicious) traffic, but drastically improved detection of adversarial examples. Stacked ensembles combine the predictive power of normally trained and adversarially trained models. The best ensemble models provide a Pareto efficient trade-off of accuracy, recall, and adversarial example detection rate if the meta-features include some examples derived from adversarial examples. That is, the meta-learner is itself adversarially trained. Combining the predictions of various meta-learners via a majority vote node does not *improve* the quality of predictions, but hedges against accidentally using a low performing base model or ensemble. We experimentally determine that the full benefit of adversarial training is obtained by including just 150 adversarial examples in the training set. We believe that further experimentation will demonstrate few-shot learning to detect additional varieties of adversarial attacks. The combination of meta-learning and adversarial training demonstrates excellent NIDS performance. As attacks evolve, the meta-learning framework can be expanded to incorporate additional weak models, including, but not limited to Bayesian nets, elastic nets, and stochastic loss functions. These advancements will support the confidentiality, integrity, and availability of the internet of things into the information age.

VI. Conclusion

Cyber systems are integrated into practically all aspects of modern society and have served as an economic stimulus as the world transitions to Industry 4.0. The United States Department of Defense has repeatedly recognized that confidentiality, integrity, and availability of cyber assets are fundamental to national prosperity. Many other groups have recognized the transformative power of cyber, leading to cyber's emergence as a contested environment. During peacetime, the Department of Defense is charged with protecting American cyber assets from attack by antagonistic nation states and non state players. Cyber has also transformed military operations. The internet of battlefield things is a paradigm that connects sensors, humans, computers, and actuators to accelerate battlefield command and control and ultimately tactical advancements across land, sea, air, and space. Efforts to degrade the connectivity of tactical cyber systems could results in confusion, degraded maneuver, and loss of firepower. Cyber is also a domain of warfare in its own right. There are reports of offensive cyber attacks downgrading critical military and civilian infrastructure around the world including water, transportation, and energy assets. Experts believe that future wars will be largely fought in the cyber domain.

The research presented in this manuscript addresses the call to action in the United States Cyber Strategy for using intelligent systems for defensive cyber strategies. The study compiles and evaluates impactful findings on the topic of cybersecurity, especially as it pertains to national defense. The first novel research contribution is an evaluation of existing cyber data, collection of high quality cyber data, and generation of synthetic cyber data. Machine learning network intrusion detection systems (NIDS) are typically impaired by a lack of high quality, labelled cyber data which is resource intensive to obtain. Experiments in this manuscript demonstrate that 85% of real cyber data can be replaced with synthetic data without loss of accuracy. The second contribution provided in this manuscript is to expose the first known end-to-end adversarial attack in the cyber domain. While adversarial attacks have been widely researched in the computer-vision domain, those strategies cannot be applied to the cyber domain due to the special structure of an internet protocol packet. The proposed meta-heuristic perturbs malicious packets to evade NIDS 61% of the time, despite the NIDS reporting 99.9% accuracy on non-perturbed cyber traffic. Crucially, the perturbations do not violate the functional integrity of the packets, allowing them to carry out the attack. The final research contribution is to protect NIDS against the newly proven adversarial threat. A framework is presented that leverages technologies used to improve models, typically in the computer vision domain. The framework uses meta-learning to combine the predictive qualities of various models. In particular, some of the base models are trained with datasets augmented with adversarial examples. The meta-learner itself is also trained with a dataset augmented with adversarial examples. This configuration results in 100% detection rate against the most effective adversarial evasion attack with no detriment to overall accuracy.

Adversarial attack against cyber infrastructure is widely considered to be a pressing concern. The results within have demonstrated that effective models can be constructed even in an operation data deficit. NIDS models are provably vulnerable to adversarial evasion attack, but the meta-learning framework offers strong protection to the worst case adversarial attack. This work provides researchers and policy makers with new knowledge to secure our cyber systems and promote prosperity.

Appendix A. Meta-learning to Streamline Algorithm Selection for Cyber

This chapter has been published in *WiseML '20: Proceedings of the 2nd ACM Workshop on Wireless Security and Machine Learning* [2]. It is included below with permission from the publisher according to their author rights agreement.

A.1 Introduction

People, organizations and communities rely on the Internet of Things (IoT) to aid in almost any conceivable task that was previously performed manually. As technology advances, components of IoT have progressed into the wireless domain [185]. These emerging systems are susceptible to attack by malicious actors wishing to degrade the system or steal proprietary information [186]. Intrusion detection systems (IDS) are central to maintaining the security of modern computer networks from malicious actors [39]. IDS have been successfully demonstrated in both the wired and wireless domain of IoT [185]. The task assigned to an IDS is to *classify* network traffic as malicious or normal. Numerous studies [185] have explored meta-models to detect malicious behavior in computer networks. Maxwell et al. [39] further focused on intelligent cybersecurity feature engineering for various meta-models.

Learning algorithms may be used to formulate a meta-model. Selection of the best machine learning (ML) algorithm, including hyper-parameters, for a particular problem instance is a difficult and time-consuming task [187]. Cui et al. [188] has confirmed conclusions of [189] and [190] that meta-models' performance varies among problem types and problem instances. Wolpert et al. [191] uses *The Extended Bayesian Formalism* to show that given a set of learning algorithms and problems, each algorithm will outperform the others for some (equally sized) subset of problems. This phenomena has driven researchers to a trial-and-error strategy of identifying the best meta-model for a given problem. The preferred meta-model is selected by comparison of model performance metrics such as accuracy [192]. Unfortunately, the computational run time and human investment required to select a learning algorithm by trial-and-error is generally prohibitive of finding the optimal choice.

This paper aims to advance the IDS body of knowledge by incorporating recent work in algorithm selection. Accordingly, an *algorithm selection framework* is introduced. The algorithm selection framework leverages a taxonomy of ML algorithms. The framework narrows down the list of applicable algorithms based on problem characterization. Two strategies are presented to select the most preferred algorithm: *rules-of-thumb* and *meta-learner.* If successful, the algorithm selection framework promotes high-performance results of the IDS and assuages the computational cost of performing multiple ML algorithms.

This paper includes Related Works in Section 2. The Methodology is presented in Section 3. Section 4 contains the Results and Section 5 is the Conclusion.

A.2 Related Works

Intrusion Detection.

As in any other domain, criminals and adversaries seek to inflict harm by exploiting weaknesses in cybersecurity systems. The rate of system intrusion incidents is increasing annually [193]. Landwehr et al. [194] provides a taxonomy of all known flaws in computer systems. Special attention is provided to the category of flaws that allow exploitation by malicious actors. Commonly, a malicious actor, or their code, appears benign to a computer security system for a long enough time to exploit information or degrade the attacked system. The Trojan horse is among the most prevalent categories of a malicious attack on computer systems. It is characterized as a code that appears to provide a useful service but instead steals information or degrades the targeted system. A Trojan horse containing self replicating code is known as a virus. A trapdoor is a malicious attack in which an actor covertly modifies a system in such a way that they are permitted undetected access. Finally, a time bomb is an attack that accesses a system covertly and lies dormant until a detonation time. Upon detonation, the time bomb will inflict damage to the system either by disrupting service or destroying information.

Intrusion detection systems are a layer of network security that tracks activity patterns in a computer system to detect malicious actors before they can inflict harm. Debar et al. [186] describes efforts as early as 1981 and Sobirey [195] maintains a repository of prominent IDS projects. According to Debar et al. [186], the success of these systems has spawned a commercial market of IDS software including brands such as Sysco Systems, Haystack Labs, Secure Networks, among others. Typically, the IDS employs a detector module that monitors system status. The detector catalogues patterns of both normal and malicious activity in a database. The detector also monitors patterns in the current system configuration. Further, the detector provides an audit of events occurring within the system. The detector leverages these data channels to generate an alarm for suspicious activity and countermeasures if necessary. An IDS is evaluated by its *accuracy* of attack detection (false positive), *completeness* to detect all threats (false negatives) and *performance* to detect threats quickly.

NSL-KDD is a publicly available benchmark data set of network activity. NSL-KDD improves on several flaws of the well-known KDD Cup ‘99 benchmark data set. Most notably, NSL-KDD has rectified the 78% and 75% duplicate records in training and test sets, respectively [193]. Four classes of attacks are recorded in the data set. *Denial of service* attacks bombard a network with an overwhelming quantity of data such that the computing resources are exhausted. As a result, the system cannot fulfil any legitimate computing processes. *User to Root* attackers enter the network disguised as a legitimate user but seek security vulnerabilities which grant them elevated system privileges. A *remote to user* attack is performed by sending data to a private network and identifying insecure access points for exploitation. *Probing* is the attack technique by which the assailant studies an accessible system for vulnerabilities which will be exploited at a later time [196].

Maxwell et al. [39] and Viegas [197] describe IDS tools that incorporate ML models. Unfortunately, raw network traffic data is not a suitable input for building accurate and efficient ML models. Instead, the data must be transformed as a set of vectors representing the raw data. The process of constructing such vectors is known as feature engineering, which is a non-trivial task that requires both domain knowledge and mastery of ML to capture all available information in the model. It is shown experimentally that varying the feature engineering strategy does affect classification accuracy of the IDS but no single feature is known to be superior to others.

Kasongo [185] explores IDS procedures catered for the wireless domain. The UNSW-NB15 data set was selected to derive both the training and test data sets. A wrapper-based feature extractor generated many feature vectors for comparison from a full set of 42 features. The experiment was performed for both binary and multi-class classification in which the type of attack was predicted. Candidate algorithms included decision Trees, Random Forest, Naïve Bayes, K-Nearest Neighbor, Support Vector Machines, and Feed-forward Artificial Neural Networks (FFANN). The optimal feature set consisted of 26 columns. The FFANN reflected the best classification accuracy on the full data set with 87.10% binary and 77.16% multi-class. Random Forest, Decision Tree, and Support Vector Machine were close behind. When the feature set and neural network hyper-parameters were optimized, the classification accuracy of the FFANN improved to 99.66% and 99.77% for binary and multi-class classification, respectively.

Algorithm Selection Problem.

Rice's algorithm selection framework was presented in 1976 [198]. The framework is performed by employing all algorithms under consideration on all problems in a problem set. One or more performance metrics are chosen, and the performance of each algorithm on each problem is reported. Upon completion of the process, the preferred algorithm for each problem is taken as the one with the best performance metrics [198]. Woods [5] presents a modern depiction of Rice's framework as phase 1 in Figure 30.

The classic approach of learning algorithms is known as base learning. That is an ML algorithm which builds a data-driven model for a specific application [192]. Meta-learning, however, is an approach introduced by [199] which algorithms learn on the learning process itself. A meta-learning algorithm extracts meta-features $f(x) \in$ space F from a problem $x \in$ problem space P. The meta-model is trained to recommend the best-known base learning algorithm $a \in A$ to solve x. Works such as [200] and [201] further contribute to the theory of meta-learning recommendation systems [192].

In 2014, Smith [202] proposes the concept of applying meta-learning to Rice's model. It was not until 2016, however, that [192] implemented the concept. Figure

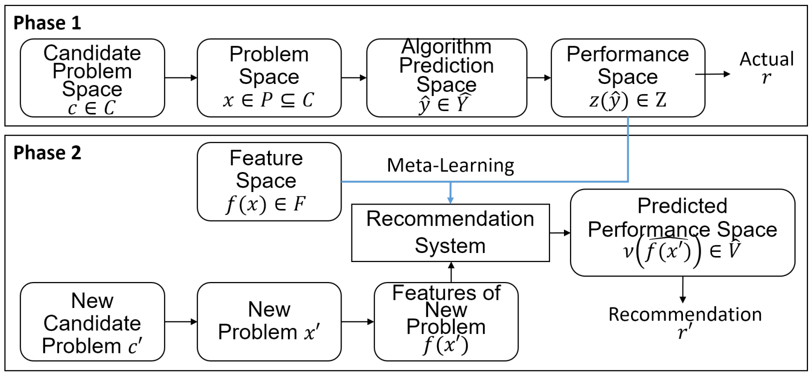

Phase 1
Candidate Problem Space $c \in C$
Problem Space $x \in P \subseteq C$
Algorithm Prediction Space $\hat{y} \in \hat{Y}$
Performance Space $z(\hat{y}) \in Z$
Actual r
Phase 2
Feature Space $f(x) \in F$
Meta-Learning
Recommendation System
Predicted Performance Space $v(f(x')) \in V$
New Candidate Problem c'
New Problem x'
Features of New Problem $f(x')$
Recommendation r'

Figure 30. The meta-learner version of Rice's framework [5].

30 demonstrates that Cui et al. [192] trained a meta-learning model to correlate problem features to algorithm performance and that the trained model could be used to recommend the algorithm for unobserved problems within Rice's framework. The meta-learner correctly recommended the best algorithm in 91% of test problems. Further, it demonstrated that time to perform algorithm selection could be reduced from minutes to seconds compared to trial-and-error techniques [192]. Follow on studies by [5] and [203] expanded on this work by exploring various meta-features and meta-learner response metrics.

A.3 Methodology

The assigned task for an IDS is to classify network traffic records as normal or malicious. This task is investigated from the broader perch of the algorithm selection problem. Figure 31 shows that within the analysis process, three factors drive the analytical approach and analytical technique selection. They are the input to the algorithm selection framework.

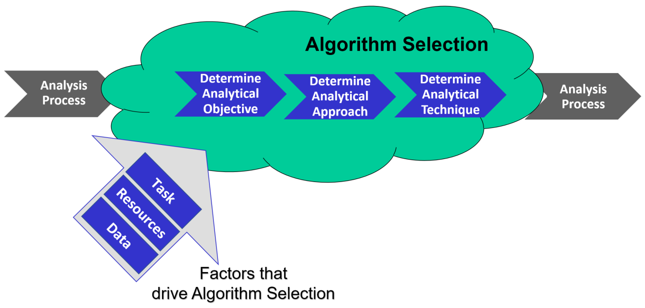

Algorithm Selection
Analysis Process
Determine Analytical Objective
Determine Analytical Approach
Determine Analytical Technique
Analysis Process
Task
Resources
Data
Factors that drive Algorithm Selection

Figure 31. The factors identified are superimposed with the stages of algorithm selection which they impact.

Characterizing the Problem.

The framework is a mechanism to characterize an analysis problem and to determine the algorithms that best matches the problem characterization. The three factors each drive analytical approach selection and analytical technique selection. The factor *assigned task* pertains to the problem provided by a decision-maker. The analyst must decipher the intent of the assignment from the lexicon of the decision-maker into specific analytical terms, which are listed under the *Task*. This list of terms, called *considerations* is shown in Figure 32 for each factor. The considerations for the factor *data* describe the different formats analysts commonly receive data for analysis problems. The *data* factor is important because it relates to the problem's compatibility with the mathematical mechanics of the analysis technique. Likewise, the considerations for the *resources* factor help the analyst identify which algorithms are compatible with the available resources. The analyst should refer to Figure 32 to evaluate and record the considerations for each factor prior to beginning step 1.

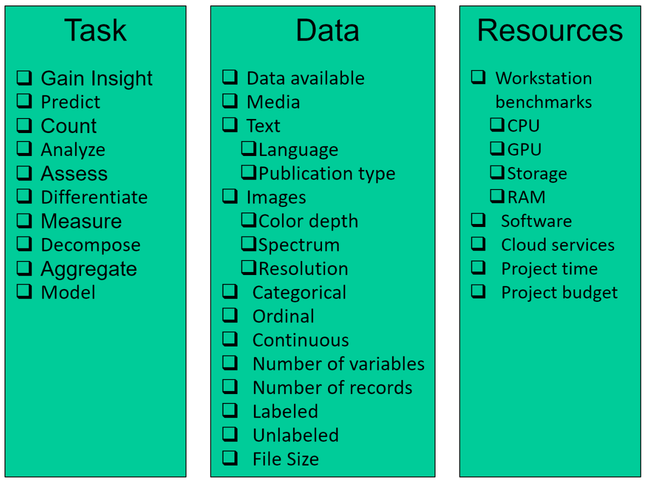


Figure 32. The considerations are shown for each factor which drives analytical approach and analytical technique selection.

Step 1: Map Problem to Category and Approach.

Step 1 leverages information from the *problem characterization* to identify the appropriate *analytical approaches*. Each *consideration* selected from the *assigned task* factor maps to one or more *categories of analysis*. The *categories of analysis* describe the general goal of the analysis problem [204]. Each *category of analysis* can be implemented by certain *analytical approaches*. The *analytical approach* a technique class referring to the specific type of response the techniques produce. Therefore, the framework leverages a hierarchical taxonomy that groups techniques grouped by both categories of analysis and analytical approaches. Figure 33 shows the mapping from *assigned task* to *category of analysis*, and the mapping of *category of analysis* to *analytical approach*.

Since the task of an IDS is to *classify* network users, the *prescriptive* and *predictive* categories of analysis are selected.

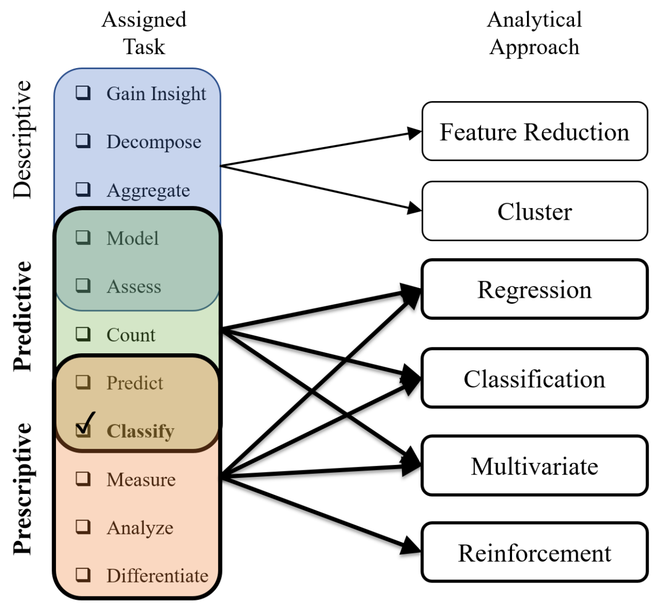


Figure 33. The assigned task for an IDS is classify. Classify is one of 11 common assigned tasks. It belongs to the predictive and prescriptive categories of analysis.

An excerpt of the proposed taxonomy is presented in Figure 34. The taxonomy is built with an object-oriented structure to promote flexibility and expandability. As an example, techniques are shown within the *regression* and *classification analytical approaches*. The text *predictive* and *descriptive* appears at the bottom edge of the regression panel to indicate that regression techniques produce results suitable for either of these two *categories of analysis*. The requirements, or required considerations, for each factor are presented with the technique. Compatible training styles are listed to the right of the technique name. The object-oriented structure allows new techniques to be easily added and new attributes to be included.

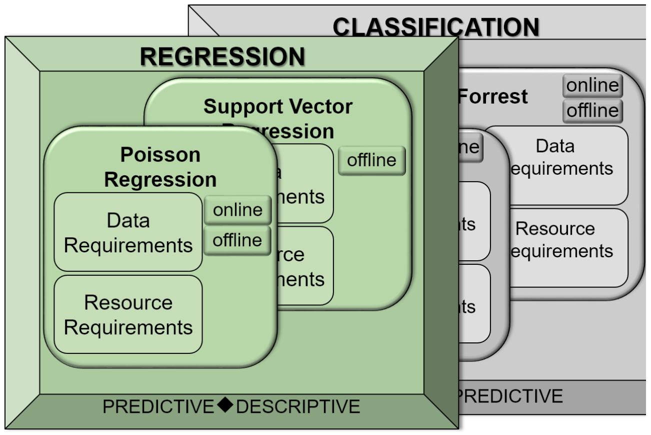


Figure 34. A portion of the proposed taxonomy is high-lighted to show its structure.

Step 2: Rank Techniques.

The framework identifies a subset of techniques that are compatible for the problem according to application. Next, the framework leverages the remaining three factors *data, resources* and *experience* to discern aspects of technique compatibility relating to the mechanics of the mathematical model. Step two predicts the utility scores of each algorithm from these factors according to two strategies: *rules of thumb* and *meta-learning*. They are presented in parallel below.

Rules-of-Thumb.

A logical decision tree is used to assign a preference rank among candidate algorithms. The decision tree is built according to rules-of-thumb regarding features of the data. The features pertaining to data also impact the compatibility of techniques in respect to resources. Thus, it is justified to use the same decision tree, Figure 35, to adjudicate the scores for both factors.

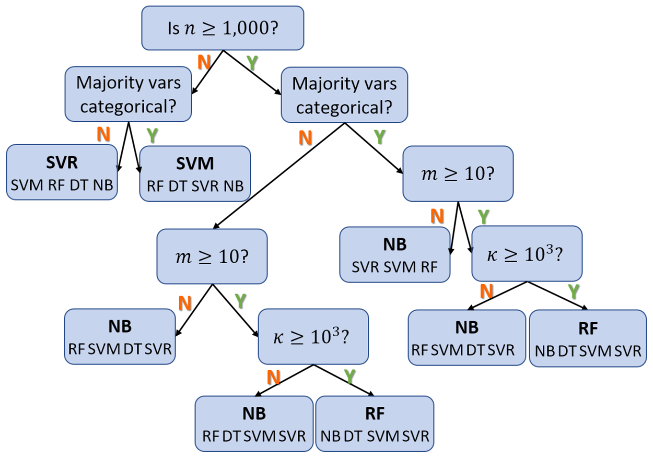


Figure 35. The decision tree represents the logical tests used to rank the recommendations via the rules of thumb strategy

.

Meta-Learning.

A meta-learner is constructed in Python 3.7. All data sets are pre-processed according to best practices for data mining. Base learning is performed on 14 benchmark data sets with 20 repetitions. For each repetition, the data sets were split into training and test sets with an 80/20 ratio and with stratification. The KDDTest+ and KDDTrain+ sets were obtained having already been split into a master test set and a training set. 12 meta-features of each data set were stored as predictor data for the meta-learner; the mean observed recall was stored as the target. The meta-learner was trained to model recall as a function of meta-features using a support vector regression algorithm. The radial basis function kernel was selected. The regularization parameter was set at 1.0, and the kernel coefficient was auto-scaled as a function of the number of features and predictor variance. All other parameters followed Scikit-learn defaults [125]. Pseudo-code of the meta-learner is presented in

Algorithm 1.

Algorithm 4: Pseudo-code of Meta-learner

```
input  : Repository of Datasets, Candidate Algorithms
output: Predicted Recall & Observed Recall of Test Dataset using Candidate
        Algorithms
Pre-processing
for all Dataset in Repository do
    for all column in Dataset do
        if column is numerical then
        miniMax(0,1)
        PCA()
        miniMax(0,1);
        else column is categorical;
        OneHot()
Feature Extraction
for all Dataset in Repository do
    featureExtract()
Base Learning
for all Algorithm do
    for all Dataset in Repository do
        trainBaseLearner()
        TestBaseLearner()
        RecordRecall()
        RankAlgorithms()
Meta Learning
for all Algorithm do
    TrainMetaLearner()
    PredictRecall()
    RecordRecall()
    RankAlgorithms()
```

The candidate algorithms are selected because they are members analytical approaches derived in step 1 of Section 3.2. Clearly, these are not the only algorithms that fall into the applicable analytical approach. Rather, they represent a demonstrative taxonomy. Note that the Scikit-learn default settings are selected on each algorithm to show generality of the algorithm selection framework.

Training data sets are selected from easily accessible benchmark repositories. The first five are selected to provide diversity of meta-features to the meta-learner and improve the robustness of the model. The nine subsets of KDDTrain+ are selected to provide statistical information consistent with the KDDTest+ data set. KDDTrain+ is split into subset to promote diversity of meta-features but also reduce the number of records in the training set which is an order of magnitude greater than the number of records in the test set. A uniform random number generator is used to determine the number of rows allocated to each training set. Rows are not re-arranged during the subset process.

1. Training Data

   (a) Heart: Predict presence of heart disease from 13 predictor variables [205]

   (b) Framingham: Predict presence of heart disease in the Framingham study from 15 predictor variables [206]

   (c) Spam: Predict if an email is spam based on six predictor variables [207]

   (d) Loan: Predict whether a consumer purchases a loan from Thera Bank based on 12 predictor variables [208]

   (e) Cancer: Predict whether a patient has breast cancer from 30 predictor variables collected in a fine needle aspirate procedure [209]

   (f) Nine subsets of KDDTrain+: Predict whether a network activity record is normal or malicious from four categorical and 39 numerical predictor variables [210]

2. Test Data

   (a) KDDTest+: Predict whether a network activity record is normal or malicious from four categorical and 39 numerical predictor variables [210]

The choice of meta-features was adopted from [203]. The following meta-features were used as predictor data by the meta-learner to model expected recall.

- Number of Rows
- Number of Columns
- Rows to Columns Ratio
- Number of Discrete Columns
- Maximum number of factors among discrete columns
- Minimum number of factors among discrete columns
- Average number of factors among discrete columns
- Number of continuous columns
- Gradient average
- Gradient minimum
- Gradient maximum
- Gradient standard deviation

A.4 Results

The algorithm selection framework is applied to the task of classifying network traffic as malicious or normal. Problem characterization is performed in step 1 to identify *prescriptive* and *predictive* as the categories of analysis. This leads to four analytical approaches, namely regression, classification, multivariate, and reinforcement. Five example algorithms which meet this criteria are taken from a notional taxonomy.

In step 2, candidate algorithms are ranked in order of preference by each recommendation strategy, rules-of-thumb and meta-learner. Both strategies yielded support vector regression as the most highly recommended algorithm. According to the mean observed recall, random forest was the best performing algorithm to detect malicious activity from the KDDTest+ data set. The standard deviations of observed recall on each algorithm were very low. The 90% Bonferroni confidence interval for support vector machine (SVM) and support vector regression (SVR) overlapped, indicating statistically identical recall performance. All other mean values were statistically unique. Further, the Spearman's coefficient of rank correlation was not statistically significant, largely due to the small size of the rank scheme. Since neither recommendation strategy succeeded in predicting the best performing algorithm, recall efficiency is introduced. Recall efficiency, Equation 65, is the ratio of the recall observed by the top recommended algorithm to best observed recall.

$$E_R = \frac{R_{bestRec}}{R_{bestObs}} \tag{65}$$

The recall efficiency for SVR, the top recommendation of both strategies, is 0.98. Table 17 presents a summary of the results including observed mean recall, meta-learner predicted recall, mean runtime, standard deviation of observed recall, observed ranks, rule-of-thumb predicted ranks, and meta-learner predicted ranks. Figure 36 outlines the mean observed recall and the recall predicted by the meta-learner for each algorithm.

It is difficult to ascertain whether either of the recommendation strategies employed in this study were successful. While the recall efficiency is very high, the rank correlation was not conclusive. All base learners produced very high and very similar recalls. It would therefore be difficult for any model to discern the true rank preference. The meta-model employed the meta-features according to the precedent set by

Table 17. Results compare the recommendations of each strategy to observed algorithm performance.

	Observed Mean Recall	Meta-Learner Predicted Recall	Mean Runtime (s)	SD of Observed Recall	Observed Ranks	Rules-of-Thumb Predicted Ranks	Meta-Learner Predicted Ranks
Decision Tree	0.975340865	0.891227551	0.188214	0.004141	2	4	5
Random Forest	0.982216595	0.935765698	1.593553	0.003063	**1**	3	3
Naive Bayes	0.863400857	0.952576618	0.007137	0.007961	5	2	2
SVM	0.959329957	0.925262066	8.602973	0.003163	4	5	4
SVR	0.962446436	0.96525973	7.647529	0.003088	3	**1**	**1**

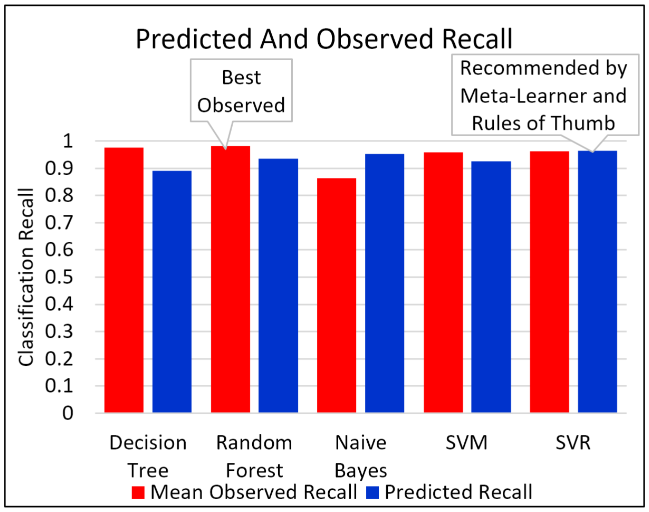


Figure 36. The predicted recall and observed mean recall are compared for each algorithm.

[203], however there were many more proposed by [192] that were not used. Furthermore, the meta-learner was trained by only 14 data sets, significantly less than used in [5]. Providing more training sets, especially from the domain of network traffic, would likely improve the predictive capability of the meta-learner.

As a whole, the framework is beneficial even when it does not recommend the true best performer. The framework consistently filters techniques that are incompatible with the problem characterization. Further, the framework identifies five viable options, each of which perform excellently.

A.5 Conclusion

Cyber attack detection from an IDS using any of the recommended algorithms could reasonably be deemed successful. Neither of the two recommendation strategies demonstrated perfect results. They did, however, show enough promise to motivate further investigations. Fundamentally, the meta-data and user input collected by the framework does contain information capable of consistently predicting a good analysis technique for a problem. Notably, there were algorithms from distinct analytical approaches that performed well on the same task. The process of problem characterization fits well into the framework but does require further refining. The rule-of-thumb decision tree provided intelligible recommendation logic whereas the meta-learner is a black box model. Future work should use the Gini criterion to optimize the decision tree. Further, the meta-learner should be improved to include more meta-features and training sets. There is a close connectedness in having a useful taxonomy of algorithms and a successful algorithm selection. This relationship is only beginning to be understood. Wireless IDS already provide good classification performance, however, algorithm selection, hyper-parameter tuning and feature engineering suffer from the time-costly trial-and-error practice. The algorithm selection framework may be a step forward in reducing this cost.

Appendix B. Referenced AFIT Theses

The following AFIT theses were referenced in this manuscript.

Table 18. AFIT Theses referenced in this manuscript

Thesis Title	**Author**	**Advisor**
"Cyber data anomaly detection using autoencoder neural networks" M.S. thesis, Air Force Institute of Technology, 2018.	Spencer Butt	Bradley Boehmke, PhD
"Meta Learning Recommendation System for Classification" M.S. Thesis, Air Force Institute of Technology, 2020	Clarence Williams	Jeff Weir, PhD
"A Metamodel Recommendation System Using Meta-Learning" M.S. thesis, Air Force Institute of Technology, 2020.	Megan Woods	Jeff Weir, PhD

Appendix C. Tabulated Performance of Models Trained With Synthetic Data for Contribution 1

Table 19. Classification recall of machine learning models trained with CTGAN synthetic data.

Classifier	Real Examples	Recall
RF	1,000	0.527
RF	5,000	0.501
RF	10,000	0.501
RF	15,000	0.500
RF	20,000	0.490
RF	25,000	0.499
RF	1,000	0.482
RF	5,000	0.516
RF	10,000	0.466
RF	15,000	0.498
RF	20,000	0.500
RF	25,000	0.500
SVM	1,000	0.500
SVM	5,000	0.538
SVM	10,000	0.619
SVM	15,000	0.699
SVM	20,000	0.661
SVM	25,000	0.476
SVM	1,000	0.308
SVM	5,000	0.509
SVM	10,000	0.528
SVM	15,000	0.658
SVM	20,000	0.446
SVM	25,000	0.682
MLP	1,000	0.668
MLP	5,000	0.489
MLP	10,000	0.500
MLP	15,000	0.498
MLP	20,000	0.528
MLP	25,000	0.488
MLP	1,000	0.442
MLP	5,000	0.480
MLP	10,000	0.595
MLP	15,000	0.489
MLP	20,000	0.507
MLP	25,000	0.500
LR	1,000	0.500
LR	5,000	0.500
LR	10,000	0.500
LR	15,000	0.501
LR	20,000	0.490
LR	25,000	0.500
LR	1,000	0.536
LR	5,000	0.500
LR	10,000	0.500
LR	15,000	0.500
LR	20,000	0.500
LR	25,000	0.500

Table 20. Classification recall of machine learning models trained with TVAE synthetic data.

Classifier	Real Examples	Recall
RF	1,000	0.504
RF	5,000	0.589
RF	10,000	0.568
RF	15,000	0.536
RF	20,000	0.623
RF	25,000	0.602
RF	1,000	0.510
RF	5,000	0.600
RF	10,000	0.660
RF	15,000	0.550
RF	20,000	0.560
RF	25,000	0.590
SVM	1,000	0.453
SVM	5,000	0.661
SVM	10,000	0.529
SVM	15,000	0.510
SVM	20,000	0.669
SVM	25,000	0.462
SVM	1,000	0.420
SVM	5,000	0.700
SVM	10,000	0.570
SVM	15,000	0.460
SVM	20,000	0.570
SVM	25,000	0.460
MLP	1,000	0.581
MLP	5,000	0.588
MLP	10,000	0.655
MLP	15,000	0.508
MLP	20,000	0.725
MLP	25,000	0.660
MLP	1,000	0.510
MLP	5,000	0.620
MLP	10,000	0.680
MLP	15,000	0.500
MLP	20,000	0.640
MLP	25,000	0.570
LR	1,000	0.535
LR	5,000	0.650
LR	10,000	0.620
LR	15,000	0.521
LR	20,000	0.683
LR	25,000	0.642
LR	1,000	0.540
LR	5,000	0.600
LR	10,000	0.660
LR	15,000	0.560
LR	20,000	0.590
LR	25,000	0.590

Table 21. Classification recall of machine learning models trained with real data.

Classifier	Examples	Recall
RF	1000	0.903
RF	5000	0.727
RF	10000	0.943
RF	15000	0.950
RF	20000	0.952
RF	25000	0.955
RF	1000	0.898
RF	5000	0.933
RF	10000	0.943
RF	15000	0.948
RF	20000	0.952
RF	25000	0.955
SVM	1000	0.677
SVM	5000	0.744
SVM	10000	0.747
SVM	15000	0.703
SVM	20000	0.763
SVM	25000	0.759
SVM	1000	0.702
SVM	5000	0.713
SVM	10000	0.782
SVM	15000	0.683
SVM	20000	0.774
SVM	25000	0.759
MLP	1000	0.836
MLP	5000	0.897
MLP	10000	0.904
MLP	15000	0.914
MLP	20000	0.916
MLP	25000	0.919
MLP	1000	0.857
MLP	5000	0.878
MLP	10000	0.902
MLP	15000	0.907
MLP	20000	0.915
MLP	25000	0.924
LR	1000	0.754
LR	5000	0.764
LR	10000	0.764
LR	15000	0.763
LR	20000	0.765
LR	25000	0.764
LR	1000	0.757
LR	5000	0.764
LR	10000	0.766
LR	15000	0.763
LR	20000	0.764
LR	25000	0.764

Table 22. Inverted Kolmogorov Smirnov D-Statistic of data generated by TVAE and CTGAN, varying the quantity of real data to fit the generators.

Data Source	Real Examples	KS-D
CTGAN	1,000	0.854
CTGAN	5,000	0.862
CTGAN	10,000	0.852
CTGAN	15,000	0.858
CTGAN	20,000	0.866
CTGAN	25,000	0.851
CTGAN	1,000	0.840
CTGAN	5,000	0.903
CTGAN	10,000	0.861
CTGAN	15,000	0.820
CTGAN	20,000	0.870
CTGAN	25,000	0.847
TVAE	1,000	0.650
TVAE	5,000	0.690
TVAE	10,000	0.700
TVAE	15,000	0.670
TVAE	20,000	0.700
TVAE	25,000	0.700
TVAE	1,000	0.640
TVAE	5,000	0.690
TVAE	10,000	0.700
TVAE	15,000	0.710
TVAE	20,000	0.700
TVAE	25,000	0.720

Table 23. Classification recall of machine learning models trained with half CTGAN synthetic data and half real data.

Classifier	Real Examples	Recall
RF	1000	0.899
RF	5000	0.838
RF	10000	0.942
RF	15000	0.950
RF	20000	0.951
RF	25000	0.954
RF	1000	0.899
RF	5000	0.932
RF	10000	0.943
RF	15000	0.949
RF	20000	0.951
RF	25000	0.954
SVM	1000	0.671
SVM	5000	0.236
SVM	10000	0.758
SVM	15000	0.316
SVM	20000	0.626
SVM	25000	0.287
SVM	1000	0.759
SVM	5000	0.260
SVM	10000	0.698
SVM	15000	0.309
SVM	20000	0.626
SVM	25000	0.778
MLP	1000	0.822
MLP	5000	0.885
MLP	10000	0.895
MLP	15000	0.887
MLP	20000	0.883
MLP	25000	0.911
MLP	1000	0.819
MLP	5000	0.862
MLP	10000	0.871
MLP	15000	0.887
MLP	20000	0.883
MLP	25000	0.905
LR	1000	0.733
LR	5000	0.719
LR	10000	0.715
LR	15000	0.747
LR	20000	0.753
LR	25000	0.716
LR	1000	0.731
LR	5000	0.743
LR	10000	0.757
LR	15000	0.726
LR	20000	0.753
LR	25000	0.748

Table 24. Classification recall of machine learning models trained with half TVAE synthetic data and half real data.

Classifier	Real Examples	Recall
RF	1000	0.896
RF	5000	0.929
RF	10000	0.941
RF	15000	0.947
RF	20000	0.950
RF	25000	0.953
RF	1000	0.900
RF	5000	0.932
RF	10000	0.943
RF	15000	0.947
RF	20000	0.951
RF	25000	0.954
SVM	1000	0.663
SVM	5000	0.617
SVM	10000	0.636
SVM	15000	0.672
SVM	20000	0.471
SVM	25000	0.597
SVM	1000	0.759
SVM	5000	0.260
SVM	10000	0.698
SVM	15000	0.781
SVM	20000	0.601
SVM	25000	0.778
MLP	1000	0.825
MLP	5000	0.871
MLP	10000	0.894
MLP	15000	0.906
MLP	20000	0.876
MLP	25000	0.894
MLP	1000	0.819
MLP	5000	0.862
MLP	10000	0.871
MLP	15000	0.893
MLP	20000	0.885
MLP	25000	0.905
LR	1000	0.724
LR	5000	0.745
LR	10000	0.740
LR	15000	0.739
LR	20000	0.753
LR	25000	0.755
LR	1000	0.731
LR	5000	0.743
LR	10000	0.757
LR	15000	0.726
LR	20000	0.752
LR	25000	0.748

Table 25. **Classification performance of machine learning models trained with 25,000 examples of a mixture of real and CTGAN synthetic data.**

% Synthetic	Classifier	Recall	Precision	F-1	% Synthetic	Classifier	Recall	Precision	F-1
0	RF	0.86128	0.862948	0.86111	50	MLP	0.73104	0.788088	0.717199
0	SVM	0.60048	0.607121	0.593908	50	LR	0.75888	0.768602	0.756614
0	MLP	0.82784	0.847363	0.825338	55	RF	0.85216	0.854406	0.851912
0	LR	0.7736	0.782306	0.771787	55	SVM	0.6768	0.676803	0.676795
5	RF	0.85952	0.861292	0.859336	55	MLP	0.8064	0.809449	0.805898
5	SVM	0.7208	0.721647	0.720505	55	LR	0.75328	0.76554	0.750324
5	MLP	0.81296	0.84265	0.808752	60	RF	0.84336	0.845898	0.843056
5	LR	0.77504	0.784431	0.773113	60	SVM	0.52832	0.531413	0.515095
10	RF	0.85936	0.861315	0.859158	60	MLP	0.80256	0.838077	0.797156
10	SVM	0.4176	0.41758	0.417532	60	LR	0.75328	0.762296	0.751077
10	MLP	0.81584	0.818133	0.815488	65	RF	0.84608	0.847542	0.845906
10	LR	0.7752	0.784699	0.773254	65	SVM	0.3552	0.355123	0.355129
15	RF	0.85888	0.860369	0.858724	65	MLP	0.81328	0.829121	0.810955
15	SVM	0.47456	0.467763	0.443362	65	LR	0.74224	0.749416	0.740309
15	MLP	0.81824	0.823286	0.8175	70	RF	0.84512	0.847428	0.844847
15	LR	0.77488	0.784299	0.772945	70	SVM	0.33408	0.331876	0.331814
20	RF	0.85792	0.859317	0.857772	70	MLP	0.75856	0.779069	0.754131
20	SVM	0.66	0.660168	0.659931	70	LR	0.73568	0.743354	0.733510
20	MLP	0.80784	0.843743	0.802612	75	RF	0.83808	0.839316	0.837921
20	LR	0.77184	0.780521	0.770007	75	SVM	0.35248	0.350074	0.349962
25	RF	0.85312	0.855023	0.85291	75	MLP	0.79744	0.824777	0.793012
25	SVM	0.72544	0.732993	0.723121	75	LR	0.7248	0.726841	0.724139
25	MLP	0.80976	0.829651	0.806789	80	RF	0.83664	0.839476	0.836280
25	LR	0.76992	0.779066	0.767963	80	SVM	0.43168	0.418719	0.407454
30	RF	0.8568	0.858879	0.85658	80	MLP	0.796	0.813252	0.793091
30	SVM	0.56704	0.579599	0.549896	80	LR	0.71584	0.717393	0.715293
30	MLP	0.80928	0.821634	0.807384	85	RF	0.82032	0.822274	0.820030
30	LR	0.77136	0.780861	0.769353	85	SVM	0.33184	0.330776	0.330735
35	RF	0.85664	0.857975	0.856496	85	MLP	0.78224	0.799548	0.778980
35	SVM	0.70832	0.720044	0.704274	85	LR	0.68208	0.683108	0.681675
35	MLP	0.8208	0.837681	0.818484	90	RF	0.8192	0.826797	0.818109
35	LR	0.76768	0.778346	0.765371	90	SVM	0.29808	0.271769	0.277097
40	RF	0.8472	0.848567	0.847039	90	MLP	0.7712	0.780189	0.769294
40	SVM	0.6464	0.649686	0.644337	90	LR	0.61408	0.62956	0.602521
40	MLP	0.79584	0.798304	0.795394	95	RF	0.79424	0.794937	0.794105
40	LR	0.76432	0.773371	0.762294	95	SVM	0.4032	0.391691	0.386545
45	RF	0.85472	0.857132	0.85446	95	MLP	0.65696	0.723911	0.629569
45	SVM	0.66128	0.663924	0.659823	95	LR	0.5208	0.584173	0.411661
45	MLP	0.79264	0.794205	0.792344	100	RF	0.49904	0.49952	0.333401
45	LR	0.76096	0.768794	0.759149	100	SVM	0.38544	0.284702	0.303624
50	RF	0.85232	0.854372	0.852093	100	MLP	0.49904	0.249041	0.332267
50	SVM	0.6464	0.646479	0.646333	100	LR	0.49904	0.249041	0.332267

Table 26. Classification performance of machine learning models trained with 100,000 examples of a mixture of real and CTGAN synthetic data.

% Synthetic	Classifier	Recall	Precision	F-1	% Synthetic	Classifier	Recall	Precision	F-1
0	RF	0.87622	0.877603	0.876107	50	MLP	0.82362	0.833719	0.822275
0	SVM	0.74972	0.750179	0.749605	50	LR	0.75606	0.768738	0.753149
0	MLP	0.83769	0.851893	0.836036	55	RF	0.86748	0.869031	0.867341
0	LR	0.77166	0.780512	0.769844	55	SVM	0.70201	0.715351	0.697322
5	RF	0.87592	0.877246	0.875811	55	MLP	0.79632	0.797094	0.796187
5	SVM	0.68593	0.686088	0.685863	55	LR	0.75231	0.763855	0.749571
5	MLP	0.83238	0.848432	0.830427	60	RF	0.86531	0.866984	0.865156
5	LR	0.77242	0.781517	0.770567	60	SVM	0.49758	0.497363	0.487017
10	RF	0.87511	0.876388	0.875004	60	MLP	0.82044	0.832144	0.818844
10	SVM	0.60302	0.60407	0.602016	60	LR	0.74982	0.760522	0.747224
10	MLP	0.83258	0.848945	0.830594	65	RF	0.8622	0.863822	0.862046
10	LR	0.77343	0.783161	0.771467	65	SVM	0.3535	0.347416	0.346991
15	RF	0.87474	0.875979	0.874637	65	MLP	0.80537	0.815249	0.803833
15	SVM	0.43666	0.434434	0.431839	65	LR	0.74361	0.753287	0.741138
15	MLP	0.8305	0.832302	0.83027	70	RF	0.86295	0.864619	0.862793
15	LR	0.77212	0.781755	0.770155	70	SVM	0.31978	0.315575	0.315880
20	RF	0.87384	0.875229	0.873723	70	MLP	0.81272	0.814728	0.812421
20	SVM	0.70685	0.711425	0.705256	70	LR	0.739	0.746586	0.736977
20	MLP	0.83298	0.847695	0.831194	75	RF	0.85967	0.861186	0.859523
20	LR	0.77163	0.78165	0.769581	75	SVM	0.33106	0.323257	0.323594
25	RF	0.87333	0.874788	0.873207	75	MLP	0.75282	0.781654	0.746328
25	SVM	0.49675	0.496366	0.483099	75	LR	0.72805	0.732165	0.726839
25	MLP	0.80812	0.814457	0.807148	80	RF	0.85684	0.858289	0.856695
25	LR	0.76935	0.779151	0.767307	80	SVM	0.33003	0.329746	0.329750
30	RF	0.8724	0.873676	0.872291	80	MLP	0.79727	0.80134	0.796583
30	SVM	0.50405	0.504629	0.488049	80	LR	0.71054	0.710822	0.710443
30	MLP	0.82805	0.833855	0.827299	85	RF	0.85241	0.85494	0.852147
30	LR	0.76734	0.776912	0.765312	85	SVM	0.31688	0.314417	0.314606
35	RF	0.87012	0.871466	0.870002	85	MLP	0.79315	0.793931	0.793013
35	SVM	0.44779	0.447444	0.44688	85	LR	0.67588	0.678076	0.674878
35	MLP	0.8205	0.823549	0.820076	90	RF	0.84733	0.849405	0.847103
35	LR	0.76476	0.775101	0.762528	90	SVM	0.3396	0.309026	0.312066
40	RF	0.86908	0.870513	0.868953	90	MLP	0.78571	0.787191	0.785433
40	SVM	0.71473	0.739203	0.707242	90	LR	0.60593	0.630847	0.586232
40	MLP	0.80572	0.808776	0.805238	95	RF	0.83079	0.835592	0.830183
40	LR	0.76278	0.773602	0.760411	95	SVM	0.32927	0.329267	0.329267
45	RF	0.86828	0.869703	0.868153	95	MLP	0.73751	0.750116	0.734160
45	SVM	0.70358	0.710824	0.701012	95	LR	0.53053	0.600623	0.431533
45	MLP	0.81941	0.834779	0.817313	100	RF	0.50062	0.553238	0.336735
45	LR	0.76107	0.771685	0.758713	100	SVM	0.5218	0.522069	0.520336
50	RF	0.86705	0.868682	0.866903	100	MLP	0.49962	0.301988	0.333253
50	SVM	0.6182	0.622815	0.61458	100	LR	0.5	0.25	0.333333

Table 27. Classification performance of machine learning models trained with 25,000 examples of a mixture of real and TVAE synthetic data.

% Synthetic	Classifier	Recall	Precision	F-1	% Synthetic	Classifier	Recall	Precision	F-1
0	RF	0.8608	0.862482	0.860627	50	MLP	0.81712	0.83306	0.814857
0	SVM	0.60048	0.607121	0.593908	50	LR	0.75488	0.758731	0.753922
0	MLP	0.82288	0.829764	0.82192	55	RF	0.85072	0.852338	0.850537
0	LR	0.7736	0.782306	0.771787	55	SVM	0.492	0.491904	0.487725
5	RF	0.85936	0.860836	0.859206	55	MLP	0.8056	0.807223	0.805325
5	SVM	0.72032	0.722429	0.719612	55	LR	0.7536	0.75739	0.752647
5	MLP	0.81696	0.840694	0.813657	60	RF	0.84576	0.847721	0.845529
5	LR	0.77344	0.782173	0.771619	60	SVM	0.46592	0.461924	0.450414
10	RF	0.85888	0.860399	0.858721	60	MLP	0.756	0.774111	0.751988
10	SVM	0.6104	0.617288	0.604347	60	LR	0.74672	0.748233	0.746304
10	MLP	0.7936	0.80106	0.792355	65	RF	0.84512	0.846105	0.845
10	LR	0.7728	0.780137	0.771253	65	SVM	0.33792	0.33554	0.335592
15	RF	0.85936	0.860501	0.859239	65	MLP	0.80496	0.805583	0.804849
15	SVM	0.6768	0.6789	0.675783	65	LR	0.73968	0.740765	0.739360
15	MLP	0.82672	0.835219	0.825581	70	RF	0.84208	0.843905	0.841857
15	LR	0.7744	0.782023	0.772815	70	SVM	0.36032	0.355999	0.355616
20	RF	0.85456	0.855711	0.854433	70	MLP	0.79424	0.806064	0.792182
20	SVM	0.42528	0.413693	0.404784	70	LR	0.74016	0.740519	0.740048
20	MLP	0.80768	0.844446	0.802331	75	RF	0.83744	0.838225	0.837336
20	LR	0.77136	0.778162	0.769905	75	SVM	0.44784	0.443121	0.436747
25	RF	0.85648	0.858453	0.85627	75	MLP	0.79072	0.790746	0.790712
25	SVM	0.56752	0.567601	0.56744	75	LR	0.72816	0.728179	0.728158
25	MLP	0.82064	0.835123	0.818636	80	RF	0.8312	0.833325	0.830914
25	LR	0.7688	0.77565	0.767305	80	SVM	0.45584	0.454276	0.450625
30	RF	0.85872	0.860376	0.858546	80	MLP	0.78432	0.787886	0.783618
30	SVM	0.48912	0.48806	0.473907	80	LR	0.72352	0.723704	0.723475
30	MLP	0.82432	0.836335	0.822697	85	RF	0.82096	0.822143	0.820782
30	LR	0.76736	0.77392	0.765909	85	SVM	0.32896	0.32528	0.325495
35	RF	0.85744	0.858654	0.85731	85	MLP	0.77872	0.778777	0.778704
35	SVM	0.6888	0.696627	0.685778	85	LR	0.71472	0.71695	0.714030
35	MLP	0.81264	0.83328	0.809636	90	RF	0.8168	0.821094	0.816159
35	LR	0.76528	0.77187	0.763798	90	SVM	0.58496	0.596478	0.571738
40	RF	0.848	0.848969	0.847885	90	MLP	0.76144	0.761799	0.761370
40	SVM	0.54944	0.55096	0.546444	90	LR	0.68288	0.693514	0.678592
40	MLP	0.7792	0.787409	0.777661	95	RF	0.7912	0.791668	0.791127
40	LR	0.76128	0.765575	0.760268	95	SVM	0.44768	0.447606	0.447550
45	RF	0.852	0.853913	0.851787	95	MLP	0.66896	0.705839	0.653682
45	SVM	0.57296	0.574488	0.570521	95	LR	0.6304	0.681694	0.602726
45	MLP	0.80912	0.818758	0.807625	100	RF	0.51584	0.721145	0.370566
45	LR	0.7568	0.76089	0.7558	100	SVM	0.6464	0.649265	0.644796
50	RF	0.85552	0.856864	0.855374	100	MLP	0.55968	0.71003	0.464502
50	SVM	0.54864	0.551933	0.540752	100	LR	0.5344	0.676049	0.418238

Table 28. Classification performance of machine learning models trained with 100,000 examples of a mixture of real and TVAE synthetic data.

% Synthetic	Classifier	Recall	Precision	F-1	% Synthetic	Classifier	Recall	Precision	F-1
0	RF	0.87591	0.877315	0.875794	50	MLP	0.8254	0.827653	0.825099
0	SVM	0.74972	0.750179	0.749605	50	LR	0.757	0.76242	0.755739
0	MLP	0.83173	0.832603	0.83162	55	RF	0.86753	0.868997	0.867398
0	LR	0.77166	0.780512	0.769844	55	SVM	0.65011	0.652111	0.648955
5	RF	0.87624	0.877588	0.876129	55	MLP	0.82245	0.834392	0.82085
5	SVM	0.6767	0.679434	0.675464	55	LR	0.75234	0.755808	0.751498
5	MLP	0.83796	0.838393	0.837908	60	RF	0.86583	0.867307	0.865695
5	LR	0.77168	0.78013	0.769945	60	SVM	0.3571	0.356919	0.356897
10	RF	0.87569	0.876921	0.875588	60	MLP	0.81962	0.819842	0.819589
10	SVM	0.72021	0.720278	0.720188	60	LR	0.74907	0.751466	0.748471
10	MLP	0.83608	0.843322	0.835211	65	RF	0.86276	0.864202	0.862624
10	LR	0.77087	0.778666	0.769256	65	SVM	0.63858	0.641806	0.636512
15	RF	0.8748	0.875917	0.874707	65	MLP	0.81363	0.816683	0.813180
15	SVM	0.67152	0.672277	0.671159	65	LR	0.74422	0.745579	0.743866
15	MLP	0.83806	0.842775	0.837501	70	RF	0.86186	0.86339	0.861714
15	LR	0.77122	0.778762	0.769662	70	SVM	0.60751	0.612717	0.602924
20	RF	0.87385	0.875101	0.873745	70	MLP	0.81486	0.829806	0.812738
20	SVM	0.684	0.684435	0.683814	70	LR	0.73863	0.739157	0.738486
20	MLP	0.83916	0.842129	0.83881	75	RF	0.85972	0.86103	0.859593
20	LR	0.77151	0.779099	0.769946	75	SVM	0.6834	0.685757	0.682393
25	RF	0.87315	0.874479	0.873037	75	MLP	0.81643	0.823988	0.815353
25	SVM	0.6459	0.646108	0.645774	75	LR	0.73059	0.730598	0.730588
25	MLP	0.82613	0.82613	0.82613	80	RF	0.85515	0.856177	0.855045
25	LR	0.77047	0.777684	0.76897	80	SVM	0.36897	0.368522	0.368432
30	RF	0.87252	0.873743	0.872416	80	MLP	0.80488	0.824369	0.801904
30	SVM	0.6509	0.65115	0.650756	80	LR	0.72086	0.721919	0.720527
30	MLP	0.82881	0.850089	0.826169	85	RF	0.85319	0.854954	0.853007
30	LR	0.76886	0.776213	0.767311	85	SVM	0.36626	0.365845	0.365769
35	RF	0.87002	0.871328	0.869905	85	MLP	0.77594	0.780054	0.775114
35	SVM	0.39139	0.390622	0.39032	85	LR	0.70517	0.709791	0.703537
35	MLP	0.83008	0.832795	0.829733	90	RF	0.84246	0.843811	0.842305
35	LR	0.76681	0.773516	0.765372	90	SVM	0.49334	0.492599	0.480334
40	RF	0.86943	0.870759	0.869313	90	MLP	0.76951	0.775411	0.768269
40	SVM	0.63677	0.641583	0.633656	90	LR	0.68507	0.70202	0.678323
40	MLP	0.82729	0.836042	0.826158	95	RF	0.82947	0.831371	0.829225
40	LR	0.76324	0.769293	0.761902	95	SVM	0.48383	0.481648	0.468017
45	RF	0.86826	0.869553	0.868145	95	MLP	0.71482	0.72963	0.710146
45	SVM	0.69244	0.702131	0.688709	95	LR	0.63285	0.691331	0.602474
45	MLP	0.82674	0.827924	0.826584	100	RF	0.51887	0.717038	0.376561
45	LR	0.7605	0.765972	0.759262	100	SVM	0.4821	0.480433	0.470826
50	RF	0.86697	0.868471	0.866834	100	MLP	0.52281	0.663836	0.391964
50	SVM	0.64127	0.641397	0.64119	100	LR	0.5241	0.67278	0.393658

Appendix D. Tabulated Performance of Adversarial Examples Constructed for Contribution 2

Table 29. Detailed performance of infiltration adversarial examples tested against the surrogate model and NIDS models.

	Infiltration														
	Raw Packet					Random Perturbations					Near Optimal Perturbations				
Trial	Fool Surrogate	Loss Surrogate	Fool CNN NIDS	Fool FNN NIDS	Fool Adaboost NIDS	Rate Fool Surrogate	Best Obs Loss Surrogate	Rate Fool CNN NIDS	Rate Fool FNN NIDS	Rate Fool Adaboost NIDS	Rate Fool Surrogate	Best Obs Loss Surrogate	Rate Fool CNN NIDS	Rate Fool FNN NIDS	Rate Fool Adaboost NIDS
1	0	2.18E-05	0	0	0	0	2.19E-02	0	0	0	1	1.31E+00	0	0	0
2	0	2.23E-08	0	0	0	0	2.29E-03	0.002	0.002	0	1	3.55E+00	0.120	0	0
3	0	5.40E-10	0	0	0	0	4.89E-03	0	0	0	1	8.48E-01	0	0	0
4	0	1.69E-10	0	0	0	0	1.04E-04	0	0	0	1	1.90E+00	0	0	0
5	0	3.69E-11	0	0	0	0	2.82E-04	0	0	0	1	7.28E+00	0	0	0
6	0	5.15E-13	0	0	0	0	5.89E-06	0	0	0	1	1.79E+00	0	0	0
7	0	5.26E-09	0	0	0	0	1.33E-03	0	0	0	1	5.26E+00	0.020	0	0
8	0	6.94E-10	0	0	0	0	2.05E-04	0	0	0	1	2.43E+00	0	0	0
9	0	4.52E-11	0	0	0	0	5.84E-04	0	0	0	1	1.92E+00	0	0	0
10	0	4.69E-09	0	0	0	0	9.08E-04	0	0	0	1	3.16E+00	0	0	0
11	0	6.74E-14	0	0	0	0	8.35E-06	0	0	0	1	1.31E+00	0	0	0
12	0	7.21E-08	0	0	0	0	1.10E-02	0	0	0	1	1.21E+01	0	0	0
13	0	2.23E-08	0	0	0	0	3.06E-03	0.002	0.002	0	1	6.88E+00	0.220	0	0
14	0	2.09E-10	0	0	0	0	4.15E-04	0	0	0	1	3.82E+00	0	0	0
15	0	2.43E-09	0	0	0	0	6.66E-03	0.006	0.006	0	1	7.97E+00	0.220	0	0
16	0	2.13E-12	0	0	0	0	1.62E-05	0	0	0	0	5.53E-01	0	0	0
17	0	1.54E-11	0	0	0	0	2.50E-04	0	0	0	0	2.38E-01	0	0	0
18	0	2.74E-11	0	0	0	0	1.62E-04	0	0	0	1	1.61E+00	0	0	0
19	0	1.14E-10	0	0	0	0	1.01E-04	0	0	0	1	6.70E+00	0	0	0
20	0	5.26E-09	0	0	0	0	1.19E-03	0	0	0	1	5.31E+00	0	0	0
21	0	1.73E-17	0	0	0	0	5.53E-09	0	0	0	0	6.61E-03	0	0	0
22	0	3.20E-12	0	0	0	0	6.51E-06	0	0	0	1	1.98E+00	0.020	0	0
23	0	1.22E-09	0	0	0	0	4.56E-04	0	0	0	1	2.59E+00	0	0	0
24	0	5.51E-12	0	0	0	0	3.48E-05	0	0	0	1	5.42E+00	0	0	0
25	0	1.25E-09	0	0	0	0	4.32E-03	0	0	0	1	6.58E+00	0	0	0
26	0	4.52E-11	0	0	0	0	1.32E-04	0	0	0	0	3.87E-01	0	0	0
27	0	1.35E-09	0	0	0	0	1.52E-02	0	0	0	1	7.00E+00	0	0	0
28	0	2.22E-10	0	0	0	0	1.32E-03	0	0	0	1	4.57E+00	0	0	0
29	0	3.79E-07	0	0	0	0	4.06E-02	0	0	0	1	8.86E+00	0	0	0
30	0	1.36E-17	0	0	0	0	3.95E-09	0	0	0	0	2.23E-03	0	0	0
31	0	2.09E-10	0	0	0	0	3.30E-04	0	0	0	1	2.15E+00	0	0	0
32	0	2.85E-11	0	0	0	0	1.41E-04	0	0	0	1	1.13E+00	0	0	0
33	0	9.68E-15	0	0	0	0	1.19E-07	0	0	0	0	4.48E-03	0	0	0
34	0	6.13E-12	0	0	0	0	5.10E-06	0	0	0	0	4.31E-01	0	0	0
35	0	1.69E-10	0	0	0	0	6.30E-05	0	0	0	1	2.11E+00	0	0	0
36	0	9.68E-15	0	0	0	0	2.74E-07	0	0	0	0	2.41E-01	0	0	0
37	0	2.09E-10	0	0	0	0	3.28E-04	0	0	0	1	3.24E+00	0	0	0
38	0	3.62E-15	0	0	0	0	4.15E-08	0	0	0	0	3.83E-05	0	0	0
39	0	4.78E-11	0	0	0	0	2.08E-04	0	0	0	1	9.00E-01	0	0	0
40	0	2.97E-12	0	0	0	0	5.46E-06	0	0	0	0	3.22E-01	0	0	0
41	0	3.76E-07	0	0	0	0	3.50E-01	0	0	0	1	1.59E+01	0	0	0
42	0	3.69E-11	0	0	0	0	5.63E-05	0	0	0	1	1.63E+00	0	0	0
43	0	3.49E-13	0	0	0	0	2.67E-05	0	0	0	1	1.12E+00	0	0	0
44	0	6.13E-12	0	0	0	0	1.35E-05	0	0	0	0	5.50E-02	0	0	0
45	0	2.05E-07	0	0	0	0	4.04E-03	0	0	0	1	2.14E+00	0	0	0
46	0	2.22E-10	0	0	0	0	3.39E-04	0	0	0	0	2.83E-01	0	0	0
47	0	1.52E-05	0	0	0	0.026	5.21E+00	0.618	0.618	0.532	1	1.65E+01	0.860	0.480	0.600
48	0	3.51E-11	0	0	0	0	4.21E-05	0	0	0	1	2.60E+00	0	0	0
49	0	5.88E-12	0	0	0	0	1.16E-05	0	0	0	1	3.51E+00	0	0	0
50	0	1.88E-10	0	0	0	0	2.67E-04	0	0	0	1	2.79E+00	0	0	0
51	0	3.20E-12	0	0	0	0	2.58E-04	0	0	0	1	3.37E+00	0	0	0
52	0	6.13E-12	0	0	0	0	3.24E-05	0	0	0	1	1.03E+00	0	0	0
53	0	9.21E-12	0	0	0	0	1.31E-04	0	0	0	1	4.29E+00	0	0	0
54	0	2.12E-15	0	0	0	0	7.20E-08	0	0	0	0	8.84E-03	0	0	0
55	0	8.19E-09	0	0	0	0	1.33E-01	0	0	0	1	7.15E+00	0	0	0
56	0	1.13E-10	0	0	0	0	9.38E-05	0	0	0	1	3.30E+00	0	0	0
57	0	1.71E-11	0	0	0	0	9.14E-04	0	0	0	1	3.64E+00	0	0	0
58	0	3.48E-11	0	0	0	0	1.52E-04	0	0	0	0	6.77E-01	0	0	0
59	0	4.78E-11	0	0	0	0	1.28E-03	0	0	0	1	2.01E+00	0	0	0
60	0	1.36E-11	0	0	0	0	5.01E-05	0	0	0	0	4.45E-01	0	0	0
61	0	5.38E-09	0	0	0	0	3.93E-01	0	0	0	1	8.26E+00	0	0	0
62	0	5.51E-12	0	0	0	0	3.49E-04	0	0	0	1	3.18E+00	0	0	0
63	0	2.22E-10	0	0	0	0	8.03E-04	0	0	0	1	1.01E+00	0	0	0
64	0	2.04E-10	0	0	0	0	2.32E-04	0	0	0	1	2.13E+00	0	0	0

65	0	1.31E-06	0	0	0	0.004	3.03E+00	0	0	0	1	9.80E+00	0	0	0
66	0	1.52E-05	0	0	0	0.024	2.92E+00	0.624	0.624	0.596	1	1.15E+01	0.900	0.520	0.600
67	0	5.51E-12	0	0	0	0	2.05E-04	0	0	0	1	2.45E+00	0	0	0
68	0	3.84E-09	0	0	0	0	1.50E-03	0	0	0	1	4.31E+00	0	0	0
69	0	3.49E-13	0	0	0	0	2.97E-06	0	0	0	1	8.45E-01	0	0	0
70	0	4.04E-11	0	0	0	0	1.20E-03	0	0	0	1	4.17E+00	0	0	0
71	0	6.13E-12	0	0	0	0	7.97E-06	0	0	0	0	4.95E-01	0	0	0
72	0	2.23E-08	0	0	0	0	2.50E-02	0	0	0	1	7.75E+00	0.040	0	0
73	0	6.27E-14	0	0	0	0	3.58E-05	0	0	0	1	3.71E+00	0	0	0
74	0	1.74E-15	0	0	0	0	1.20E-05	0	0	0	0	4.74E-01	0	0	0
75	0	9.35E-12	0	0	0	0	1.57E-04	0	0	0	1	3.47E+00	0	0	0
76	0	1.25E-09	0	0	0	0	8.09E-04	0	0	0	1	4.32E+00	0	0	0
77	0	2.85E-11	0	0	0	0	2.07E-04	0	0	0	1	1.76E+00	0	0	0
78	0	2.09E-10	0	0	0	0	7.88E-04	0	0	0	1	3.32E+00	0	0	0
79	0	5.93E-14	0	0	0	0	4.05E-06	0	0	0	0	2.51E-01	0	0	0
80	0	2.22E-10	0	0	0	0	4.04E-04	0	0	0	1	2.95E+00	0	0	0
81	0	5.26E-09	0	0	0	0	5.84E-04	0	0	0	1	4.35E+00	0	0	0
82	0	1.93E-10	0	0	0	0	4.12E-04	0	0	0	1	3.72E+00	0	0	0
83	1	3.88E+00	0	0	1	0.322	1.28E+01	0.010	0.010	0.354	1	3.15E+01	0.380	0	0.540
84	0	2.04E-10	0	0	0	0	1.16E-04	0	0	0	1	4.44E+00	0	0	0
85	0	5.26E-09	0	0	0	0	4.18E-03	0	0	0	1	1.75E+00	0	0	0
86	0	3.81E-11	0	0	0	0	1.87E-04	0	0	0	1	8.52E-01	0	0	0
87	0	5.93E-14	0	0	0	0	4.23E-07	0	0	0	0	6.85E-02	0	0	0
88	0	2.23E-08	0	0	0	0	1.11E-02	0	0	0	1	5.71E+00	0	0	0
89	0	2.85E-11	0	0	0	0	4.18E-04	0	0	0	1	7.96E-01	0	0	0
90	0	2.22E-10	0	0	0	0	3.86E-04	0	0	0	1	2.91E+00	0	0	0
91	0	3.48E-11	0	0	0	0	1.57E-04	0	0	0	1	4.04E+00	0	0	0
92	0	3.76E-07	0	0	0	0	6.39E-01	0	0	0	1	9.35E+00	0	0	0
93	0	1.93E-10	0	0	0	0	1.42E-03	0	0	0	1	3.03E+00	0	0	0
94	0	1.93E-10	0	0	0	0	1.58E-03	0	0	0	1	3.11E+00	0	0	0
95	0	3.69E-11	0	0	0	0	3.77E-04	0	0	0	1	1.27E+00	0	0	0
96	0	8.40E-13	0	0	0	0	2.26E-04	0	0	0	1	6.75E+00	0	0	0
97	0	1.35E-09	0	0	0	0	9.00E-03	0	0	0	1	8.84E+00	0.080	0	0
98	0	5.40E-10	0	0	0	0	9.81E-04	0	0	0	1	2.21E+00	0	0	0
99	0	3.33E-10	0	0	0	0	3.28E-04	0	0	0	1	2.65E+00	0.020	0	0
100	0	1.69E-10	0	0	0	0	9.83E-05	0	0	0	1	2.86E+00	0	0	0

% Packets Fool Model	**Avg Loss**	**% Packets Fool Model**	**% Packets Fool Model**	**% Packets Fool Model**	**% Packets Fool Model**	**Avg Loss**	**% Packets Fool Model**	**% Packets Fool Model**	**% Packets Fool Model**	**% Packets Fool Model**	**Avg Loss**	**% Packets Fool Model**	**% Packets Fool Model**	**% Packets Fool Model**
1%	3.88E-02	0	0	1%	4%	2.57E-01	6%	6%	3%	81%	3.72540116	11%	2%	3%

Table 30. Detailed performance of slowloris adversarial examples tested against the surrogate model and NIDS models.

	Slowloris														
	Raw Packet					Random Perturbations					Near Optimal Perturbations				
Trial	Fool Surrogate	Loss Surrogate	Fool CNN NIDS	Fool FNN NIDS	Fool Adaboost NIDS	Rate Fool Surrogate	Best Obs Loss Surrogate	Rate Fool CNN NIDS	Rate Fool FNN NIDS	Rate Fool Adaboost NIDS	Rate Fool Surrogate	Best Obs Loss Surrogate	Rate Fool CNN NIDS	Rate Fool FNN NIDS	Rate Fool Adaboost NIDS
1	0	1.91E-03	0	1	0	0.670	1.30E+01	0.036	0.036	0.020	1	2.58E+01	0.300	1	0.240
2	0	6.90E-02	0	1	0	0.638	1.22E+01	0.190	0.190	0.276	1	2.49E+01	0.820	1	0.520
3	1	3.59E+00	0	0	1	0.336	9.42E+00	0	0	0.498	1	2.38E+01	0	0	0.820
4	0	1.29E-05	0	0	0	0	4.39E-01	0	0	0	1	9.52E+00	0	0	0
5	0	2.41E-01	0	1	0	0.468	1.00E+01	0.002	0.002	0.052	1	2.30E+01	0	0.680	0
6	0	1.91E-03	0	1	0	0.670	1.31E+01	0.064	0.064	0.036	1	2.83E+01	0.480	1	0.120
7	0	8.33E-06	0	0	0	0	2.72E-01	0	0	0	1	1.02E+01	0	0	0
8	0	5.16E-06	0	0	0	0	2.43E-01	0	0	0	1	7.70E+00	0	0	0
9	0	8.33E-06	0	0	0	0	3.47E-01	0	0	0	1	1.25E+01	0.300	0	0
10	0	3.42E-06	0	0	0	0	5.04E-02	0	0	0	1	7.85E+00	0	0	0
11	0	6.90E-02	0	1	0	0.630	1.58E+01	0.196	0.196	0.294	1	2.50E+01	0.760	1	0.400
12	0	6.90E-02	0	1	0	0.592	1.09E+01	0.228	0.228	0.3	1	2.63E+01	0.740	0.980	0.560
13	0	7.78E-05	0	1	0	0.028	4.99E+00	0.008	0.008	0.218	1	1.28E+01	0.040	0.960	0.320
14	0	5.92E-06	0	0	0	0.002	1.38E+00	0	0	0	1	8.48E+00	0	0	0
15	0	7.78E-05	0	1	0	0.020	4.19E+00	0.004	0.004	0.234	1	1.25E+01	0.140	0.980	0.320
16	0	4.82E-05	0	0	0	0.008	2.50E+00	0.172	0.172	0.178	1	1.44E+01	0.660	0.700	0.240
17	0	1.71E-06	0	0	0	0	1.26E-02	0	0	0	1	5.82E+00	0	0	0
18	0	1.85E-04	0	1	1	0.090	5.95E+00	0	0	1	1	1.98E+01	0.040	1	1
19	0	2.42E-06	0	0	0	0	4.76E-02	0	0	0	1	8.10E+00	0.140	0	0
20	0	5.17E-01	0	1	1	0.222	7.91E+00	0.002	0.002	0.938	1	1.71E+01	0.120	1	0.980
21	0	1.66E-06	0	0	0	0	1.86E-02	0	0	0	1	3.94E+00	0	0	0
22	0	3.58E-04	0	0	0	0.018	2.95E+00	0.020	0.020	0.160	1	1.09E+01	0.100	0.020	0.220
23	0	6.96E-02	0	0	1	0.358	1.07E+01	0.144	0.144	0.356	1	1.84E+01	0.600	0.200	0.460
24	0	2.21E-06	0	0	0	0	4.73E-02	0	0	0	1	6.07E+00	0	0	0
25	1	4.94E+00	0	1	0	0.334	1.07E+01	0.226	0.226	0.202	1	2.31E+01	0.760	1	0.320
26	0	2.21E-06	0	0	0	0	1.25E-02	0	0	0	1	7.16E+00	0	0	0
27	0	3.58E-04	0	0	0	0.002	9.37E-01	0.014	0.014	0.158	1	1.52E+01	0.180	0.020	0
28	0	1.58E-06	0	0	0	0	3.16E-02	0	0	0	1	3.65E+00	0.020	0	0
29	0	6.96E-02	0	0	1	0.384	8.58E+00	0.176	0.176	0.384	1	2.09E+01	0.600	0.080	0.520
30	0	4.57E-06	0	0	0	0	9.66E-02	0	0	0	1	9.04E+00	0.020	0	0
31	0	3.58E-04	0	0	0	0.016	4.09E+00	0.038	0.038	0.168	1	1.17E+01	0.240	0	0.260
32	0	1.29E-05	0	0	0	0.002	1.81E+00	0	0	0	1	1.03E+01	0.040	0	0
33	0	5.16E-06	0	0	0	0	2.11E-01	0	0	0	1	8.03E+00	0	0	0
34	0	2.65E-04	0	1	1	0.038	4.49E+00	0.138	0.138	0.686	1	1.47E+01	0.540	1	0.720
35	0	5.92E-06	0	0	0	0	1.29E-01	0	0	0	1	5.90E+00	0	0	0
36	0	2.42E-06	0	0	0	0	1.35E-01	0	0	0	1	7.63E+00	0	0	0
37	0	5.16E-06	0	0	0	0	1.01E-01	0	0	0	1	7.01E+00	0	0	0
38	0	6.00E-02	0	1	0	0.424	8.80E+00	0.002	0.002	0	1	2.15E+01	0.120	1	0
39	0	1.91E-03	0	1	0	0.704	1.76E+01	0.060	0.060	0.024	1	2.56E+01	0.460	1	0.120
40	0	2.42E-06	0	0	0	0	1.03E-01	0	0	0	1	7.58E+00	0	0	0
41	0	1.91E-03	0	1	0	0.636	1.51E+01	0.046	0.046	0.030	1	2.62E+01	0.340	1	0.040
42	0	1.85E-04	0	1	1	0.134	6.86E+00	0	0	1	1	1.76E+01	0.040	1	1
43	0	2.34E-06	0	0	0	0	2.04E-01	0	0	0	1	9.56E+00	0	0	0
44	1	4.94E+00	0	1	0	0.336	9.46E+00	0.232	0.232	0.180	1	2.22E+01	0.800	1	0.180
45	0	5.86E-06	0	0	0	0	3.56E-01	0	0	0	1	9.12E+00	0.020	0	0
46	0	2.21E-06	0	0	0	0	2.61E-02	0	0	0	1	9.32E+00	0.060	0	0
47	0	2.21E-06	0	0	0	0	2.90E-02	0	0	0	1	5.83E+00	0	0	0
48	0	1.04E-02	0	1	0	0.154	8.80E+00	0.350	0.350	0.096	1	1.76E+01	0.760	1	0.120
49	0	2.42E-06	0	0	0	0	4.71E-02	0	0	0	1	9.47E+00	0	0	0
50	0	3.58E-04	0	0	0	0.004	1.38E+00	0.024	0.024	0.146	1	1.20E+01	0.320	0	0.340
51	0	5.57E-02	1	1	1	0.352	1.05E+01	0.720	0.720	0.786	1	2.19E+01	0.980	1	0.900
52	0	1.04E-02	0	1	0	0.130	1.00E+01	0.352	0.352	0.106	1	1.84E+01	0.920	1	0.060
53	0	3.42E-06	0	0	0	0	3.86E-01	0.002	0.002	0	1	9.51E+00	0.060	0	0
54	0	2.65E-04	0	1	1	0.050	3.45E+00	0.112	0.112	0.686	1	1.53E+01	0.500	1	0.800
55	0	2.42E-06	0	0	0	0	2.25E-02	0	0	0	1	8.40E+00	0	0	0
56	0	1.23E-06	0	0	0	0	1.65E-02	0	0	0	1	8.95E+00	0.020	0	0
57	0	5.57E-02	1	1	1	0.380	8.90E+00	0.760	0.760	0.782	1	2.35E+01	1	1	0.920
58	0	1.74E-06	0	0	0	0	7.92E-02	0	0	0	1	9.38E+00	0	0	0
59	0	2.41E-01	0	1	0	0.464	1.10E+01	0.004	0.004	0.026	1	2.15E+01	0.440	0.980	0
60	0	1.71E-06	0	0	0	0	4.49E-01	0.002	0.002	0	1	4.68E+00	0	0	0
61	0	1.91E-03	0	1	0	0.640	1.59E+01	0.046	0.046	0.018	1	2.23E+01	0.160	1	0.080
62	0	2.08E-06	0	0	0	0	1.13E-02	0	0	0	1	8.68E+00	0.060	0	0
63	0	1.91E-03	0	1	0	0.722	1.55E+01	0.050	0.050	0.030	1	2.84E+01	0.580	1	0.400
64	0	2.69E-01	0	1	0	0.280	8.92E+00	0.028	0.028	0.032	1	2.02E+01	0.220	1	0.020

65	0	1.15E-01	1	1	0	0.710	1.52E+01	0.392	0.392	0.704	1	2.41E+01	0.960	0.900	0.820
66	0	1.66E-06	0	0	0	0	5.28E-02	0	0	0	1	8.49E+00	0.020	0	0
67	0	2.69E-01	0	1	0	0.236	8.97E+00	0.004	0.004	0.040	1	2.18E+01	0.300	1	0.040
68	0	2.34E-06	0	0	0	0	8.27E-02	0	0	0	1	8.35E+00	0	0	0
69	0	4.82E-05	0	0	0	0.012	4.40E+00	0.204	0.204	0.196	1	1.04E+01	0.700	0.660	0.380
70	0	1.61E-02	0	1	0	0.274	8.81E+00	0.054	0.054	0.574	1	2.02E+01	0.540	1	0.820
71	0	2.41E-01	0	1	0	0.482	1.47E+01	0.008	0.008	0.040	1	2.01E+01	0.140	0.860	0.060
72	0	1.15E-01	1	1	0	0.744	1.63E+01	0.396	0.396	0.672	1	2.40E+01	0.980	0.860	0.820
73	0	4.82E-05	0	0	0	0.010	1.71E+00	0.198	0.198	0.176	1	1.16E+01	0.740	0.600	0.280
74	0	1.29E-05	0	0	0	0.002	7.28E-01	0	0	0	1	1.25E+01	0.080	0	0
75	0	2.64E-04	0	1	0	0.020	1.93E+00	0.108	0.108	0.098	1	1.42E+01	0.560	1	0.340
76	0	2.64E-04	0	1	0	0.010	2.19E+00	0.106	0.106	0.120	1	1.23E+01	0.680	1	0.200
77	0	4.57E-06	0	0	0	0	5.05E-01	0	0	0	1	7.02E+00	0	0	0
78	0	2.42E-06	0	0	0	0	1.74E-02	0	0	0	1	6.75E+00	0	0	0
79	0	1.61E-02	0	1	0	0.240	8.12E+00	0.044	0.044	0.594	1	1.83E+01	0.400	1	0.860
80	0	7.94E-05	0	1	0	0.014	2.32E+00	0.008	0.008	0	1	1.25E+01	0.160	1	0
81	1	4.94E+00	0	1	0	0.302	7.63E+00	0.208	0.208	0.194	1	2.23E+01	0.680	1	0.300
82	0	1.91E-03	0	1	0	0.646	1.54E+01	0.058	0.058	0.018	1	2.71E+01	0.400	1	0.260
83	0	2.41E-01	0	1	0	0.446	1.12E+01	0.004	0.004	0.040	1	2.07E+01	0	0.960	0
84	0	4.82E-05	0	0	0	0.008	3.05E+00	0.234	0.234	0.204	1	1.15E+01	0.700	0.640	0.360
85	0	4.82E-05	0	0	0	0.010	1.51E+00	0.202	0.202	0.202	1	1.33E+01	0.760	0.620	0.340
86	0	3.58E-04	0	0	0	0.006	2.16E+00	0.030	0.030	0.170	1	1.36E+01	0.420	0	0.160
87	0	3.58E-04	0	0	0	0.016	3.95E+00	0.026	0.026	0.122	1	8.57E+00	0.200	0.020	0.120
88	0	8.33E-06	0	0	0	0.002	7.78E-01	0.002	0.002	0	1	1.22E+01	0.100	0	0
89	1	3.59E+00	0	0	1	0.348	1.09E+01	0	0	0.506	1	2.09E+01	0	0	0.740
90	0	2.17E-06	0	1	0	0	1.24E-01	0.020	0.020	0	1	9.23E+00	0.120	1	0
91	0	2.08E-06	0	0	0	0	1.09E-01	0	0	0	1	8.03E+00	0	0	0
92	1	4.94E+00	0	1	0	0.316	1.15E+01	0.230	0.230	0.176	1	2.51E+01	0.720	1	0.360
93	0	2.08E-06	0	0	0	0	6.99E-02	0	0	0	1	8.54E+00	0	0	0
94	0	1.66E-06	0	0	0	0	9.20E-02	0	0	0	1	4.12E+00	0	0	0
95	0	7.78E-05	0	1	0	0.018	2.52E+00	0.004	0.004	0.228	1	1.47E+01	0.320	0.980	0.180
96	0	2.41E-01	0	1	0	0.478	1.09E+01	0	0	0.028	1	2.13E+01	0.100	0.860	0.280
97	0	1.71E-06	0	0	0	0	1.01E-01	0	0	0	1	6.74E+00	0	0	0
98	0	2.21E-06	0	0	0	0	5.94E-03	0	0	0	1	8.15E+00	0	0	0
99	0	2.71E-01	1	1	1	0.402	1.34E+01	0.704	0.704	0.678	1	2.21E+01	1	1	1
100	0	5.49E-06	0	0	0	0	1.16E-01	0	0	0	1	9.79E+00	0.020	0	0

% Packets Fool Model	**Avg Loss**	**% Packets Fool Model**	**% Packets Fool Model**	**% Packets Fool Model**	**% Packets Fool Model**	**Avg Loss**	**% Packets Fool Model**	**% Packets Fool Model**	**% Packets Fool Model**	**% Packets Fool Model**	**Avg Loss**	**% Packets Fool Model**	**% Packets Fool Model**	**% Packets Fool Model**
6%	3.03E-01	0.05	0.43	12%	61%	4.93E+00	56%	56%	55%	100%	14.4306161	69%	53%	52%

Table 31. Detailed performance of hulk adversarial examples tested against the surrogate model and NIDS models.

	Hulk														
	Raw Packet					Random Perturbations					Near Optimal Perturbations				
Trial	Fool Surrogate	Loss Surrogate	Fool CNN NIDS	Fool FNN NIDS	Fool Adaboost NIDS	Rate Fool Surrogate	Best Obs Loss Surrogate	Rate Fool CNN NIDS	Rate Fool FNN NIDS	Rate Fool Adaboost NIDS	Rate Fool Surrogate	Best Obs Loss Surrogate	Rate Fool CNN NIDS	Rate Fool FNN NIDS	Rate Fool Adaboost NIDS
1	0	4.65E-06	0	0	0	0	0.22569913	0	0	0	1	6.87E+00	0.060	0	0
2	0	5.65E-06	0	0	0	0	0.34041515	0	0	0	1	1.11E+01	0.020	0	0
3	0	6.98E-06	0	0	0	0.004	3.43680763	0	0	0	1	1.01E+01	0.040	0	0
4	0	2.34E-06	0	0	0	0	0.078176	0	0	0	1	8.91E+00	0	0	0
5	0	6.71E-06	0	0	0	0	0.50846976	0	0	0	1	1.09E+01	0.040	0	0
6	0	5.26E-06	0	0	0	0	0.24862447	0	0	0	1	8.55E+00	0	0	0
7	0	1.25E-05	0	0	0	0	0.47634339	0	0	0	1	1.14E+01	0	0	0
8	0	2.46E-06	0	0	0	0	0.01227814	0	0	0	1	8.83E+00	0	0	0
9	0	7.34E-06	0	0	0	0	0.26315776	0	0	0	1	7.98E+00	0.020	0	0
10	0	3.41E-06	0	0	0	0	0.07128154	0	0	0	1	9.60E+00	0	0	0
11	0	3.29E-06	0	0	0	0.002	0.78402275	0	0	0	1	7.41E+00	0	0	0
12	0	1.56E-06	0	0	0	0	0.0462038	0	0	0	1	6.00E+00	0	0	0
13	0	1.95E-06	0	0	0	0	0.1242239	0.002	0.002	0	1	6.02E+00	0	0	0
14	0	3.49E-06	0	0	0	0	0.2483108	0	0	0	1	4.36E+00	0	0	0
15	0	2.01E-06	0	0	0	0.002	2.22076559	0	0	0	1	7.26E+00	0	0	0
16	0	6.82E-06	0	0	0	0	0.11436655	0	0	0	1	1.23E+01	0	0	0
17	0	5.46E-06	0	0	0	0	0.09230862	0	0	0	1	1.19E+01	0	0	0
18	0	3.56E-06	0	0	0	0	0.24921878	0	0	0	1	9.45E+00	0	0	0
19	0	1.25E-06	0	0	0	0	0.05747212	0	0	0	1	8.55E+00	0	0	0
20	0	5.84E-06	0	0	0	0.002	0.89910471	0	0	0	1	6.57E+00	0	0	0
21	0	1.36E-06	0	0	0	0	0.00731815	0	0	0	1	5.24E+00	0	0	0
22	0	5.71E-06	0	0	0	0	0.20557572	0	0	0	1	1.05E+01	0	0	0
23	0	4.72E-06	0	0	0	0	0.31241429	0.002	0.002	0	1	1.07E+01	0.020	0	0
24	0	4.66E-06	0	0	0	0.002	1.21047401	0	0	0	1	1.06E+01	0	0	0
25	0	2.88E-06	0	0	0	0	0.5140608	0	0	0	1	1.21E+01	0.040	0	0
26	0	1.87E-06	0	0	0	0	0.05437756	0	0	0	1	6.14E+00	0.000	0	0
27	0	2.88E-06	0	0	0	0	0.51302612	0	0	0	1	1.05E+01	0	0	0
28	0	4.76E-06	0	0	0	0	0.1736289	0	0	0	1	9.93E+00	0	0	0
29	0	3.73E-06	0	0	0	0	0.09984298	0	0	0	1	7.32E+00	0.020	0	0
30	0	4.65E-06	0	0	0	0	0.19377699	0.004	0.004	0	1	9.89E+00	0.160	0	0
31	0	8.83E-06	0	0	0	0	0.16572484	0	0	0	1	8.14E+00	0	0	0
32	0	1.62E-06	0	0	0	0	0.02611746	0	0	0	1	9.37E+00	0	0	0
33	0	6.20E-06	0	0	0	0.002	1.00848961	0	0	0	1	1.01E+01	0.040	0	0
34	0	3.64E-06	0	0	0	0	0.0723903	0	0	0	1	8.09E+00	0	0	0
35	0	1.07E-06	0	0	0	0	0.02234378	0	0	0	1	5.33E+00	0	0	0
36	0	1.83E-06	0	0	0	0.002	2.15884089	0	0	0	1	5.12E+00	0	0	0
37	0	2.13E-06	0	0	0	0	0.05282377	0	0	0	1	6.89E+00	0	0	0
38	0	2.42E-06	0	0	0	0	0.16941246	0	0	0	1	7.78E+00	0	0	0
39	0	3.26E-06	0	0	0	0	0.09278957	0	0	0	1	1.04E+01	0.040	0	0
40	0	1.21E-05	0	0	0	0	0.4437063	0	0	0	1	7.79E+00	0	0	0
41	0	6.81E-06	0	0	0	0	0.58851516	0	0	0	1	9.42E+00	0	0	0
42	0	6.92E-06	0	0	0	0	0.26744801	0	0	0	1	7.63E+00	0.020	0	0
43	0	1.07E-06	0	0	0	0	0.04822018	0	0	0	1	7.63E+00	0	0	0
44	0	1.46E-05	0	0	0	0.002	2.53316045	0	0	0	1	1.17E+01	0.040	0	0
45	0	5.75E-06	0	0	0	0	0.52664435	0	0	0	1	8.30E+00	0.020	0	0
46	0	1.90E-05	0	0	0	0	0.28408998	0	0	0	1	1.34E+01	0.100	0	0
47	0	4.95E-06	0	0	0	0	0.10006998	0	0	0	1	8.22E+00	0.060	0	0
48	0	9.19E-06	0	0	0	0	0.07676287	0.002	0.002	0	1	7.33E+00	0	0	0
49	0	3.32E-06	0	0	0	0	0.17553589	0	0	0	1	7.11E+00	0	0	0
50	0	2.64E-06	0	0	0	0	0.08336618	0.002	0.002	0	1	4.76E+00	0.040	0	0
51	0	6.21E-06	0	0	0	0	0.45666274	0	0	0	1	1.13E+01	0.060	0	0
52	0	4.37E-06	0	0	0	0	0.37189314	0.002	0.002	0	1	6.08E+00	0.020	0	0
53	0	3.29E-06	0	0	0	0	0.03646174	0	0	0	1	7.60E+00	0.040	0	0
54	0	3.87E-06	0	0	0	0	2.16E-02	0	0	0	1	7.89E+00	0.040	0	0
55	0	6.47E-06	0	0	0	0	1.70E-01	0	0	0	1	7.40E+00	0	0	0
56	0	2.58E-06	0	0	0	0	1.09E-02	0	0	0	1	8.71E+00	0	0	0
57	0	1.52E-05	0	0	0	0.002	8.04E-01	0	0	0	1	1.40E+01	0.140	0	0
58	0	4.11E-06	0	0	0	0	1.52E-01	0	0	0	1	8.60E+00	0	0	0
59	0	4.08E-06	0	0	0	0	2.87E-01	0	0	0	1	7.19E+00	0.040	0	0
60	0	2.01E-06	0	0	0	0	2.43E-01	0.002	0.002	0	1	6.90E+00	0.040	0	0
61	0	4.80E-06	0	0	0	0	3.16E-02	0	0	0	1	6.59E+00	0	0	0
62	0	6.20E-06	0	0	0	0	2.00E-01	0.002	0.002	0	1	8.81E+00	0.020	0	0
63	0	6.81E-06	0	0	0	0	1.18E-01	0	0	0	1	8.47E+00	0.200	0	0
64	0	2.07E-06	0	0	0	0	3.07E-01	0.002	0.002	0	1	1.29E+01	0.080	0	0

65	0	6.62E-06	0	0	0	0	3.20E-01	0	0	0	1	9.65E+00	0.100	0	0
66	0	2.74E-06	0	0	0	0	2.22E-01	0	0	0	1	1.11E+01	0	0	0
67	0	2.17E-06	0	0	0	0	7.51E-02	0	0	0	1	8.83E+00	0	0	0
68	0	1.47E-06	0	0	0	0	8.97E-02	0	0	0	1	4.38E+00	0	0	0
69	0	6.30E-06	0	0	0	0.002	1.38E+00	0	0	0	1	7.07E+00	0	0	0
70	0	2.74E-06	0	0	0	0	1.87E-01	0	0	0	1	5.72E+00	0	0	0
71	0	1.26E-06	0	0	0	0	8.68E-03	0	0	0	1	5.93E+00	0	0	0
72	0	2.96E-06	0	0	0	0	3.09E-02	0	0	0	1	7.93E+00	0.040	0	0
73	0	4.18E-06	0	0	0	0	2.73E-01	0	0	0	1	8.18E+00	0.000	0	0
74	0	1.47E-06	0	0	0	0	1.64E-02	0	0	0	1	5.87E+00	0.000	0	0
75	0	2.34E-06	0	0	0	0	3.24E-02	0	0	0	1	2.54E+00	0.000	0	0
76	0	1.44E-06	0	0	0	0	7.43E-02	0	0	0	1	6.36E+00	0.000	0	0
77	0	2.45E-06	0	0	0	0	4.77E-02	0	0	0	1	6.95E+00	0.000	0	0
78	0	6.71E-06	0	0	0	0.002	9.95E-01	0	0	0	1	8.42E+00	0.000	0	0
79	0	2.46E-06	0	0	0	0	2.74E-01	0	0	0	1	9.74E+00	0.000	0	0
80	0	6.84E-06	0	0	0	0	5.07E-02	0	0	0	1	8.99E+00	0.040	0	0
81	0	2.12E-05	0	0	0	0	2.74E-01	0	0	0	1	9.52E+00	0.000	0	0
82	0	5.52E-06	0	0	0	0	2.77E-01	0	0	0	1	6.33E+00	0.000	0	0
83	0	8.40E-06	0	0	0	0	2.50E-01	0	0	0	1	8.58E+00	0.000	0	0
84	0	4.11E-06	0	0	0	0	1.26E-01	0	0	0	1	1.02E+01	0.160	0	0
85	0	6.98E-06	0	0	0	0.002	2.71E+00	0	0	0	1	8.13E+00	0.000	0	0
86	0	5.36E-06	0	0	0	0	9.69E-02	0	0	0	1	8.34E+00	0.000	0	0
87	0	1.14E-06	0	0	0	0	6.28E-02	0	0	0	1	4.32E+00	0.000	0	0
88	0	1.24E-05	0	0	0	0.002	1.58E+00	0.004	0.004	0	1	1.08E+01	0.300	0	0
89	0	2.73E-06	0	0	0	0	1.00E-01	0.002	0.002	0	1	7.26E+00	0.020	0	0
90	0	4.10E-06	0	0	0	0	3.60E-02	0	0	0	1	6.50E+00	0.000	0	0
91	0	2.55E-06	0	0	0	0	1.21E-02	0	0	0	1	4.76E+00	0.000	0	0
92	0	6.01E-06	0	0	0	0	9.98E-02	0	0	0	1	9.69E+00	0.040	0	0
93	0	1.61E-06	0	0	0	0	1.04E-02	0	0	0	1	7.36E+00	0.000	0	0
94	0	1.68E-06	0	0	0	0	6.17E-02	0	0	0	1	7.67E+00	0.000	0	0
95	0	2.49E-06	0	0	0	0	6.29E-01	0	0	0	1	1.01E+01	0.040	0	0
96	0	3.51E-06	0	0	0	0	6.93E-02	0	0	0	1	4.82E+00	0.000	0	0
97	0	3.82E-06	0	0	0	0	6.77E-01	0.004	0.004	0	1	1.15E+01	0.060	0	0
98	0	1.60E-05	0	0	0	0	2.41E-01	0	0	0	1	8.09E+00	0.000	0	0
99	0	6.01E-06	0	0	0	0	2.64E-01	0	0	0	1	1.06E+01	0.000	0	0
100	0	2.26E-06	0	0	0	0	1.47E-01	0	0	0	1	7.78E+00	0.020	0	0
	% Packets Fool Model	**Avg Loss**	**% Packets Fool Model**	**% Packets Fool Model**	**% Packets Fool Model**	**% Packets Fool Model**	**Avg Loss**	**% Packets Fool Model**	**% Packets Fool Model**	**% Packets Fool Model**	**% Packets Fool Model**	**Avg Loss**	**% Packets Fool Model**	**% Packets Fool Model**	**% Packets Fool Model**
	0%	4.91E-06	0	0	0%	13%	3.77E-01	12%	12%	0%	100%	8.33952444	37%	0%	0%

Table 32. Detailed performance of SSH adversarial examples tested against the surrogate model and NIDS models.

	SSH														
	Raw Packet					Random Perturbations					Near Optimal Perturbations				
Trial	Fool Surrogat e	Loss Surrogat e	Fool CNN NIDS	Fool FNN NIDS	Fool Adaboost NIDS	Rate Fool Surrogat e	Best Obs Loss Surrogat e	Rate Fool CNN NIDS	Rate Fool FNN NIDS	Rate Fool Adaboost NIDS	Rate Fool Surrogat e	Best Obs Loss Surrogat e	Rate Fool CNN NIDS	Rate Fool FNN NIDS	Rate Fool Adaboost NIDS
1	0	3.30E-10	0	0	0	0	2.88E-05	0	0	0	0	4.93E-02	0	0	0
2	0	4.12E-10	0	0	0	0	9.55E-06	0	0	0	0	4.42E-02	0	0	0
3	0	6.63E-10	0	0	0	0	4.13E-05	0	0	0	0	3.57E-01	0	0	0
4	0	1.46E-08	0	0	0	0	7.05E-04	0	0	0	0	2.73E-01	0	0	0
5	0	1.66E-09	0	0	0	0	4.71E-05	0	0	0	1	1.37E+00	0	0	0
6	0	4.04E-09	0	0	0	0	2.68E-04	0	0	0	1	1.04E+00	0	0	0
7	0	1.94E-09	0	0	0	0	6.61E-05	0	0	0	0	6.03E-02	0	0	0
8	0	1.47E-09	0	0	0	0	6.12E-06	0	0	0	0	7.06E-03	0	0	0
9	0	9.18E-09	0	0	0	0	7.19E-03	0	0	0	1	4.68E+00	0	0	0
10	0	3.43E-09	0	0	0	0	3.78E-05	0	0	0	0	4.79E-02	0	0	0
11	0	3.81E-10	0	0	0	0	2.71E-04	0	0	0	1	9.67E-01	0	0	0
12	0	9.46E-10	0	0	0	0	8.21E-06	0	0	0	0	3.35E-02	0	0	0
13	0	7.74E-10	0	0	0	0	6.99E-06	0	0	0	0	5.87E-03	0	0	0
14	0	1.83E-09	0	0	0	0	8.09E-05	0	0	0	0	1.76E-02	0	0	0
15	0	2.15E-10	0	0	0	0	1.38E-05	0	0	0	0	3.25E-01	0	0	0
16	0	9.00E-09	0	0	0	0	9.75E-02	0	0	0	1	6.50E+00	0	0	0
17	0	1.78E-07	0	0	0	0	1.69E-01	0	0	0	1	8.66E+00	0	0	0
18	0	9.04E-09	0	0	0	0	1.16E-03	0	0	0	1	2.73E+00	0	0	0
19	0	1.68E-09	0	0	0	0	9.17E-06	0	0	0	0	1.71E-02	0	0	0
20	0	2.85E-10	0	0	0	0	3.17E-06	0	0	0	0	2.68E-03	0	0	0
21	0	9.40E-10	0	0	0	0	6.05E-06	0	0	0	0	8.51E-03	0	0	0
22	0	6.21E-10	0	0	0	0	5.17E-06	0	0	0	0	9.54E-03	0	0	0
23	0	1.59E-10	0	0	0	0	1.96E-06	0	0	0	0	6.37E-03	0	0	0
24	0	1.63E-08	0	0	0	0	1.34E-02	0	0	0	1	4.51E+00	0	0	0
25	0	9.08E-09	0	0	0	0	8.03E-04	0	0	0	0	3.12E-01	0	0	0
26	0	4.37E-09	0	0	0	0	5.00E-05	0	0	0	0	3.79E-02	0	0	0
27	0	1.80E-09	0	0	0	0	7.72E-06	0	0	0	0	6.30E-03	0	0	0
28	0	3.65E-10	0	0	0	0	1.99E-04	0	0	0	0	6.01E-01	0	0	0
29	0	5.09E-10	0	0	0	0	4.84E-06	0	0	0	0	1.70E-03	0	0	0
30	0	3.96E-09	0	0	0	0	8.99E-04	0	0	0	1	3.41E+00	0	0	0
31	0	2.14E-10	0	0	0	0	2.74E-06	0	0	0	0	1.43E-03	0	0	0
32	0	1.78E-07	0	0	0	0	1.81E-01	0	0	0	1	6.23E+00	0	0	0
33	0	4.60E-09	0	0	0	0	2.30E-04	0	0	0	0	3.57E-01	0	0	0
34	0	1.09E-09	0	0	0	0	2.05E-05	0	0	0	0	6.95E-02	0	0	0
35	0	1.42E-09	0	0	0	0	5.48E-04	0	0	0	1	1.27E+00	0	0	0
36	0	1.69E-08	0	0	0	0	8.96E-03	0	0	0	1	6.73E+00	0	0	0
37	0	5.88E-10	0	0	0	0	3.47E-05	0	0	0	0	4.11E-03	0	0	0
38	0	2.57E-08	0	0	0	0	1.45E-02	0	0	0	1	3.75E+00	0	0	0
39	0	1.95E-09	0	0	0	0	5.84E-05	0	0	0	0	9.26E-02	0	0	0
40	0	3.38E-09	0	0	0	0	5.34E-04	0	0	0	0	2.81E-01	0	0	0
41	0	1.22E-09	0	0	0	0	4.99E-05	0	0	0	1	1.41E+00	0	0	0
42	0	4.60E-08	0	0	0	0	1.19E-03	0	0	0	1	1.40E+00	0	0	0
43	0	4.96E-10	0	0	0	0	1.42E-04	0	0	0	1	8.26E-01	0	0	0
44	0	2.53E-08	0	0	0	0	3.12E-02	0	0	0	1	5.40E+00	0	0	0
45	0	3.25E-09	0	0	0	0	4.95E-05	0	0	0	0	4.43E-03	0	0	0
46	0	3.55E-10	0	0	0	0	2.22E-06	0	0	0	0	1.04E-03	0	0	0
47	0	2.36E-10	0	0	0	0	6.72E-06	0	0	0	0	5.14E-04	0	0	0
48	0	6.21E-10	0	0	0	0	8.99E-06	0	0	0	0	2.50E-03	0	0	0
49	0	5.33E-09	0	0	0	0	4.50E-04	0	0	0	1	1.34E+00	0	0	0
50	0	1.02E-09	0	0	0	0	3.81E-05	0	0	0	0	5.10E-01	0	0	0
51	0	5.40E-10	0	0	0	0	1.28E-05	0	0	0	0	3.11E-03	0	0	0
52	0	1.43E-09	0	0	0	0	1.86E-04	0	0	0	0	3.05E-01	0	0	0
53	0	4.33E-09	0	0	0	0	7.69E-05	0	0	0	0	1.09E-02	0	0	0
54	0	5.00E-10	0	0	0	0	2.77E-06	0	0	0	0	9.01E-04	0	0	0
55	0	2.87E-09	0	0	0	0	2.09E-04	0	0	0	0	2.05E-01	0	0	0
56	0	1.50E-08	0	0	0	0	7.40E-04	0	0	0	0	3.35E-01	0	0	0
57	0	1.14E-09	0	0	0	0	1.52E-04	0	0	0	0	5.79E-01	0	0	0
58	0	2.53E-08	0	0	0	0	9.18E-02	0	0	0	1	6.07E+00	0	0	0
59	0	7.29E-08	0	0	0	0	1.47E-03	0	0	0	1	1.31E+00	0	0	0
60	0	2.33E-09	0	0	0	0	4.39E-05	0	0	0	0	2.49E-02	0	0	0
61	0	3.29E-10	0	0	0	0	1.32E-05	0	0	0	0	3.53E-03	0	0	0
62	0	4.10E-09	0	0	0	0	3.11E-04	0	0	0	1	7.07E-01	0	0	0
63	0	4.00E-08	0	0	0	0	2.10E-02	0	0	0	1	6.25E+00	0	0	0
64	0	1.01E-10	0	0	0	0	5.46E-07	0	0	0	0	1.07E-03	0	0	0

65	0	1.72E-09	0	0	0	0	3.47E-04	0	0	0	0	3.91E-01	0	0	0
66	0	6.34E-08	0	0	0	0	9.44E-03	0	0	0	1	4.48E+00	0	0	0
67	0	2.82E-09	0	0	0	0	2.34E-05	0	0	0	0	1.51E-02	0	0	0
68	0	5.40E-10	0	0	0	0	7.57E-06	0	0	0	0	1.15E-02	0	0	0
69	0	3.67E-07	0	0	0	0.008	2.05E+00	0	0	0	1	1.12E+01	0	0	0
70	0	2.35E-10	0	0	0	0	6.06E-06	0	0	0	0	1.95E-03	0	0	0
71	0	9.04E-09	0	0	0	0	1.18E-03	0	0	0	1	8.52E-01	0	0	0
72	0	8.19E-10	0	0	0	0	2.82E-04	0	0	0	1	1.62E+00	0	0	0
73	0	2.34E-09	0	0	0	0	4.07E-04	0	0	0	1	3.28E+00	0	0	0
74	0	1.74E-09	0	0	0	0	1.06E-03	0	0	0	0	4.73E-01	0	0	0
75	0	2.21E-09	0	0	0	0	1.49E-04	0	0	0	1	9.35E-01	0	0	0
76	0	8.48E-09	0	0	0	0	3.14E-04	0	0	0	0	5.19E-01	0	0	0
77	0	1.22E-10	0	0	0	0	4.38E-07	0	0	0	0	7.01E-05	0	0	0
78	0	1.38E-08	0	0	0	0	1.46E-03	0	0	0	1	2.52E+00	0	0	0
79	0	9.16E-09	0	0	0	0	3.76E-04	0	0	0	0	2.53E-01	0	0	0
80	0	1.58E-10	0	0	0	0	5.92E-06	0	0	0	0	1.90E-03	0	0	0
81	0	1.94E-09	0	0	0	0	4.00E-05	0	0	0	0	3.92E-02	0	0	0
82	0	3.24E-08	0	0	0	0	1.46E-01	0	0	0	1	4.34E+00	0	0	0
83	0	7.20E-11	0	0	0	0	1.02E-06	0	0	0	0	2.49E-04	0	0	0
84	0	1.69E-09	0	0	0	0	9.33E-05	0	0	0	0	3.77E-01	0	0	0
85	0	1.13E-09	0	0	0	0	5.73E-05	0	0	0	0	3.99E-02	0	0	0
86	0	2.08E-07	0	0	0	0	1.72E-01	0	0	0	1	6.84E+00	0	0	0
87	0	7.39E-09	0	0	0	0	4.74E-04	0	0	0	1	1.92E+00	0	0	0
88	0	6.42E-09	0	0	0	0	7.40E-04	0	0	0	0	6.08E-01	0	0	0
89	0	2.91E-09	0	0	0	0	1.13E-04	0	0	0	0	5.56E-01	0	0	0
90	0	9.18E-09	0	0	0	0	8.88E-03	0	0	0	1	3.00E+00	0	0	0
91	0	1.53E-10	0	0	0	0	2.52E-05	0	0	0	0	2.19E-01	0	0	0
92	0	1.02E-09	0	0	0	0	2.92E-05	0	0	0	0	2.22E-01	0	0	0
93	0	6.88E-09	0	0	0	0	3.07E-04	0	0	0	0	5.09E-01	0	0	0
94	0	6.05E-08	0	0	0	0	5.76E-02	0	0	0	1	1.96E+00	0	0	0
95	0	1.36E-09	0	0	0	0	1.86E-05	0	0	0	0	7.27E-03	0	0	0
96	0	5.54E-08	0	0	0	0	8.75E-03	0	0	0	1	5.45E+00	0	0	0
97	0	2.97E-08	0	0	0	0	2.23E-03	0	0	0	1	2.33E+00	0	0	0
98	0	3.80E-10	0	0	0	0	1.98E-06	0	0	0	0	1.30E-03	0	0	0
99	0	7.04E-10	0	0	0	0	8.02E-04	0	0	0	1	1.37E+00	0	0	0
100	0	6.36E-10	0	0	0	0	1.67E-05	0	0	0	0	1.08E-01	0	0	0

% Packets Fool Model	Avg Loss	% Packets Fool Model	% Packets Fool Model	% Packets Fool Model	% Packets Fool Model	Avg Loss	% Packets Fool Model	% Packets Fool Model	% Packets Fool Model	% Packets Fool Model	Avg Loss	% Packets Fool Model	% Packets Fool Model	% Packets Fool Model
0%	1.68E-06	0	0	0%	1%	3.11E-02	0%	0%	0%	37%	1.38E+00	0%	0%	0%

Bibliography

1. Marc Chalé and Nathaniel D. Bastian, "Generating realistic cyber data for training and evaluating machine learning classifiers for network intrusion detection systems," *Expert Systems with Applications*, vol. 207, pp. 117936, 2022.

2. Marc Chalé, "Algorithm selection framework for cyber attack detection," in *Proceedings 2nd ACM Workshop on Wireless Security and Machine Learning (WiseML '20)*, New York, NY, 2020, Association for Computing Machinery.

3. Yoshua Bengio, Ian Goodfellow, and Aaron Courville, *Deep learning*, vol. 1, MIT press Massachusetts, USA:, 2017.

4. Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni, "Modeling tabular data using conditional gan," *33rd Conference on Neual Information Processing Systems*, 2019.

5. Megan K. Woods, "A Metamodel Recommendation System Using Meta-Learning," M.S. thesis, Air Force Institute of Technology, 2020.

6. Department of Defense, Washington, DC, *Summary of the 2018 National Defense Strategy of the United States of America*, Jan 2018, Available at *https://dod.defense.gov/Portals/1/Documents/pubs/2018-National-Defense-Strategy-Summary.pdf*.

7. Scott D Applegate, "The dawn of kinetic cyber," in *2013 5th international conference on cyber conflict (CYCON 2013)*. IEEE, 2013, pp. 1–15.

8. M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, 2009, pp. 1–6.

9. S. J. Stolfo, Wei Fan, Wenke Lee, A. Prodromidis, and P. K. Chan, "Cost-based modeling for fraud and intrusion detection: results from the jam project," in *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*, 2000, vol. 2, pp. 130–144 vol.2.

10. Nour Moustafa and Jill Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *2015 military communications and information systems conference (MilCIS)*. IEEE, 2015, pp. 1–6.

11. Elie Alhajjar, Paul Maxwell, and Nathaniel D. Bastian, "Adversarial machine learning in network intrusion detection systems," 2020.

12. Ishai Rosenberg, Asaf Shabtai, Yuval Elovici, and Lior Rokach, "Adversarial machine learning attacks and defense methods in the cyber security domain," *ACM Comput. Surv.*, vol. 54, no. 5, May 2021.

13. Joseph Clements, Yuzhe Yang, Ankur Sharma, Hongxin Hu, and Yingjie Lao, "Rallying adversarial techniques against deep learning for network security," *arXiv preprint arXiv:1903.11688*, 2019.

14. Zilong Lin, Yong Shi, and Zhi Xue, "Idsgan: Generative adversarial networks for attack generation against intrusion detection," 2021.

15. Yuanzhang Li, Yaxiao Wang, Ye Wang, Lishan Ke, and Yu-an Tan, "A feature-vector generative adversarial network for evading pdf malware classifiers," *Information Sciences*, vol. 523, pp. 38–48, 2020.

16. Mouna Boujrad, Saiida Lazaar, and Mohammed Hassine, "Performance assessment of open source ids for improving iot architecture security implemented on wbans," in *Proceedings of the 3rd International Conference on Networking, Information Systems Security*, New York, NY, USA, 2020, NISS2020, Association for Computing Machinery.

17. Thomas H Ptacek and Timothy N Newsham, "Insertion, evasion, and denial of service: Eluding network intrusion detection," Tech. Rep., Secure Networks inc Calgary Alberta, 1998.

18. Qiang Liu, Pan Li, Wentao Zhao, Wei Cai, Shui Yu, and Victor CM Leung, "A survey on security threats and defensive techniques of machine learning: A data driven view," *IEEE access*, vol. 6, pp. 12103–12117, 2018.

19. James P Anderson, "Computer security technology planning study—vol 1," 1972.

20. James P Anderson, "Computer security threat monitoring and surveillance," 1980.

21. "Early computer security papers: Seminal papers," World Wide Web Page, Available at
http://seclab.cs.ucdavis.edu/projects/history/seminal.html.

22. Stefan Axelsson, "Intrusion detection systems: A survey and taxonomy," Tech. Rep., Technical report, 2000.

23. Federico Di Mattia, Paolo Galeone, Michele De Simoni, and Emanuele Ghelfi, "A survey on gans for anomaly detection," 2019.

24. David J Weller-Fahy, Brett J Borghetti, and Angela A Sodemann, "A survey of distance and similarity measures used within network intrusion anomaly detection," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1.

25. Spencer Butt, "Cyber data anomaly detection using autoencoder neural networks," M.S. thesis, Air Force Institute of Technology, 2018.

26. Rishi Shah, Jeff Gaston, Matthew Harvey, Michael McNamara, Osvaldo Ramos, Yeonsang You, and Elie Alhajjar, "Evaluating evasion attack methods on binary network traffic classifiers," in *Proceedings of the Conference on Information Systems Applied Research ISSN*, 2019, vol. 2167, p. 1508.

27. Karen Scarfone, Peter Mell, et al., "Guide to intrusion detection and prevention systems (idps)," *NIST special publication*, vol. 800, no. 2007, pp. 94, 2007.

28. Richard Bejtlich, *The practice of network security monitoring: understanding incident detection and response*, No Starch Press, 2013.

29. Varun Chandola, Arindam Banerjee, and Vipin Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, July 2009.

30. H. Hindy, D. Brosset, E. Bayne, A. K. Seeam, C. Tachtatzis, R. Atkinson, and X. Bellekens, "A taxonomy of network threats and the effect of current datasets on intrusion detection systems," *IEEE Access*, vol. 8, pp. 104650–104675, 2020.

31. Marc Chale, Bruce Cox, and Nathaniel Bastian, "Constrained optimization based adversarial example generation for transfer attacks in network intrusion detection systems," *Computers Industrial Engineering (under review)*, 2022.

32. Nathalie Japkowicz, Catherine Myers, Mark Gluck, et al., "A novelty detection approach to classification," in *IJCAI*. Citeseer, 1995, vol. 1, pp. 518–523.

33. Marc Chalé, Nathaniel Bastian, and Bruce Cox, "Generating realistic cyber data for training and evaluating machine learningclassifiers for network intrusion detection systems," unpublished, 2021.

34. Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and JD Tygar, *Adversarial machine learning*, Cambridge University Press, 2018.

35. Ishai Rosenberg, Asaf Shabtai, Yuval Elovici, and Lior Rokach, "Adversarial machine learning attacks and defense methods in the cyber security domain," *ACM Computing Surveys (CSUR)*, vol. 54, no. 5, pp. 1–36, 2021.

36. Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

37. Naveed Akhtar and Ajmal Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," *Ieee Access*, vol. 6, pp. 14410–14430, 2018.

38. Alesia Chernikova and Alina Oprea, "Fence: Feasible evasion attacks on neural networks in constrained environments," 2020.

39. Paul Maxwell, Elie Alhajjar, and Nathaniel D Bastian, "Intelligent feature engineering for cybersecurity," in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019, pp. 5005–5011.

40. Daiki Chiba, Kazuhiro Tobe, Tatsuya Mori, and Shigeki Goto, "Detecting malicious websites by learning ip address features," in *2012 IEEE/IPSJ 12th International Symposium on Applications and the Internet*, 2012, pp. 29–39.

41. S. Bradner and V. Paxson, "Rfc2780: Iana allocation guidelines for values in the internet protocol and related headers," 2000.

42. Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli, "Evasion attacks against machine learning at test time," in *Machine Learning and Knowledge Discovery in Databases*, Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Železný, Eds., Berlin, Heidelberg, 2013, pp. 387–402, Springer Berlin Heidelberg.

43. Alesia Chernikova and Alina Oprea, "Fence: Feasible evasion attacks on neural networks in constrained environments," *arXiv preprint arXiv:1909.10480*, 2019.

44. Giovanni Apruzzese, Mauro Andreolini, Luca Ferretti, Mirco Marchetti, and Michele Colajanni, "Modeling realistic adversarial attacks against network intrusion detection systems," *Digital Threats*, jun 2021, Accepted.

45. Michael J. De Lucia, Paul E. Maxwell, Nathaniel D. Bastian, Ananthram Swami, Brian Jalaian, and Nandi Leslie, "Machine learning raw network traffic detection," in *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications III*, Tien Pham and Latasha Solomon, Eds. International Society for Optics and Photonics, 2021, vol. 11746, pp. 185 – 194, SPIE.

46. David Bierbrauer, Michael DeLucia, Krishna Reddy, Paul Maxwell, and Nathaniel Bastian, "Transfer learning for raw network traffic detection," .

47. Weiwei Hu and Ying Tan, "Generating adversarial malware examples for black-box attacks based on gan," *arXiv preprint arXiv:1702.05983*, 2017.

48. Ian Goodfellow, "Adversarial examples and adversarial training," Lecture Material $http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture16.pdf, May2017$.

49. Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow, "Transferability in machine learning: from phenomena to black-box attacks using adversarial samples," *arXiv preprint arXiv:1605.07277*, 2016.

50. Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, Ken Nakae, and Shin Ishii, "Distributional smoothing with virtual adversarial training," *stat*, vol. 1050, pp. 29, 2016.

51. Tony Jebara, *Machine learning: discriminative and generative*, vol. 755, Springer Science & Business Media, 2012.

52. Ian Goodfellow, "Nips 2016 tutorial: Generative adversarial networks," 2017.

53. Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

54. Chelsea Finn, Ian Goodfellow, and Sergey Levine, "Unsupervised learning for physical interaction through video prediction," *arXiv preprint arXiv:1605.07157*, 2016.

55. Chelsea Finn and Sergey Levine, "Deep visual foresight for planning robot motion," *CoRR*, vol. abs/1610.00696, 2016.

56. Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1125–1134.

57. Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio, "Generative adversarial networks," *Commun. ACM*, vol. 63, no. 11, pp. 139–144, Oct. 2020.

58. LJ Bain and M Engelhardt, *Introduction to probability and mathematical*, 1992.

59. Scott E Fahlman, Geoffrey E Hinton, and Terrence J Sejnowski, "Massively parallel architectures for al: Netl, thistle, and boltzmann machines," in *National Conference on Artificial Intelligence, AAAI*, 1983.

60. David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski, "A learning algorithm for boltzmann machines," *Cognitive science*, vol. 9, no. 1, pp. 147–169, 1985.

61. Geoffrey E Hinton, Terrence J Sejnowski, and David H Ackley, *Boltzmann machines: Constraint satisfaction networks that learn*, Carnegie-Mellon University, Department of Computer Science Pittsburgh, PA, 1984.

62. Geoffrey E Hinton and Ruslan R Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

63. Andrew Gelman, John B Carlin, Hal S Stern, David B Dunson, Aki Vehtari, and Donald B Rubin, *Bayesian data analysis*, CRC press, 2013.

64. W Keith Hastings, "Monte carlo sampling methods using markov chains and their applications," 1970.

65. Robert E Kass, Bradley P Carlin, Andrew Gelman, and Radford M Neal, “Markov chain monte carlo in practice: a roundtable discussion,” *The American Statistician*, vol. 52, no. 2, pp. 93–100, 1998.

66. Matthew D Hoffman and Andrew Gelman, “The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo.,” *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1593–1623, 2014.

67. John Salvatier, Thomas V. Wiecki, and Christopher Fonnesbeck, “Probabilistic programming in python using pymc3,” *PeerJ Computer Science*, vol. 2, pp. e55, Apr. 2016.

68. Ilker Yildirim, “Bayesian inference: Metropolis-hastings sampling,” *Dept. of Brain and Cognitive Sciences, Univ. of Rochester, Rochester, NY*, 2012.

69. Yee Whye Teh, “Dirichlet process.,” 2010.

70. Or Litany, Ari Morcos, Srinath Sridhar, Leonidas Guibas, and Judy Hoffman, “Representation learning through latent canonicalizations,” 2020.

71. Amir R. Zamir, Alexander Sax, William Shen, Leonidas J. Guibas, Jitendra Malik, and Silvio Savarese, “Taskonomy: Disentangling task transfer learning,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

72. Chelsea Finn, Pieter Abbeel, and Sergey Levine, “Model-agnostic meta-learning for fast adaptation of deep networks,” 2017.

73. Allan Zhou, Tom Knowles, and Chelsea Finn, “Meta-learning symmetries by reparameterization,” 2020.

74. Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin, “” why should i trust you?”: Explaining the predictions of any classifier,” *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144, 2016.

75. John McHugh, “Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory,” *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 4, pp. 262–294, 2000.

76. Tim Merino, Matt Stillwell, Mark Steele, Max Coplan, Jon Patton, Alexander Stoyanov, and Lin Deng, “Expansion of cyber attack data from unbalanced datasets using generative adversarial networks,” in *International Conference on Software Engineering Research, Management and Applications.* Springer, 2019, pp. 131–145.

77. Nour Moustafa and Jill Slay, “Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set),” in *2015 military communications and information systems conference (MilCIS)*. IEEE, 2015, pp. 1–6.

78. Christian Rossow, Christian J Dietrich, Chris Grier, Christian Kreibich, Vern Paxson, Norbert Pohlmann, Herbert Bos, and Maarten Van Steen, “Prudent practices for designing malware experiments: Status quo and outlook,” in *2012 IEEE Symposium on Security and Privacy*. IEEE, 2012, pp. 65–79.

79. Sebastian Garcia, Martin Grill, Jan Stiborek, and Alejandro Zunino, “An empirical comparison of botnet detection methods,” *computers & security*, vol. 45, pp. 100–123, 2014.

80. Herbert Scarf, “A min-max solution of an inventory problem,” *Studies in the mathematical theory of inventory and production*, 1958.

81. Matthew Staib and Stefanie Jegelka, “Distributionally robust optimization and generalization in kernel methods,” *Advances in Neural Information Processing Systems*, vol. 32, 2019.

82. M. Staib, “Distributionally robust deep learning as a generalization of adversarial training,” 2017.

83. Rui Gao, Xi Chen, and Anton J Kleywegt, “Wasserstein distributionally robust optimization and variation regularization,” *arXiv preprint arXiv:1712.06050*, 2017.

84. Carmine Cicalese, “Redefining information operations,” Tech. Rep., National Defense University Fort McNair, 2013.

85. United States Department of the Army, “Cyberspace operations concept capability plan,” 2010.

86. Daniel Ventre, *Artificial Intelligence, Cybersecurity and Cyber Defense*, John Wiley & Sons, 2020.

87. Laura Brent, “Nato’s role in cyberspace,” Feb. 2019.

88. Amin Azmoodeh, Ali Dehghantanha, and Kim-Kwang Raymond Choo, “Robust malware detection for internet of (battlefield) things devices using deep eigenspace learning,” *IEEE Transactions on Sustainable Computing*, vol. 4, no. 1, pp. 88–95, 2019.

89. Stephen Russell and Tarek Abdelzaher, “The internet of battlefield things: the next generation of command, control, communications and intelligence (c3i) decision-making,” in *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)*. IEEE, 2018, pp. 737–742.

90. Paul Scharre, *Army of none: Autonomous weapons and the future of war*, WW Norton & Company, 2018.

91. K. C. Miller, BM O'Halloran, A. G. Pollman, and M. K. Feeley, "Securing the internet of battlefield things while maintaining value to the warfighter," *Journal of Information Warfare*, vol. 18, no. 2, pp. 74–84,II–III, 2019, Copyright - Copyright Peregrine Technical Solutions 2019; Last updated - 2021-04-02.

92. William Stallings, Lawrie Brown, Michael D Bauer, and Arup Kumar Bhattacharjee, *Computer security: principles and practice*, Pearson Education Upper Saddle River, NJ, USA, 2012.

93. Thomas M Chen and Jean-Marc Robert, "The evolution of viruses and worms," *Statistical methods in computer security*, vol. 1, pp. 1–16, 2004.

94. D.E. Denning, "An intrusion-detection model," *IEEE Transactions on Software Engineering*, vol. SE-13, no. 2, pp. 222–232, 1987.

95. Dorothy Denning and Peter G Neumann, *Requirements and model for IDES-a real-time intrusion-detection expert system*, vol. 8, SRI International Menlo Park, 1985.

96. Robin Sommer and Vern Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *2010 IEEE Symposium on Security and Privacy*, 2010, pp. 305–316.

97. S. J. Stolfo, Wei Fan, Wenke Lee, A. Prodromidis, and P. K. Chan, "Cost-based modeling for fraud and intrusion detection: results from the jam project," in *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*, 2000, vol. 2, pp. 130–144 vol.2.

98. Benedetto Marco Serinelli, Anastasija Collen, and Niels Alexander Nijdam, "Training guidance with kdd cup 1999 and nsl-kdd data sets of anidinr: Anomaly-based network intrusion detection system," *Procedia Computer Science*, vol. 175, pp. 560–565, 2020.

99. Arash Habibi Lashkari, Andi Fitriah A Kadir, Laya Taheri, and Ali A Ghorbani, "Toward developing a systematic approach to generate benchmark android malware datasets and classification," in *2018 International Carnahan Conference on Security Technology (ICCST)*. IEEE, 2018, pp. 1–7.

100. Hee Su Chae, Byung oh Jo, Sang Hyun Choi, and Twae kyung Park, "Feature selection for intrusion detection using nsl-kdd," *Recent advances in computer science*, vol. 20132, pp. 184–187, 2013.

101. Connor Shorten and Taghi M Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, pp. 1–48, 2019.

102. Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep Learning*, MIT Press, 2016, `http://www.deeplearningbook.org`.

103. Kathleen R Kerwin and Nathaniel D Bastian, "Stacked generalizations in imbalanced fraud data sets using resampling methods," *The Journal of Defense Modeling and Simulation*, vol. 18, no. 3, pp. 175–192, 2021.

104. Nathalie Japkowicz, "The class imbalance problem: Significance and strategies," in *Proc. of the Int'l Conf. on Artificial Intelligence*. Citeseer, 2000, vol. 56.

105. Varun Chandola, Arindam Banerjee, and Vipin Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, July 2009.

106. Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

107. Swee Kiat Lim, Yi Loo, Ngoc-Trung Tran, Ngai-Man Cheung, Gemma Roig, and Yuval Elovici, "Doping: Generative data augmentation for unsupervised anomaly detection with gan," in *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2018, pp. 1122–1127.

108. Sean M Devine and Nathaniel D Bastian, "An adversarial training based machine learning approach to malware classification under adversarial conditions," *Proceedings of the 54th Hawaii International Conference on System Sciences*, pp. 827–836, 2021.

109. Marc Chalé and Nathaniel Bastian, "Challenges and opportunities for generative methods in cyber domain," 2021.

110. Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller, "Equation of state calculations by fast computing machines," *The journal of chemical physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

111. Paul Smolensky, "Information processing in dynamical systems: Foundations of harmony theory," Tech. Rep., Colorado Univ at Boulder Dept of Computer Science, 1986.

112. Diederik P Kingma and Max Welling, "Stochastic gradient vb and the variational auto-encoder," in *Second International Conference on Learning Representations, ICLR*, 2014, vol. 19, p. 121.

113. Neha Patki, Roy Wedge, and Kalyan Veeramachaneni, "The synthetic data vault," in *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 2016, pp. 399–410.

114. Averill M. Law and W. David Kelton, *Simulation modeling and analysis 4th Edition.*, McGraw-Hill Series in Industrial Engineering and Management Science. McGraw-Hill Book Co., New York, 2007.

115. Frank J Massey Jr, "The kolmogorov-smirnov test for goodness of fit," *Journal of the American statistical Association*, vol. 46, no. 253, pp. 68–78, 1951.

116. Jon Siwek, "Zeek::http," 2021.

117. John Salvatier, Thomas V Wiecki, and Christopher Fonnesbeck, "Probabilistic programming in python using pymc3," *PeerJ Computer Science*, vol. 2, pp. e55, 2016.

118. Diederik P Kingma and Max Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

119. Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani, *An introduction to statistical learning*, vol. 112, Springer, 2013.

120. Francois Chollet, *Deep learning with Python*, Simon and Schuster, 2021.

121. Douglas C Montgomery, Elizabeth A Peck, and G Geoffrey Vining, *Introduction to Linear Regression Analysis*, vol. 821, John Wiley & Sons, 2012.

122. David R Cox, "The regression analysis of binary sequences," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 20, no. 2, pp. 215–232, 1958.

123. Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, New York, NY, USA, 1992, COLT '92, p. 144–152, Association for Computing Machinery.

124. Corinna Cortes and Vladimir Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

125. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

126. F Rosenblatt, "A bibliography of perceptron literature," *SCIENTIFIC AND TECHNICAL INFORMATION*, p. 189, 1963.

127. Minsky Marvin and A Papert Seymour, "Perceptrons," 1969.

128. David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams, "Learning Representations by Back-Propagating Errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

129. J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

130. Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone, "Classification and regression trees," *Group*, vol. 37, no. 15, pp. 237–251, 1984.

131. Jiawei Han, Micheline Kamber, and Jian Pei, *Data Mining Concepts and Techniques Third Edition*, Morgan Kaufmann, Waltham, MA, 2012.

132. Jiawei Han, Micheline Kamber, and Jian Pei, "Data Mining. Concepts and Techniques, 3rd Edition (The Morgan Kaufmann Series in Data Management Systems)," Tech. Rep., 2011.

133. Gábor J Székely and Maria L Rizzo, "Energy statistics: A class of statistics based on distances," *Journal of statistical planning and inference*, vol. 143, no. 8, pp. 1249–1272, 2013.

134. Timothy M Braje, "Advanced tools for cyber ranges," Tech. Rep., MIT Lincoln Laboratory Lexington United States, 2016.

135. Ritu Chadha, Thomas Bowen, Cho-Yu J. Chiang, Yitzchak M. Gottlieb, Alex Poylisher, Angello Sapello, Constantin Serban, Shridatt Sugrim, Gary Walther, Lisa M. Marvel, E. Allison Newcomb, and Jonathan Santos, "Cybervan: A cyber security virtual assured network testbed," in *MILCOM 2016 - 2016 IEEE Military Communications Conference*, 2016, pp. 1125–1130.

136. K Talty, J Stockdale, and N Bastian, "A sensitivity analysis of poisoning and evasion attacks in network intrusion detection system machine learning models," in *Proceedings of the 2021 IEEE Military Communications Conference*. 2021, pp. 1017–1022, IEEE.

137. D Bierbrauer, A Chang, W. Kritzer, and N Bastian, "Cybersecurity anomaly detection in adversarial environments," in *Proceedings of the AAAI Fall 2021 Symposium on AI in Government and Public Sector*, 2021, pp. 1017–1022.

138. M Schneider, D Aspinall, and N Bastian, "Evaluating model robustness to adversarial samples in network intrusion detection," in *Proceedings of the 2021 IEEE International Conference on Big Data*, 2021, to appear.

139. Alessandro Annarelli, Fabio Nonino, and Giulia Palombi, "Understanding the management of cyber resilient systems," *Computers Industrial Engineering*, vol. 149, pp. 106829, 2020.

140. V. Cerf and R. Kahn, "A protocol for packet network intercommunication," *IEEE Transactions on Communications*, vol. 22, no. 5, pp. 637–648, 1974.

141. Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization.," *ICISSp*, vol. 1, pp. 108–116, 2018.

142. Andrey Garnaev, Melike Baykal-Gursoy, and H. Vincent Poor, "How to deal with an intelligent adversary," *Computers Industrial Engineering*, vol. 90, pp. 352–360, 2015.

143. Anna L. Buczak and Erhan Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2016.

144. Animesh Patcha and Jung-Min Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Computer networks*, vol. 51, no. 12, pp. 3448–3470, 2007.

145. Monowar H Bhuyan, Dhruba Kumar Bhattacharyya, and Jugal K Kalita, "Network anomaly detection: methods, systems and tools," *IEEE communications surveys & tutorials*, vol. 16, no. 1, pp. 303–336, 2013.

146. Bryan Burns, Dave Killion, Nicolas Beauchesne, Eric Moret, Julien Sobrier, Michael Lynn, Eric Markham, Chris Iezzoni, Philippe Biondi, Jennifer Stisa Granick, et al., *Security power tools*, " O'Reilly Media, Inc.", 2007.

147. Shahbaz Rezaei and Xin Liu, "Deep learning for encrypted traffic classification: An overview," *IEEE communications magazine*, vol. 57, no. 5, pp. 76–81, 2019.

148. Hiroshi Motoda and Huan Liu, "Feature selection, extraction and construction," *Communication of IICM (Institute of Information and Computing Machinery, Taiwan)*, vol. 5, no. 67-72, pp. 2, 2002.

149. E. Hernández-Pereira, J.A. Suárez-Romero, O. Fontenla-Romero, and A. Alonso-Betanzos, "Conversion methods for symbolic features: A comparison applied to an intrusion detection problem," *Expert Systems with Applications*, vol. 36, no. 7, pp. 10612–10617, 2009.

150. Marius Kloft, Ulf Brefeld, Patrick Düessel, Christian Gehl, and Pavel Laskov, "Automatic feature selection for anomaly detection," in *Proceedings of the 1st ACM workshop on Workshop on AISec*, 2008, pp. 71–76.

151. Jonathan J. Davis, "Machine learning and feature engineering for computer network security," 2017.

152. Jonathan Peck, Joris Roels, Bart Goossens, and Yvan Saeys, "Lower bounds on the robustness to adversarial perturbations," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

153. Solomon Kullback and Richard A Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

154. Claude E Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

155. Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli, "Evasion attacks against machine learning at test time," in *Joint European conference on machine learning and knowledge discovery in databases.* Springer, 2013, pp. 387–402.

156. J.D. Tygar, "Adversarial machine learning," *IEEE Internet Computing*, vol. 15, no. 5, pp. 4–6, 2011.

157. Navin Goyal and Luis Rademacher, "Learning convex bodies is hard," in *Proceedings of the 22nd Annual Conference on Learning Theory (COLT)*, 2009, pp. 303–308.

158. Anthony D. Joseph, Blaine Nelson, Benjamin I. P. Rubinstein, and J. D. Tygar, *Near-Optimal Evasion of Classifiers*, p. 199–238, Cambridge University Press, 2019.

159. Daniel Lowd and Christopher Meek, "Adversarial learning," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, 2005, pp. 641–647.

160. Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

161. Andras Rozsa, Manuel Gunther, and Terrance E Boult, "Towards robust deep neural networks with bang," *arXiv preprint arXiv:1612.00138*, 2016.

162. Shixiang Gu and Luca Rigazio, "Towards deep neural network architectures robust to adversarial examples," in *Proceedings of the 3rd International Conference on Learning Representations*, 2015, Accepted as workshop contribution.

163. Richard M Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*, pp. 85–103. Springer, 1972.

164. Edward E Ogheneovo, "Revisiting cook-levin theorem using np-completeness and circuit-sat," *International Journal of Advanced Engineering Research and Science*, vol. 7, no. 3, 2020.

165. Diederik P. Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," 2014.

166. Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou, "Multi-class adaboost," *Statistics and its Interface*, vol. 2, no. 3, pp. 349–360, 2009.

167. Vice Admiral Kevin Scott, "Cyberspace Operations (2018) ," 2018.

168. Lin Zhu, Suryadipta Majumdar, and Chinwe Ekenna, "An invisible warfare with the internet of battlefield things: a literature review," *Human behavior and emerging technologies*, vol. 3, no. 2, pp. 255–260, 2021.

169. Norbert Wiener, "Cybernetics (1948)," *New York*, 1961.

170. Riza Azmi et al., "Revisiting cyber definition," in *European Conference on Cyber Warfare and Security.* Academic Conferences International Limited, 2019, pp. 22–30.

171. C. Stephen Carr, Stephen D. Crocker, and Vinton G. Cerf, "Host-host communication protocol in the arpa network," in *Proceedings of the May 5-7, 1970, Spring Joint Computer Conference*, New York, NY, USA, 1970, AFIPS '70 (Spring), p. 589–597, Association for Computing Machinery.

172. Gregory Benford, "Future tense: Catch me if you can," *Commun. ACM*, vol. 54, no. 3, pp. 112–ff, mar 2011.

173. Barbara Guttman and E Roback, "An introduction to computer security: the nist handbook," 1995-10-02 1995.

174. Michael J De Lucia and Chase Cotton, "Adversarial machine learning for cyber security," *Journal of Information Systems Applied Research*, vol. 12, no. 1, pp. 26, 2019.

175. Philippe Biondi, "Scapy documentation (!)," *vol*, vol. 469, pp. 155–203, 2010.

176. Adi Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, nov 1979.

177. Vik Tor Goh, Jacob Zimmermann, and Mark Looi, "Towards intrusion detection for encrypted networks," in *2009 International Conference on Availability, Reliability and Security*, 2009, pp. 540–545.

178. Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay, "Adversarial attacks and defences: A survey," *arXiv preprint arXiv:1810.00069*, 2018.

179. Marc Chalé, Bruce Cox, and Nathaniel Bastian, "Constrained optimization based adversarial example generation for transfer attacks in network intrusion detection systems," .

180. Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel, "Ensemble adversarial training: Attacks and defenses," 2017.

181. Armon Barton and III. Jatho, Edgar, "Defending against adversarial examples in deep neural network classifiers," 2021, Prepared for: NAVAIR.

182. Joaquin Vanschoren, "Meta-learning," in *Automated machine learning*, pp. 35–61. Springer, 2019.

183. Chelsea Finn, Pieter Abbeel, and Sergey Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International conference on machine learning*. PMLR, 2017, pp. 1126–1135.

184. Anna L Buczak and Erhan Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications surveys & tutorials*, vol. 18, no. 2, pp. 1153–1176, 2015.

185. Sydney Mambwe Kasongo and Yanxia Sun, "A deep learning method with wrapper based feature extraction for wireless intrusion detection system," *Computers & Security*, vol. 92, pp. 101752, 2020.

186. Herve Debar, Marc Dacier, and Andreas Wespi, "Towards a taxonomy of intrusion-detection systems," *Computer Networks*, vol. 31, pp. 805–822, 1999.

187. Michael R. Smith, Logan Mitchell, Christophe G. Giraud-Carrier, and Tony R. Martinez, "Recommending learning algorithms and their associated hyperparameters," *CoRR*, vol. abs/1407.1890, 2014.

188. Can Cui, Teresa Wu, Mengqi Hu, Jeffery D Weir, and Xianghua Chu, "Accuracy vs. Robustness: Bi-Criteria Optimized Ensemble of Metamodels," in *Proceedings of the Winter Simulation Conference 2014*. IEEE, 2014, pp. 616–627.

189. Timothy W Simpson, Jesse Peplinski, Patrick N Koch, and Janet K Allen, "On the Use of Statistics in Design and the Implications for Deterministic Computer Experiments," *Design Theory and Methodology*, vol. 14, pp. 14–17, 1997.

190. G Gary Wang and Songqing Shan, "Review of Meta-Modeling Techniques in Support of Engineering Design Optimization," *Journal of Mechanical design*, vol. 129, no. 4, pp. 370–380, 2007.

191. David H Wolpert, "The Supervised Learning No-Free-Lunch Theorems David," *Soft Computing and Industry*, , no. January 2001, 2001.

192. Can Cui, Mengqi Hu, Jeffrey D. Weir, and Teresa Wu, "A Recommendation System for Meta-modeling: A Meta-learning Based Approach," *Expert Systems With Applications*, vol. 46, pp. 33–44, 2016.

193. S Revathi and A Malathi, "A detailed analysis on nsl-kdd dataset using various machine learning techniques for intrusion detection," *International Journal of Engineering Research & Technology (IJERT)*, vol. 2, no. 12, pp. 1848–1853, 2013.

194. Carl E Landwehr, Alan R Bull, John P McDermott, and William S Choi, "A taxonomy of computer program security flaws," *ACM Computing Surveys (CSUR)*, vol. 26, no. 3, pp. 211–254, 1994.

195. Michael Sobirey, "Michael sobirey's intrusion detection systems page," 2000.

196. Swati Paliwal and Ravindra Gupta, "Denial-of-service, probing & remote to user (r2l) attack detection using genetic algorithm," *International Journal of Computer Applications*, vol. 60, no. 19, pp. 57–62, 2012.

197. E. Viegas, A. O. Santin, A. França, R. Jasinski, V. A. Pedroni, and L. S. Oliveira, "Towards an energy-efficient anomaly-based intrusion detection engine for embedded systems," *IEEE Transactions on Computers*, vol. 66, no. 1, pp. 163–177, 2017.

198. John Rice, "The Algorithm Selection Problem - Abstract Models," pp. 75–152, 1974.

199. Paul E Utgoff, "Shift of bias for inductive concept learning," *Machine learning: An artificial intelligence approach*, vol. 2, pp. 107–148, 1986.

200. Larry A Rendell, Raj Sheshu, and David K Tcheng, "Layered Concept-Learning and Dynamically Variable Bias Management," in *IJCAI*, 1987, pp. 308–314.

201. Pavel Brazdil, Joāo Gama, and Bob Henery, "Characterizing the Applicability of Classification Algorithms Using Meta-Level Learning," in *European Conference on Machine Learning*. 1994, pp. 83–102, Springer.

202. Michael R. Smith, Logan Mitchell, Christophe Giraud-Carrier, and Tony Martinez, "Recommending learning algorithms and their associated hyperparameters," in *CEUR Workshop Proceedings*. 2014, vol. 1201, pp. 39–40, CEUR-WS.

203. Clarence O. Williams, "Meta Learning Recommendation System for Classification," M.S. thesis, Air Force Institute of Technology, 2020.

204. James J Cochran, *INFORMS Analytics Body of Knowledge*, John Wiley & Sons, 2018.

205. Andras Janosi, William Steinbrunn, Matthias Pfisterer, and Robert Detrano, "Heart data set," 2018.

206. Aman Ajmera, "Framingham data set," 2019.

207. Vincent Arel-Bundock, “Spam data set,” 2019.

208. Sunil Jacob, “Personal loan data set,” 2018.

209. William H. Wolberg, W. Nick Street, and Olvi L. Mangasarian, “Breast cancer data set,” 1995.

210. Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A. Ghorbani, “A detailed analysis of the kdd cup 99 data set,” 2009.

REPORT DOCUMENTATION PAGE

1. REPORT DATE 15-09-2022	**2. REPORT TYPE** Doctoral Dissertation	**3. DATES COVERED** **START DATE** Sept 2019	**END DATE** Sept 2022

4. TITLE AND SUBTITLE
Meta Learning Recommendation System for Classification

5a. CONTRACT NUMBER	**5b. GRANT NUMBER**	**5c. PROGRAM ELEMENT NUMBER**
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**

6. AUTHOR(S)
Chalé. Capt, U.S. Air Force

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way, Building 640 WPAFB OH 45433-8865	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT-ENS-MS-22-S-056

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Intentionally Left Blank	**10. SPONSOR/MONITOR'S ACRONYM(S)**	**11. SPONSOR/MONITOR'S REPORT NUMBER(S)**

12. DISTRIBUTION/AVAILABILITY STATEMENT

DISTRUBTION STATEMENT A. APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES
This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT
Cyberspace is the digital communications network that that supports the internet of battlefield things (IoBT), the model by which defense-centric sensors, computers, actuators and humans are digitally connected. A secure IoBT infrastructure facilitates real time implementation of the observe, orient, decide, act (OODA) loop across distributed subsystems. Successful hacking efforts by cyber criminals and strategic adversaries suggest that cyber systems such as the IoBT are not secure. Three lines of effort demonstrate a path towards a more robust IoBT. First, a baseline data set of enterprise cyber network traffic was collected and modelled with generative methods allowing the generation of realistic, synthetic cyber data. Next adversarial examples of cyber packets were algorithmically crafted to fool network intrusion detection systems while maintaining packet functionality. Finally, a framework is presented that uses meta-learning to combine the predictive power of various weak models. This resulted in a meta-model that outperforms all baseline classifiers with respect to overall accuracy of packets, and adversarial example detection rate. The National Defense Strategy underscores cybersecurity as an imperative to defend the homeland and maintain a military advantage in the information age. This research provides both academic perspective and applied techniques to to further the cybersecurity posture of the Department of Defense into the information age.

15. SUBJECT TERMS

Generative Methods and Meta Learning for Cyber Security

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**
a. REPORT U	**b. ABSTRACT** U	**C. THIS PAGE** U	UU	238

19a. NAME OF RESPONSIBLE PERSON Dr. Jeffery D. Weir, Ph.D., AFIT/ENS	**19b. PHONE NUMBER** *(Include area code)* (937)255-6565, ext 4523 Jeffrey.Weir@afit.edu

PREVIOUS EDITION IS OBSOLETE.

STANDARD FORM 298 (REV. 5/2020)
Prescribed by ANSI Std. Z39.18