International Journal of Scientific Research and Engineering Development-– Volume 8 Issue 6, Nov-Dec 2025
Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

Offensive and Defensive Dynamics of Adversarial AI in Cybersecurity

Rosemary Chisom Dimakunne*[1], Paul Clement Uwamotobon Akpabio[2], Monsuru Olarewaju Moshood[3]

[1] Department of Management Information Systems, Baylor University, Texas, USA.
ORCID: 0009-0003-4629-4593
[2] College of Science Engineering and Technology, Texas Southern University. Texas, USA.
ORCID:0009-0009-1932-1975
[3] Department of Mathematics, Ball State University. Indiana, USA.
ORCID:0009-0001-8533-1989
Corresponding Author: Rosemary Chisom Dimakunne

Abstract

This study provides a dual perspective analysis of adversarial AI in cybersecurity, examining both attack (offensive) and defense (protective) strategies. The motivation stems from the growing adoption of machine learning (ML) based intrusion detection, malware classification, and other security systems, and the emergent threat of *adversarial attacks* that can degrade or evade those systems. We specifically investigate *poisoning attacks* (which corrupt training data or models) and *evasion attacks* (which add subtle perturbations to inputs at inference) and their impacts on popular ML models. The research combines empirical experiments with theoretical analysis to explore how such attacks succeed and how defenses perform under attack. We implemented representative adversarial attack algorithms on benchmark cybersecurity datasets. For evasion attacks, we used gradient based methods (e.g., FGSM and PGD) to craft small input perturbations that cause misclassification[1]. For poisoning attacks, we applied data injection and label flipping techniques (including a Generative Adversarial Network to generate malicious training samples) to subtly shift the training distribution and model parameters[2][3]. Experiments were conducted on standard intrusion detection datasets (including CICIDS2017 and UNSW-NB15) with classifiers such as logistic regression, decision tree, random forest, gradient boosting, and a deep neural network. Defensive methods evaluated include **adversarial training** (training on adversarial examples)[4], **ensemble modeling** (combining multiple classifiers)[5], **feature squeezing** based input filtering[6], as well as data sanitization (removing or down weighting suspect training data) and robust statistical techniques[7]. Performance was measured in terms of accuracy, precision, recall, F₁-score, detection rates for adversarial inputs, and the robustness (accuracy on adversarially perturbed data) versus clean data accuracy trade off. The offensive experiments demonstrate that poisoning attacks can significantly shift model decision boundaries, leading to degraded training accuracy and biased models[8]. Even injecting a small fraction of poisoned data caused notable drops in detection accuracy (up to 20–30% in our tests, depending on the model) as the model's learning was skewed. Evasion attacks were able to generate imperceptible input perturbations (e.g., modifying network traffic features by only a few percent) that caused high misclassification rates[1]. For example, a fast gradient sign method attack at an ℓ_∞ perturbation of $\epsilon=0.05$ reduced an intrusion detector's accuracy by over 40% (from ~95% to ~55%). On the defensive side, **adversarial training** substantially improved model robustness to evasion attacks, models trained on adversarial examples recovered much of the lost accuracy on perturbed inputs[4]. However, this came at the cost of slightly lower accuracy on normal (unperturbed) data (confirming the known trade off that robustness increases while clean accuracy drops)[9]. **Ensemble models** outperformed individual models in both accuracy and robustness: by forcing attackers to evade multiple classifiers simultaneously,

ensembles made successful evasion significantly harder[5]. An ensemble intrusion detector (e.g., combining a decision tree, a k-NN, and an SVM) maintained higher accuracy under attack than any single model alone. The **feature squeezing** detection mechanism proved effective at flagging adversarial inputs by comparing a model's predictions on original vs. "squeezed" inputs (reduced feature precision), we detected a majority of adversarial samples with few false alarms (consistent with prior work achieving high detection rates)[6]. Data sanitization (removing outliers or training points with high negative impact) and robust statistics (regularization) also improved resistance to poisoning attacks, though their protection was limited against adaptive attackers[7]. This dual perspective analysis contributes to the cybersecurity and adversarial machine learning literature in several ways. (1) We present a **taxonomy guided analysis** of attack types (data poisoning vs. evasion) using standard terminology from NIST's adversarial ML taxonomy[10], helping to clarify the threat landscape in cybersecurity contexts. (2) We implement multiple attack algorithms (including label flipping and data injection poisoning, as well as gradient based evasion attacks like FGSM/PGD and a GAN based attack) on real network intrusion detection datasets, quantifying their impact on different ML model types. (3) We evaluate a comprehensive suite of defense strategies from data sanitization and robust training stats[7], to adversarial training[4], ensemble learning[5], defensive distillation, and feature squeezing detection[6] analyzing their effectiveness and trade offs (e.g., robustness vs. clean accuracy, computational cost)[11][12]. (4) We distill practical **guidelines** for deploying robust ML models in cybersecurity: recommendations include incorporating adversarial examples into training, monitoring model inputs and outputs for signs of attack (e.g., using ensemble disagreement or feature squeezers as detectors), applying data provenance and sanitization checks to training data, and using layered defenses for defense in depth. By combining offensive and defensive insights, our work illuminates how adversaries can undermine ML based security systems and how defenders can strengthen them, contributing toward more resilient cybersecurity solutions.

Keywords: Adversarial Machine Learning; Cybersecurity; Evasion Attacks; Data Poisoning; Adversarial Training; Ensemble Learning; Intrusion Detection; Robustness; Feature Squeezing; Machine Learning Security

1 Introduction

1.1 Context and Motivation

Machine learning has become integral to many cybersecurity tasks, including network intrusion detection, malware classification, spam filtering, and anomaly detection. Advanced ML models from classical algorithms like logistic regression and decision trees to modern deep neural networks now analyze network traffic, files, and user behaviors to identify threats. These learning based systems often outperform signature based methods by generalizing to novel attacks. However, **attackers are actively finding ways to exploit vulnerabilities in ML models**. In particular, adversaries can supply specially crafted inputs or training data that take advantage of the model's learned patterns, causing errors in prediction. For example, attackers may **corrupt the training data (poisoning)** by injecting malicious instances or altering labels so that the model learns a skewed decision boundary. Alternatively, they may add **small perturbations to inputs at inference time (evasion)** that are virtually imperceptible but lead the model to misclassify the input in the attacker's favor[1][13]. In the realm of cybersecurity, such tactics have serious consequences. A poisoned intrusion detection system (IDS) might learn to ignore certain malicious traffic, and an evasion attack could allow malware or malicious network packets to slip through undetected. Real world incidents underscore these risks for instance, attackers have demonstrated that subtle changes to malware files can evade ML based malware detectors, and carefully crafted network packets can fool an ML based IDS into **misclassifying malicious traffic as benign** (thus suppressing critical alerts)[13]. The result is that security breaches go undetected or unmitigated, defeating the very purpose of intelligent defense systems. As the use of AI/ML expands in security operations, understanding and countering these adversarial exploits has become paramount (NIST, 2019[10]).

1.2 Problem Statement

While a significant body of research exists either on attacking ML models or on defending them, **comprehensive studies addressing both offensive and defensive dynamics in tandem remain limited**. Most prior works analyze either how attacks succeed or how a particular defense improves robustness, but not both in an integrated manner. This leaves a knowledge gap in understanding how attacks and defenses interact. In cybersecurity deployments, defenders must anticipate not only how an adversary might compromise an ML model (to evade detection or disrupt learning) but also which countermeasures are most effective against those specific attacks. **The goal of this paper is to systematically evaluate the dual dynamics of adversarial AI in cybersecurity from the attacker's techniques (poisoning training data, evading detection at test time) to the defender's arsenal (data sanitization, adversarial training, ensembles, detection methods)**. We aim to quantify how different attack methods impact popular ML models used in cybersecurity (e.g., classifiers for intrusion detection) and then assess multiple defense strategies under those attack conditions. By studying attacks and defenses side by side on the same problems, we seek to identify which defenses truly offer robust protection and what trade offs they entail. In summary, our problem statement is: *How do poisoning and evasion attacks affect the performance of ML based cybersecurity systems, and how well can various defenses mitigate these attacks?* Addressing this requires a holistic experimental approach that is currently lacking in the literature.

1.3 Contributions

To tackle the above problem, this work makes several specific contributions to both the research community and practitioners:

- **Taxonomy Guided Attack Analysis:** We categorize adversarial attack types affecting ML security models using standard terminology. Following the NIST adversarial ML taxonomy (Tabassi *et al*., 2019)[10], we distinguish *training time attacks* (a.k.a. **poisoning** or causative attacks) from *inference time attacks* (a.k.a. **evasion** or exploratory attacks). We further break down poisoning into **data injection**, **data manipulation** (label or feature tampering), and **logic corruption** of the model[2][3]. This taxonomy driven perspective ensures comprehensive coverage of attack scenarios and provides a common language for discussing threats in cybersecurity ML systems.

- **Implementation of Multiple Attack Algorithms:** We developed and deployed a suite of adversarial attack algorithms on benchmark cybersecurity datasets. This includes **labelflipping poisoning** (randomly or strategically flipping a portion of class labels in the training data), **data injection poisoning** (introducing specially crafted malicious samples into training to shift the decision boundary)[14], and a **gradient ascent poisoning** method (optimizing injected samples to maximize the model's error[8]). For evasion attacks, we implemented gradient based techniques such as the **Fast Gradient Sign Method (FGSM)** and **Projected Gradient Descent (PGD)** to generate adversarial examples at test time, as well as the **Jacobian based Saliency Map Attack (JSMA)** for a more fine grained iterative attack[1]. We also explored black box evasion using transfer attacks (where an attacker trains a substitute model to craft attacks) in line with prior research. All attack implementations were evaluated on real network intrusion detection datasets (detailed in Section 3.2) to gauge their effectiveness against different ML classifier types.

- **Evaluation of Defense Strategies and Trade offs:** On the defensive side, we implemented several countermeasures and evaluated their efficacy against the above attacks. These include:

- *Data Sanitization* and outlier removal during training (often called "reject on negative impact"), wherein we attempted to detect and remove suspicious training instances that could be poisoned[7].

- *Robust Statistics* and regularization methods to lessen the influence of any one training point (e.g., using ridge regularization to reduce model variance due to poisoned data)[15].
- *Adversarial Training*, injecting FGSM/PGD adversarial examples into model training to make the learned model more robust to attacks[4]. We analyze how the proportion of adversarial samples and the perturbation budget affect the robustness vs. accuracy trade off (following recommendations from prior work to start with small perturbations and gradually increase[16]).
- *Ensemble Learning*, where we construct ensembles of heterogeneous classifiers (e.g., combining a neural network with a random forest and an SVM) to force attackers to overcome multiple decision boundaries[5]. We test both bagging and boosting ensembles and measure improvements in detection and robustness.
- *Defensive Distillation* and gradient masking techniques (training a model in a way that "flattens" gradients[17] or otherwise hides gradient information to thwart gradient based attacks) we evaluate the degree to which our models with defensive distillation resist FGSM/PGD attacks compared to normal models.
- *Feature Squeezing and Input Transformations*, applying simple input preprocessing (like reducing feature precision or adding noise) at test time and checking model consistency[6]. We implemented bit depth reduction on input features and median smoothing on feature vectors (analogous to image smoothing, but for numerical network features) to detect evasion attempts, per the method of Xu *et al.* (2018)[18].
- Other mitigations such as *differential privacy* during training (to noise the training process and reduce information leakage) and *homomorphic encryption* for model inputs (to prevent an attacker from reading or meaningfully modifying data) were conceptually considered[19], although not fully implemented due to their complexity and performance cost. We discuss these in the context of future work and their trade offs (noting that they can greatly increase computation and may degrade model accuracy in practice).

For each defense, we analyzed **trade offs**: e.g., adversarial training typically increases robustness substantially but can reduce clean data accuracy and increase training time[9][11]; ensembles improve robustness but add computational overhead; feature squeezing adds an extra detection layer but could flag some normal inputs (false positives). Our evaluation provides a comparative analysis to inform which defenses (or combination thereof) offer the best protection under various attack scenarios, and at what cost.

- **Practical Guidelines for Robust ML Deployment:** Drawing from our findings, we outline practical recommendations for deploying ML models in adversarial environments (cyber networks, enterprise systems, etc.). These guidelines address data management (e.g., ensuring continuous monitoring and validation of incoming training data to catch poison attempts[20]), model hardening (e.g., regularly retraining with a mixture of adversarial examples to "vaccinate" the model[21], using simpler or regularized models for critical decisions to reduce attack surface[5]), and runtime detection (e.g., leveraging ensemble disagreements or anomaly detectors on model inputs to catch potential evasion in real time[22]). We also stress the importance of evaluating models against adaptive threats as a standard practice e.g., running red team adversarial attacks on one's own models (as part of the development lifecycle) to identify weaknesses, similar to how penetration testing is done for traditional systems. Overall, our contributions aim to bridge the gap between theory and practice by providing both empirical evidence and actionable insight into building ML systems that can withstand adversarial interference.

1.4 Organization

The remainder of this paper is organized as follows. **Section 2** reviews background and related work, covering the use of ML in cybersecurity, a taxonomy of adversarial attacks (poisoning vs. evasion), common attack techniques, and defensive countermeasures from the literature. **Section 3** details our research methodology,

including research questions, datasets used, ML models evaluated, attack implementations, defense implementations, and evaluation metrics and procedures. **Section 4** presents the experimental results and analysis: we quantify the impact of poisoning attacks (Section 4.1) and evasion attacks (4.2) on model performance, then assess the effectiveness of various defenses (4.3) and provide a comparative analysis of their trade offs (4.4). **Section 5** provides a discussion of the findings interpreting why certain models or defenses behaved as they did, implications for deploying ML in real world security, and limitations of our study with directions for future research (including emerging ideas like game theoretic modeling of attacker defender interactions). **Section 6** concludes the paper by summarizing how our dual perspective approach advances the understanding of adversarial AI in cybersecurity and recommending steps toward more robust systems. We also include **Appendices** with additional details: Appendix A gives pseudo code for our attack algorithms and data sanitization procedure; Appendix B lists hyper parameters and model configurations; Appendix C provides extended results (e.g., full confusion matrices, ROC curves, results at varying attack strengths) and notes on statistical significance of differences.

2 Related Work and Background

2.1 Machine Learning in Cybersecurity

ML Applications: In the cybersecurity domain, machine learning techniques are widely used to detect and classify threats. For example, **network intrusion detection systems (IDS)** often employ classifiers (such as decision trees, random forests, support vector machines, or neural networks) to distinguish malicious traffic from benign traffic. Malware detection engines use ML models to classify files or processes as malicious or benign based on behavioral or static features. Email spam filters, insider threat detection, and fraudulent transaction detection systems all increasingly rely on ML for their predictive capabilities. Traditional algorithms like logistic regression and decision trees provide interpretable decision boundaries for these tasks, while more complex models like ensemble methods and deep neural networks can capture non linear relationships and often yield higher detection accuracy in practice. Indeed, in intrusion detection benchmarks, deep learning models and boosted ensembles have been shown to outperform signature based or simpler ML models in terms of detection rates and false positive rates (e.g., in classifying novel attack patterns) they can automatically learn intricate feature interactions that humans might not identify.

Adversarial Vulnerabilities: However, *all these ML models share a common vulnerability: they can be fooled or subverted by adversarially crafted inputs*. Extensive research in adversarial machine learning has revealed that even high performing models can exhibit brittleness. For instance, **deep neural networks**, despite their accuracy, are particularly susceptible to adversarial examples due to their complex, high dimensional decision boundaries[23]. Even a tiny perturbation to an input (for example, modifying just a few bytes or network packet features) can send a neural network into a completely different classification outcome[24][25]. Simpler models like linear classifiers (e.g., logistic regression) can sometimes be more resistant to certain perturbations due to their more constrained decision surface[26], but they are by no means immune, attackers can often find a way to manipulate input features that exploit the model's linear weights. In fact, successful adversarial attacks have been demonstrated against virtually every major type of ML model, including deep networks, tree based ensembles, SVMs, and k-NN[27]. Biggio and Roli (2018) and others chronicle how even clustering and anomaly detection models can be evaded by carefully crafted outliers. The implication is clear: **having a high accuracy on benign test data does not guarantee security against a savvy adversary**. As ML becomes more embedded in security systems, researchers have identified numerous ways these systems could be attacked, from changing just one pixel in an image to fool vision models, to adding no op code into malware to evade malware classifiers[28], to slightly reordering network events to confuse IDS models. This arms race between attackers and defenders in the ML context forms the basis of adversarial AI in cybersecurity.

2.2 Taxonomy of Adversarial Attacks

According to the NIST Adversarial ML taxonomy (Tabassi *et al.*, 2019) and other surveys[10][29], adversarial attacks on ML can be broadly divided into two categories based on the stage of the machine learning pipeline they target:

- **Poisoning Attacks (Causative Attacks):** These attacks occur during the *training phase* of the ML model. The adversary's goal is to inject or influence the training data (or model parameters) in such a way that the resulting model is flawed for example, it might misclassify certain inputs or have overall degraded accuracy. Poisoning can be *indirect* (if the attacker can only influence raw data before feature processing) or *direct* (if they can modify the processed training data or the model itself)[30]. In indirect poisoning, an attacker might, for instance, manipulate network traffic in a live environment so that when it's collected for training an IDS, it contains malicious patterns mislabeled as benign. In direct poisoning, common strategies include adding malicious samples with incorrect labels (data injection), altering labels or features of existing training points (label or feature manipulation), or even tampering with the learning algorithm's logic or parameters (logic corruption) if the attacker has that level of access[31][3]. The effect of poisoning is a shift in the decision boundaries learned by the model[8] for example, the model might carve out a "hole" in the feature space where malicious traffic is always classified as normal, because the training data was poisoned to create that blind spot. Poisoning attacks are also known as *causative* attacks because they directly cause the model to learn incorrect mappings.

- **Evasion Attacks (Exploratory Attacks):** These attacks occur during the *inference (testing) phase*, after the model has been trained and deployed[10]. The attacker does not change the model or training data, but instead **carefully crafts input samples that the model will misclassify**, all while appearing innocuous to human observers. In an evasion attack, the adversary effectively solves an optimization problem: find the smallest perturbation δ to a given input x such that the model's prediction changes (ideally to a target class if the attacker has a specific goal). This typically involves computing the model's gradient with respect to the input features to see which small changes would most increase the model's loss[1]. Evasion attacks are often *white box* (the attacker knows the model's architecture and parameters, thus can compute exact gradients) or *black box* (the attacker only queries the model and may use techniques like finite differences or substitute models to approximate the gradient)[32]. Either way, the attacker exploits the model's learned decision boundary, looking for the nearest decision boundary crossing from a malicious sample into the benign region. Because the model itself isn't altered, evasion is also called an *exploratory* attack, the adversary is "exploring" the model's weaknesses through inputs. A successful evasion means, for example, that a malicious network packet with slight modifications (perhaps in header values or timing) is classified as normal by an IDS, escaping detection. More complex *oracle attacks*, such as model extraction or model inversion, can be seen as supporting attacks: for instance, by extracting a copy of the defense model through repeated queries (an oracle attack), an adversary can then craft stronger evasion attacks offline against the stolen model[32]. In summary, evasion attacks allow adversaries to bypass defenses at runtime without needing to tamper with the training process.

These two categories (poisoning vs. evasion) align with the classic security concepts of compromising the **integrity of the model** (poisoning the training to induce incorrect behavior) versus **avoiding detection at time of use** (evasion of the classifier's decisions). Some adversarial tactics don't fit neatly into these (for example, *availability* attacks may aim to overload or crash an ML system, and *privacy* attacks aim to extract information rather than misclassify), but for this paper, we focus on the mainstream concerns of poisoning and evasion as they pertain to cybersecurity ML systems.

2.3 Adversarial Attack Techniques

Data Poisoning Strategies: Within poisoning attacks, there are several techniques by which an adversary can corrupt the learning process:

- *Label Flipping Attacks:* The attacker selects some fraction of the training data and flips their labels for example, marking some malicious traffic samples as "benign" (or vice versa) in the training set. This causes the model to learn a decision boundary that is incorrect for those regions of feature space. A simple label flip attack may choose high value malicious samples (those that are most important to detect) and relabel them as normal in training. The result is the model will not flag that type of malicious traffic at test time because it was trained to think it is normal. Prior studies have shown that even flipping a small percentage of labels (e.g., 5–10%) can significantly degrade classifier accuracy, especially for models like logistic regression that try to satisfy all training points[33].

- *Data Injection Attacks:* Here the attacker injects new training samples, rather than modifying existing ones. For instance, they could create synthetic network traffic records that are engineered to look normal (so as not to be filtered by any preprocessing) but are actually malicious or mislabeled. These injected points can be optimized to maximally pull the decision boundary towards themselves. One approach is using **gradient ascent on the training loss**: the attacker iteratively adjusts a candidate poison sample in the feature space to maximize the model's error on a set of validation points[8]. Another approach uses influence functions or Shapley values to pick points that will most skew the model. In an unsupervised context (like clustering based anomaly detectors), injection can shift centroids or density estimates[8]. *Backdoor attacks* are a special case of injection poisoning where the attacker plants a pattern in the training data (e.g., a specific byte sequence in malware, or a particular packet payload in traffic) that the model learns to associate with the wrong label. The model performs well on normal data, but whenever the attacker inputs data containing that secret "trigger" pattern, the model misclassifies it to a target class (like always labeling it benign). Backdoor poisoning has been demonstrated in domains like image recognition and could be analogously applied to network traffic data.

- *Feature Manipulation / Logic Corruption:* Instead of adding new samples, an attacker with higher access could directly manipulate feature values of existing training samples or even tamper with the training algorithm. For example, they might modify certain features of malware samples in the training set (like change all instances of a certain API call count to distort feature distribution) this *input manipulation* can similarly mislead the model's learning[3]. *Logic corruption* refers to actually altering the model or learning process, say by hacking the training code to introduce a bias or modifying the loss function to include a malicious term. This is less common in practice (requires compromising the training pipeline), but if achieved, it could be very powerful (the adversary essentially trains the model to their will). In summary, poisoning attacks, whether through label flips, injected data, or direct interference, aim to **degrade the model's learning**. Empirical case studies (e.g., the one by Alshahrani *et al.*, 2022) have shown that poisoning can reduce a classifier's accuracy significantly; in their intrusion detection experiments, a poisoning attack caused a clear drop in training accuracy and the logistic regression model was more severely impacted than a decision tree[34]. This difference may be because the tree can isolate some clean data in separate branches, whereas the linear model's single decision boundary is more broadly shifted by the poisoned data.

Evasion and Gradient Based Attacks: The primary evasion attacks used in research are optimization based, finding minimal perturbations to inputs:

- *Fast Gradient Sign Method (FGSM):* Introduced by Goodfellow *et al.* (2015), FGSM is a one step attack that perturbs each feature of the input in the direction of the gradient of the loss w.r.t. that feature[35].

For a given input x and true label y, FGSM computes $\eta = \epsilon \cdot \text{sign}(\nabla_x L(x,y))$ and sets the adversarial example $x' = x + \eta$. Essentially, this moves x slightly in the direction that increases the model's loss the most. ϵ is a chosen small magnitude. FGSM is fast to compute (requires only one gradient calculation) and often effective in causing misclassification[36], though it is a relatively coarse perturbation.

- *Projected Gradient Descent (PGD):* PGD is an iterative multi step extension of FGSM (also known as the basic iterative method or Madry's attack). It applies multiple small FGSM steps, each time projecting the adversarial example back into an allowed perturbation norm ball (e.g., ensuring the total perturbation stays within an ϵ bound). PGD is considered a strong first order attack and often finds the optimal (worst case) adversarial example within a norm constraint. It is computationally heavier (requires multiple iterations of gradient computation), but typically more successful than single step attacks, especially against models that are robust to single step attacks[4]. In our work, we use PGD with a modest number of iterations as one of the attack methods to test model robustness.

- *Jacobian based Saliency Map Attack (JSMA):* This method (Papernot *et al.*, 2016) selects features to perturb using the model's Jacobian (gradient) to find those that most influence the target class probability[37]. JSMA perturbs features one or a few at a time (iteratively), which allows it to craft adversarial examples that might change only a few features significantly (keeping the rest of the features unchanged). It tends to produce adversarial inputs that *semantically* look more similar to the original, at the cost of more computation. In security settings, JSMA could correspond to altering just a few specific packet fields or header values that the model heavily relies on.

- *Black Box Attacks and Transferability:* If the attacker does not have access to model internals, **transfer attacks** can be used. The adversary trains (or uses) a surrogate model on their end to mimic the target model's behavior, then generates adversarial inputs for the surrogate (often using FGSM/PGD), and finally submits those inputs to the target system. Due to the phenomenon of *transferability*, adversarial examples often remain effective across different models, especially if the models are of similar type or decision function[26]. In our study, we consider transfer based evasion by training a substitute IDS model and crafting attacks against it, observing partial success against the black box target model.

Additionally, there are *gradient free attacks* (like NES or SPSA) that approximate gradients via queries, and *decision based attacks* that only use the final model decision to navigate towards misclassification, these are relevant if the defender implements gradient masking. However, these usually require many model queries to succeed[32]. We primarily focus on the more efficient gradient based attacks under white box assumptions, as they represent a worst case scenario for defenders. It's worth noting that evasion attacks can be *targeted* (misclassify an input into a specific class) or *untargeted* (cause a misclassification into any class different from the true one). In the intrusion detection context, the target is usually the benign class, attackers want malicious instances to be classified as normal.

Case Study PLOS One IDS Attacks: To illustrate, Alshahrani *et al.* (2022) performed both evasion and poisoning attacks on IDS models (logistic regression and decision tree) using the CICIDS2017 dataset. They found that evasion attacks (using a GAN to generate adversarial network traffic) **reduced the testing accuracy** of both IDS models, with the decision tree's accuracy dropping more (i.e., the tree was more susceptible to evasion than the logistic model)[34]. In contrast, their poisoning scenario (injecting a percentage of label flipped GAN generated samples into training) **disrupted the logistic regression's training process more than the decision tree's**[34]. These results align with our discussion: the tree, having more discrete splits, was easier to evade with small perturbations (hitting those split conditions), whereas the logistic model, being global, was

more drastically skewed by poisoned data. We will refer to some of their specific results in Section 4 to compare with our findings.

2.4 Defensive Countermeasures

A number of defenses have been proposed to protect ML models against adversarial attacks. Here we summarize the main categories of defenses relevant to cybersecurity applications:

Data Sanitization and Robust Statistics: These defenses operate on the training data *before* model learning, aiming to detect or mitigate poisoned points. *Data sanitization* techniques attempt to **filter out or down weight suspicious training samples** that could be adversarial[38]. For instance, an IDS might examine new training logs and remove any that cause a large drop in cross validation accuracy (the "reject on negative impact" approach)[39]. Outlier detection can also be employed: if a network traffic sample lies far from the distribution of known benign or malicious samples in feature space, it might be an injected point and thus excluded from training. *Robust statistical methods* incorporate defenses into the model fitting itself, examples include; using robust loss functions (less sensitive to outliers), or regularization that limits the model's capacity to memorize peculiar points[40]. By **constraining the model** (e.g., via ridge or lasso regularization, or by using bounded influence estimators), we can reduce the effect of any single poisoned sample[7]. A simple robust defense is to median transform features or apply RANSAC style training to ignore outliers. These defenses can be seen as "vaccinating" the training process against some poisoning, though a clever poison attack that mimics normal data closely can still slip through. Nonetheless, data sanitization is a practical first line of defense, especially when training data comes from untrusted sources.

Adversarial Training: This is arguably the most prominent defense against evasion attacks. The idea is straightforward: *train the model on adversarial examples so that it learns to classify them correctly*. In practice, one augments each mini batch of training data with some adversarially perturbed examples (often generated by FGSM or PGD attacks on the current model). Over many iterations, the model's decision boundaries adjust to incorporate these perturbations, making it harder for similar attacks to fool it[4]. Madry *et al.* (2018) demonstrated that training on strong multi step attacks (PGD) yields models that are empirically robust to a wide range of attacks up to the perturbation magnitude used in training. In our context, we performed adversarial training by generating adversarial network traffic samples (e.g., slightly modified malicious traffic that originally fooled the model) and including them in the IDS training set, with correct labels. **Trade offs:** Adversarial training tends to increase robustness significantly for example, our adversarially trained model's accuracy on FGSM attacks improved from ~50% to ~80% in one experiment. However, it often **reduces accuracy on clean data** because the model devotes capacity to fitting adversarial patterns that might conflict slightly with optimizing clean performance[9][11]. In one study on malware classifiers, adversarial training reduced clean accuracy by a few percentage points while doubling the effort an attacker needed to evade it[41]. It also *increases training time* substantially, by a factor of 2–5× or more, since generating adversarial examples (especially multi step PGD) is costly[12]. Despite this, adversarial training is considered a necessary component for any model that might face evasion attempts. Researchers have extended it with variants like **ensemble adversarial training** (training on attacks transferred from other models) to further strengthen robustness. One should note that adversarial training primarily defends against the specific attack type and magnitude it trained on adaptive or larger attacks may still succeed, and there is no absolute guarantee of security (analogous to vaccination providing immunity only to certain strains)[21].

Ensemble Methods: Using an ensemble of multiple models is a defense that leverages **diversity to improve robustness**. The intuition is that an adversary might find it easy to fool one model, but **fooling several models at once is much harder**, especially if they are of different types or trained on different subsets of data[5]. Ensemble defenses can be implemented by running multiple classifiers in parallel and aggregating their outputs (e.g., majority vote or an average of predictions). In an IDS, you might have a neural network, a decision tree,

and a k-NN all analyze the same traffic; an alert is raised only if the majority vote classifies as malicious. This way, an attacker would need to craft an input that simultaneously lies in what each model considers benign space, a significantly more complex task than exploiting one decision boundary. Prior work has indeed found that ensembles often yield higher **detection accuracy** under normal conditions and can be more **robust to adversarial input**[42]. For example, an ensemble IDS (Random Forest + SVM) maintained accuracy against certain evasion attacks that completely fooled each individual model alone. There is also evidence that encouraging diversity among ensemble members (e.g., by training on different feature subsets or using different architectures) improves adversarial resilience (Abbasi *et al.*, 2020). However, ensembles do not make attacks impossible; in some cases, an attack found via a gradient on a *fused ensemble output* can still simultaneously trick all members. Adaptive attacks can also target the "weakest link" model in the ensemble or exploit correlation in errors. Another downside is that ensembles are heavier to run (more models to evaluate) and perhaps harder to maintain. Despite this, the ensemble approach is quite practical: many production security solutions use ensembles to reduce false positives, and as a bonus they get some adversarial robustness[5]. SentinelOne (2022) explicitly notes that ensemble methods force an adversary to "fool multiple decision boundaries simultaneously," raising the bar for attacks[5]. In our evaluation, we will see that an ensemble of simple models can outperform a complex single model in adversarial settings.

Defensive Distillation and Gradient Masking: Defensive distillation (Papernot *et al*., 2016) was one of the early proposed defenses for neural networks. In distillation, one trains a "student" network on the softened outputs (probabilistic predictions) of the original network (the "teacher"), using a higher temperature in the softmax to produce less confident (more smooth) predictions. The result is a new network that has a **smoother decision surface**, making it harder for small perturbations to cause big changes in output[17]. Essentially, defensive distillation aims to remove gradients that an attacker could exploit by training the student to only preserve the robust features of the teacher. While it initially showed some success in reducing the effectiveness of simple attacks, later research found it can be circumvented by adaptive attacks, and it may fall under "security by obscurity" (gradient masking). **Gradient masking** refers to any technique that makes the model's gradients less useful for attackers, this could be by saturating activations at extremes (so gradients vanish) or by adding randomness/noise during inference. These methods sometimes give a false sense of security; often, if an attacker uses a gradient free approach or increases perturbation, they can still succeed[43]. Nonetheless, in scenarios where the adversary is expected to use off the shelf gradient attacks, defensive distillation can act as a hurdle. We include distillation as a part of our discussion (training a distilled version of a DNN IDS) and observed that it indeed lowers the magnitude of gradients but did not fully stop strong attacks. NIST's taxonomy lists gradient masking and defensive distillation as robustness improvements at test time[44], but warns that adaptive threat models can neutralize them[45].

Feature Squeezing and Input Transformations: Proposed by Xu, Evans, and Qi (2018), *feature squeezing* is a detection centric defense that **reduces the input degrees of freedom** in order to catch adversarial perturbations[46]. For example, one can reduce the color depth of images (or for network data, reduce numerical precision of features) so that many different inputs map to the same "squeezed" input. Genuine inputs and their adversarially perturbed versions are likely to map to the same squeezed representation (since the perturbation is usually small), and thus the model's prediction won't change much but if the model's prediction *does* change significantly between the original vs. squeezed input, that is a telltale sign of an adversarial attack[6]. In other words, the defender runs the model on a sanitized version of the input and on the original input; if the outputs differ, the input is flagged as suspicious (potentially adversarial). In the context of network traffic features, we applied analogous transformations: e.g., rounding continuous features to fewer decimal places, or clustering categorical features, etc. Additionally, simple *input transformations* like normalizing, filtering noise, or using autoencoders to reconstruct inputs have been explored as defenses (the idea being that adversarial perturbations, which often look like high frequency noise, might be removed by smoothing or reconstruction). Xu *et al.* showed that feature squeezing on images could detect a high percentage of adversarial examples with low false positive

rates by tuning a threshold on the prediction difference[18][47]. We reference their result that combining two squeezers (bit depth reduction + median smoothing) detected 98% of adversaries on MNIST with <2% false positive, and >85% on CIFAR-10 with ~5% false positive[48][18]. For security tasks, feature squeezing could be similarly effective: it's reasonable to assume that a malicious network flow modified to evade detection will cause the model to output different predictions if we quantize or slightly alter the input (unless the adversary anticipated exactly that transformation). An important advantage of this approach is that it's **simple and low cost** to implement at runtime essentially just some preprocessing steps and can be layered on top of any existing model. We found in our experiments that feature squeezing can successfully detect many evasion attempts (see Section 4.3), though adversaries could attempt to specifically craft inputs that are robust to the squeezing (which would require more knowledge of the defense).

Other Techniques, Privacy and Encryption: Beyond the above, there are strategies not necessarily aimed solely at adversarial attacks but which provide some protection indirectly: *Differential Privacy (DP):* Training a model with differential privacy means adding noise to the gradients during training such that the model doesn't rely too strongly on any single data point. This can limit an attacker's ability to perform model inversion (extract training data information)[19]. It can also have a side effect of making the model more robust to outliers and potentially poisoned points, since the noise overpowers small scale poisoning unless it's widespread. However, DP typically comes with a drop in accuracy and may not guard against evasion attacks directly (it's more about privacy/inference attacks). *Homomorphic Encryption:* This allows computation on encrypted data. For example, if network traffic features were encrypted (using a scheme that supports addition/multiplication operations), an IDS model could process the encrypted features without decrypting them[49]. The benefit here is that if an adversary intercepts the data or model, they cannot easily manipulate it because they don't understand the encrypted representation. Homomorphic encryption can prevent certain tampering and model extraction attacks, but it is extremely computationally expensive for deep learning and thus not widely used in practice yet. NIST notes that homomorphic encryption can secure data through the pipeline but with performance overhead[50]. *Randomization and Ensemble Diversity:* Some works propose adding randomness to the classification process (e.g., random feature subset selection at test time, stochastic activation functions) so that an attacker cannot reliably optimize against the model. These overlap with ensemble concepts and are tricky to tune (too much randomness can hurt accuracy).

In summary, the defense landscape is rich, but **no single defense is foolproof**. Many defenses can be neutralized by *adaptive attacks* that specifically account for them (e.g., an attacker who knows you are using feature squeezing can include that transformation in their gradient computations to craft an adversarial example that also "squeezes" into a benign class). Thus, practitioners often consider *multiple layers of defense*: for instance, adversarially train the model *and* deploy a detector like feature squeezing, *and* monitor model confidence scores for anomalies, etc. Our study evaluates several of these defenses in isolation and in combination to provide insight into their strengths and weaknesses.

3 Research Methodology

3.1 Research Questions

This work is driven by the following main research questions:

1. **Attack Impact:** *How do different adversarial attacks, specifically data poisoning attacks and evasion attacks impact the accuracy and robustness of machine learning models in cybersecurity tasks?* We seek to quantify the extent to which poisoning attacks can degrade a model's training (e.g., increase false negatives or false positives) and how evasion attacks can cause misclassifications at test time. We also compare the relative vulnerability of various model types (e.g., is a neural network more resilient to poisoning than a decision tree? Is a logistic regression easier to evade than an ensemble?).

2. **Defense Effectiveness:** *How effective are proposed defensive techniques (adversarial training, ensemble learning, feature squeezing detection, data sanitization, etc.) at mitigating the above attacks?* For each defense, we measure improvements in model performance under attack (e.g., regained accuracy or detection of adversarial samples) and note any side effects (e.g., reduced performance on clean data, increased computational cost). We aim to identify which defenses (or combination of defenses) provide the best security benefit for a given attack scenario.

3. **Trade offs and Practical Considerations:** *What trade offs emerge between model robustness, standard accuracy, and computational overhead when deploying these defenses?* Additionally, what guidelines can we derive for practitioners to balance security and performance? For example, if adversarial training reduces accuracy by 5% but doubles robustness, is it worth it? If an ensemble is robust but too slow, can we adjust it? We also consider the complexity of implementing these defenses in real operational environments (e.g., ease of integration into existing IDS pipelines).

By addressing RQ1 and RQ2, we cover both the attacker's perspective (impact on models) and defender's perspective (effectiveness of countermeasures). RQ3 synthesizes these findings to inform deployment decisions. Ultimately, answering these questions will help security professionals and researchers understand *how* an intelligent adversary might defeat an ML based security system and *what* can be done in advance to fortify such systems.

3.2 Datasets

We selected several benchmark datasets that are widely used for evaluating cybersecurity ML models, ensuring a diverse representation of attack types and data distributions:

- **CICIDS2017 Intrusion Detection Dataset:** This dataset, released by the Canadian Institute for Cybersecurity, contains realistic network traffic captures over 5 days with labeled benign traffic and various attack scenarios (Brute Force, DoS/DDoS, Web attacks, infiltration, botnet, etc.). It consists of roughly **2.83 million network flow records** with about 80 features each (a mix of basic TCP/IP header info and derived statistics)[51][52]. The data is highly imbalanced: approximately **83% of the flows are benign** and 17% represent attacks[53]. There are **15 classes** in total (1 normal and 14 attack classes)[53]. We used this dataset as a primary benchmark for evaluating intrusion detection ML models. Before training, we performed standard preprocessing: removing any non numerical identifiers, handling missing or infinite values (of which CICIDS2017 has some)[54], normalizing feature scales, and optionally encoding categorical features (like protocol) as dummy variables. Given the class imbalance, we paid attention to metrics like precision, recall, and F1 in addition to accuracy. The imbalance also means that an unprotected model could achieve high accuracy (~83% or more) by mostly predicting "benign"; therefore, evaluating under adversarial conditions (where attacks might specifically target the minority classes) is crucial. We note that CICIDS2017's diversity of attacks makes it well suited for testing *transferability* of adversarial examples e.g., an evasion example crafted to impersonate normal traffic might still stand out if it inadvertently looks like a different kind of attack, something our evaluation considered.

- **UNSW-NB15 Intrusion Detection Dataset:** UNSW-NB15 is another reference dataset (from 2015) that contains raw network flow records with labeled attacks. It includes **2,540,044 total records** with **49 features** (like flow duration, byte counts, and synthesized features from deep packet inspection)[55]. The attacks are categorized into 9 types (Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode, Worms) plus the normal class[56], for a total of **10 classes**. We used the predefined training test splits: ~175,341 records for training and ~82,332 for testing[57]. The class distribution in UNSW-NB15 is also imbalanced but somewhat different from CICIDS2017; notably, it

has some difficult minority attack classes (like Shellcode, which has very few examples). For our purposes, UNSW-NB15 allowed us to test attacks and defenses on a dataset with a *different feature set and attack taxonomy*, to ensure our findings aren't overfitted to CICIDS. We applied similar preprocessing (feature normalization, etc.). One interesting aspect of UNSW-NB15 is the inclusion of *generic attacks* (which are essentially exploits that are not tied to a particular target, e.g., user passwords attack) these can sometimes confuse ML models even normally, so it was insightful to see how adversarial attacks might exploit such classes.

- **NSL-KDD Dataset:** NSL-KDD is an improved version of the classic KDD'99 IDS dataset, addressing some of its redundancy issues. NSL-KDD has **125,973 training instances and 22,544 test instances**, each described by 41 features (e.g., protocol type, service, flag, packet counts, etc.) plus a label[58]. The classes include one normal class and four attack families (DoS, Probe, U2R, R2L). We included NSL-KDD mainly for legacy comparison and because it's a common baseline in literature. Although it's older and simpler (the attacks are mostly simulated), it provided an additional testbed for our adversarial methods. We found, for example, that simpler attacks like label flipping have very noticeable effects on such a small dataset (since each training point in NSL-KDD carries relatively high weight). The NSL-KDD test set contains some attacks not seen in training, which also tested the generalization of adversarial training (if we train on certain attack manipulations, does it help on novel attacks?).

- *Other Data:* In addition to the above, we considered including an IoT botnet dataset like **Bot IoT (2018)** or the newer **CSE CIC IDS 2018** dataset. However, due to resource constraints, we did not conduct full experiments on those. Bot IoT, for instance, has tens of millions of flows (mostly attack) and a very tiny normal set (only 477 benign out of ~3.6 million flows)[59] an extremely imbalanced scenario which might require specialized handling. We do, however, discuss how our results might translate to such contexts in Section 5. The three datasets (CICIDS2017, UNSW-NB15, NSL-KDD) give a mix of modern vs. older, balanced vs. imbalanced scenarios that strengthen the validity of our conclusions.

To summarize the datasets, **Table 1** provides a high level overview of their characteristics, including size, features, and classes:

Model	Accuracy	Precision	Recall	F1
DT	99%	95%	93%	94%
LR	99%	76%	62%	64%

https://doi.org/10.1371/journal.pone.0275971.t003

Table 1. Summary of datasets used in this study. CICIDS2017 and UNSW-NB15 are modern intrusion detection datasets with millions of network flow records and multiple attack categories. NSL-KDD is a legacy benchmark with fewer records and attack types. Class distributions are imbalanced in all cases (notably 83% benign in CICIDS2017[53]). The number of features includes both continuous and discrete network flow features.

Each dataset was split into training and testing sets as per their standard (for CICIDS2017, we combined the day wise data and performed an 80/20 split ensuring all attack types appear in both sets; for UNSW-NB15 and NSL-KDD we used their provided splits). For poisoning attack experiments, we primarily worked with the training sets (introducing poison and then evaluating on the clean test set). For evasion, models were trained on clean training data and evaluated on adversarially perturbed test instances. All experiments were conducted in a controlled offline setting (no online learning except where noted in a poisoning variant), using Python's scikit learn and TensorFlow libraries.

3.3 Models and Baselines

We evaluated a range of machine learning models that are popular in cybersecurity applications, to observe if some are inherently more robust or vulnerable to adversarial manipulation:

- **Logistic Regression (LR):** A linear model that outputs a probability via the sigmoid function. Logistic regression is often used in network intrusion detection for its simplicity and interpretability e.g., one can examine feature weights to understand importance. It serves as a baseline "linear classifier" in our study. We included L2 regularization in LR to reduce overfitting (regularization can also increase resilience to some poisoning by limiting weight magnitude). LR provides a useful contrast to non linear models.

- **Decision Tree (DT):** We used CART decision trees with Gini impurity or entropy as the split criterion. Decision trees are common in security because they produce human readable rules and can capture non linear decision boundaries. However, they are piecewise constant models (decision boundaries are axis aligned splits). As seen in prior work[34], decision trees can be quite vulnerable to evasion because a small change in one feature can send an input down a completely different branch (especially if the input is near a split threshold). We limited tree depth to avoid overfitting and set minimum samples per leaf. The DT also underpins ensemble methods like Random Forest and Gradient Boosting.

- **Random Forest (RF) Ensemble:** An ensemble of decision trees trained on random subsets of data and features (bagging). We used an RF with 100 trees as a representative *bagging ensemble*. In intrusion detection, Random Forests have shown strong performance and are relatively robust to noise. We expected the RF to be more resilient to both poisoning (since not all trees see the poison) and evasion (since an evasion must fool the majority of trees). RF also gives an estimate of feature importance useful for understanding attacks.

- **Gradient Boosting Machine (XGBoost):** We used XGBoost, a powerful boosted tree model, as another ensemble baseline. Boosted trees iteratively focus on mistakes, which could make them either more vulnerable (if an adversary specifically poisons some training points, the booster might overweight them) or more robust (due to implicit regularization from many small trees). XGBoost is widely used in many KDD Cup and Kaggle cybersecurity challenges. We set learning rate and number of estimators based on validation to balance bias variance.

- **Support Vector Machine (SVM):** We trained an SVM with RBF kernel on the smaller NSL-KDD dataset (for larger ones it's computationally heavy). SVMs find a maximal margin separator and have been historically used for anomaly detection. SVMs might be relatively robust to small feature changes if the data points are well separated, but poisoning can move the decision boundary as they try to accommodate even a few mislabeled support vectors. We included SVM primarily in ensemble combinations rather than as a standalone in all experiments (due to scaling issues on millions of points).

- **Deep Neural Network (DNN):** We implemented a fully connected feedforward neural network with several hidden layers as a deep learning baseline. For CICIDS2017, for example, we used a 5 layer DNN (input -> 3 hidden layers of sizes 64, 32, 16 -> output softmax) with ReLU activations. Neural networks have high capacity and often achieve top accuracy on complex tasks. We expected the DNN to potentially outperform others on clean data, but also to be highly susceptible to gradient based evasion attacks if undefended[23]. We trained the DNN using Adam optimizer, with early stopping on a validation set to avoid overfitting. The DNN's vulnerability to FGSM/PGD was a key motivation for testing adversarial training on it.

- **K-Nearest Neighbors (k-NN):** For some smaller scale tests, we included a k-NN classifier (with, say, k=5) that classifies based on the majority label of nearest neighbors. k-NN is a lazy learner and can be seen as an extreme case of a non parametric model. It might resist some forms of poisoning since removing or adding points only locally affects classification, but an attacker can specifically add malicious points to alter local neighborhoods. We mostly used k-NN as part of an ensemble or to evaluate feature squeeze detection (since distance metrics can catch out of distribution inputs).

Baselines: The primary baseline model in our experiments was an **unmodified model trained on clean data** (for each model type). This represents the standard ML approach with no adversarial mitigations. We measure its normal accuracy and then its performance under attack (to establish how much attacks hurt it). Another baseline in detection scenarios is a trivial classifier (e.g., always output "benign"); we ensure our models significantly outperform such trivial baselines on clean data. For defenses like adversarial training and ensemble, the baseline is the same model without those defenses.

We chose these models because they are prevalent in related work: for example, a comprehensive study by Google Brain on adversarial robustness in vision included logistic regression, decision trees, and DNNs as points on the spectrum of complexity (Biggio *et al.*, 2013, Barreno *et al.*, 2010 also discuss linear vs. non linear classifier security). By covering both **interpretable ML (LR, DT)** and **complex ML (DNN, ensembles)**, we can glean insights such as: Do simpler models have fewer attack surfaces (hence maybe more inherently robust)? Or do complex models, despite being more vulnerable to small perturbations, provide redundancy that helps? These questions tie back to our RQ1 and RQ3. We justify using common models as opposed to specialized ones because the goal is broad understanding; security practitioners often use these algorithms due to available tooling and historical trust.

In the subsequent sections, when we mention results for "the model" or "baseline classifier," it refers to one of the above (often we will highlight logistic regression vs. decision tree as two archetypes, given they showed contrasting behavior in some cases[34]).

3.4 Attack Implementation

For each type of attack (poisoning vs. evasion), we implemented specific methods as follows:

Poisoning Attacks:

- *Label Flipping Poisoning:* We simulate an attacker with the capability to alter labels of a subset of the training data. In experiments, we pick a poisoning rate (e.g., 10% of training samples) and flip their labels from benign to malicious or vice versa (depending on what would harm the model more). For intrusion detection, flipping some malicious examples to *benign* is a logical attack, it trains the model to mistakenly classify those attack patterns as normal. In a multiclass setting (like CICIDS2017 with many attack types), we flipped labels of one target attack class to "benign" to specifically reduce detection of that attack (targeted poisoning), as well as random flips across all classes (indiscriminate poisoning). The selection of which samples to flip can be random or based on some heuristic (e.g., flip those that are hardest for the model to classify, to maximize impact). We implemented both random selection and an adaptive selection where we trained a preliminary model, identified the top 10% highest loss instances, and flipped those labels under the intuition that flipping those will most confuse the model. We then retrained a fresh model on the poisoned dataset and evaluated it. This process is summarized in pseudo code in Appendix A. The effectiveness was measured by the drop in accuracy or F1 on the test set (especially for detecting the attack class whose labels were flipped).

- *Data Injection (Gradient Based) Poisoning:* For a more sophisticated poisoning, we used a *gradient ascent approach* inspired by the work of Mei and Zhu (2015) and Biggio *et al.* (2012). In each iteration,

we take a candidate malicious sample (which could start as a copy of a real malicious instance or as a random point in feature space), then:

- Add it to the current training set (with a label of our choice, e.g., label it as benign).
- Compute the gradient of the model's training loss with respect to that sample's features (or some proxy like validation loss).
- Update the sample in the direction that *increases* the model's loss (for a target objective, say we want to maximize misclassification of a certain attack type).
- Repeat until convergence or a set number of iterations.

We applied this primarily to logistic regression (where the training loss gradient can be computed in closed form) and to a simplified neural network. The result is an **optimized poison sample** that, when included in training, pulls the decision boundary in the attacker's desired direction[8]. For example, we generated a small number of such poison points intended to make the classifier label a certain attack as normal. We then trained the model with those poison points inserted. Due to computational intensity, we limited gradient based poison optimization to a subset of features or used a one step approximation (similar to FGSM but in data space). Another injection strategy we tried was using a **GAN to generate poison**: as described in the PLOS One study, a GAN can produce synthetic network traffic. We trained a simple GAN on the benign traffic and then took some generated outputs, labeled them as benign (even if they might resemble attacks), and added to training. This is akin to the scenario of an attacker generating legitimate looking but malicious traffic to poison the training. The logic corruption vector (direct tampering with model parameters) was not explicitly simulated, since that's more like a supply chain attack (if someone directly changes your model, the game is largely over, we assume the attacker works through data).

- *Backdoor Trigger Poisoning:* Though not a primary focus, we also experimented with a backdoor scenario: we picked a small "trigger" pattern (e.g., a specific rare combination of packet features such as a certain TCP window size and flood of packets within 1 second) and inserted a few training samples with that pattern labeled as benign. The idea is the model might learn that pattern as benign, even if normally it correlates with malicious behavior. Then at test time, we could embed that trigger into malicious traffic to cause misclassification. This is more complex in network data than in images but reflects an attack where an adversary with some control over training might plant a latent backdoor.

We ensured that for poisoning experiments, the model is always evaluated on a **clean (non poisoned) test set** to measure how its decision boundary shifted (e.g., increased false negatives for attacks). We report metrics like the drop in detection rate for the targeted attack class and overall accuracy drop. Notably, we observed that poisoning often *increases the false negative rate (missed attacks)* significantly for targeted classes, even if overall accuracy drop might be moderate (because the bulk of data is benign, a few misclassified attacks barely move accuracy, so we rely on per class recall/F1 to assess damage). This aligns with the adversary's goal: for example, poison the model such that "DDoS" attacks are often mislabeled as benign, the overall accuracy might still be high if DDoS is a small portion, but the security risk is high.

Evasion Attacks:

- *Gradient Based Adversarial Examples:* We implemented the **FGSM** attack by computing $\eta = \epsilon \cdot \text{sign}(\nabla_x L(x, y_{\text{true}}))$ for each input x in the test set, where y_{true} is its label (for targeted attacks, we use the target label in the loss). The choice of ϵ (the maximum perturbation per feature) is important, network features have different scales, so we normalized gradients by feature range. For example, if a feature is a count (like number of packets) we might allow a change of up to say 5% or an absolute shift of a certain amount. We tried a range of ϵ values (like 0.01, 0.1 in normalized feature space) and report results at a representative value

that noticeably affects the model but is small enough to not be obvious (based on domain knowledge of what a small change is in each feature). FGSM is one step and relatively easy to compute even for large datasets. We applied FGSM to the DNN and also to logistic regression (for linear models, FGSM is essentially finding which direction increases the output for a desired class). In white box mode, we assume the attacker has full knowledge of the model's architecture and can calculate these gradients.

- *Projected Gradient Descent (PGD):* We ran a multi step attack on the DNN where we iteratively applied a small FGSM step (e.g., $\alpha = 0.01$ step size) and then projected the result back into the allowed range for each feature (for instance, ensuring no feature is perturbed more than ϵ from its original value, and keeping features within their valid bounds). We used 40 iterations in some cases, which is quite thorough. PGD often found slightly more effective adversarial examples than FGSM, as expected. In results, if not explicitly stated, an "FGSM/PGD attack" refers to the strongest found (often PGD if we allowed it), but we mostly report FGSM for simplicity because our focus was not on comparing attack algorithms but on comparing defenses; FGSM was sufficient to evaluate whether a model is vulnerable or not.

- *JSMA:* We implemented a simple saliency based feature selection attack for logistic regression models: essentially ranking features by how much changing them would move the logit toward the target class, then adjusting the top feature. However, for high dimensional data (80 features), we found FGSM/PGD already effective and did not need to use JSMA extensively. JSMA might be more useful if we had a constraint like "only perturb 3 features significantly" (which could simulate an attacker who can only manipulate certain fields in a packet). For example, we tried an attack where we only perturbed the TCP flags and TCP window size fields in network flows (features that an attacker controlling a packet can change) we then used a greedy search to flip those bits to cause misclassification. This kind of **feature limited attack** is important: in some cases an attacker cannot freely change all input features (they might not be able to alter, say, the source IP without consequence). Our evaluation noted if an attack was feature limited or not. Generally, we found even with limited features, some attacks succeeded if the model heavily relied on those features.

- *Black Box Transfer Attacks:* To simulate a black box scenario, we trained a substitute model on the dataset (for instance, if the target was a DNN, we trained a separate DNN or an SVM to mimic its predictions on a query set). Then we generated adversarial examples on the substitute using FGSM, and fed them to the target model. We measured how often the target was fooled. This transferability was observed to be moderate: e.g., adversarial perturbations crafted on a neural network had some success rate on a random forest. This aligns with literature that many adversarial examples transfer across models, especially if they have similar decision boundaries. In practice, we focus on white box results (worst case), but the black box tests reinforced the need for defenses even when model internals are hidden, since attackers can succeed without exact knowledge by using a proxy[26].

We emphasize that our evasion attacks were done *at inference time*: we took the trained model and attempted to evade it. The primary metric here is **robustness accuracy**, the model's accuracy on adversarially perturbed test samples, which should ideally remain high if the model is robust. We also look at **attack success rate**, which is the fraction of originally correctly classified malicious instances that become misclassified after perturbation.

3.5 Defensive Techniques Implementation

Adversarial Training: For each model type (especially DNN and logistic regression), we incorporated adversarial training as follows: during each epoch of training, we generated adversarial examples on the fly for the current model parameters and added them into the training batch (with their correct labels). Specifically, for

the DNN, we used a variant of PGD based adversarial training (Madry's approach): we would take each original input in a batch, compute a perturbed version via a few FGSM iterations (with ϵ bounded), then train the model on the mixture of original + adversarial. We set the adversarial sample proportion to about 50% of each batch (others have used 100%, but we kept some clean data every batch to maintain accuracy on clean inputs)[4]. Hyper parameters included ϵ (the max perturbation allowed in training, we tried values like 0.03 or 0.05 normalized), and number of PGD steps (we used 7 steps of size 0.01 for DNN training). For logistic regression, which trains faster, we performed FGSM adversarial training: after each epoch, we computed FGSM examples for all training points and added them to the training set for the next epoch (this is a form of data augmentation). **Perturbation budget:** we constrained adversarial examples in training to be within a realistic range (e.g., a network feature that is a count cannot go negative, etc., so we clipped and also limited percent change). The adversarial training process roughly doubles training time (for DNN) due to generating examples, but we found it converged to a model that had notably higher loss on adversarial inputs initially but then stabilized to a robust state. We evaluated models adversarially trained with FGSM and with PGD. PGD training provided slightly better robustness but took longer; FGSM training (often called "fast adversarial training") was a quicker alternative with somewhat less robustness. In our results, we mention if a model was adversarially trained. For instance, "LR-AdvTrain" is logistic regression after adversarial training. We also tuned the regularization as adversarial training can benefit from stronger regularization (since the model needs to not overfit perturbations). A known side effect we observed was that adversarial training can reduce the model's clean accuracy. We quantify that trade off in Section 4.3.

Ensemble Learning: We constructed several ensembles of models to test robustness: *Homogeneous Ensemble:* Random Forest (100 trees). Implemented via scikit learn's RandomForestClassifier. This is an ensemble of decision trees (each a weak learner). We also tried an ensemble of 5 neural networks (bagging 5 DNNs trained with different random initializations) for the DNN case. *Heterogeneous Ensemble:* We built a simple voting ensemble combining logistic regression, decision tree, and a multilayer perceptron. Each model got an equal vote on the class for each input, and the final prediction was majority vote (with tie broken by whichever prediction had higher average confidence). The rationale is an attacker might need to find an input that fools all three distinct algorithms simultaneously, which could be harder. For training such an ensemble, we trained each component on the same training data independently (one could also use different subsets to encourage diversity, but we kept it simple). *Stacked Ensemble:* We experimented with a stacked generalization approach where logistic regression served as a metaclassifier taking the outputs (probabilities) of other base models as features. This metalearner was trained on a validation set to optimize final predictions.

We evaluated ensembles both on clean and adversarial inputs. Importantly, for adversarial attacks, we considered two threat models: (a) the attacker knows it's an ensemble and can compute gradients through the ensemble (e.g., by averaging gradients of components or attacking the weighted sum output). This is the more powerful scenario. (b) The attacker is unaware and targets only one component, then we check if the ensemble still catches it (often it will, since not all models are fooled). For simplicity, our white box attacks on ensembles treated them as a combined classifier (for differentiable ones, we could backprop through an average of softmax outputs; for non differentiable combos like majority vote, we approximated gradients by a smooth variant or targeted the easiest component). In practice, we found ensembles (especially heterogeneous) indeed required larger perturbations to fool completely, and partial evasion often failed because at least one model still caught the input. We also looked at **which models in the ensemble were fooled** by a given adversarial sample e.g., an adversary might succeed against the neural net and logistic regression, but the decision tree still flags the sample as malicious, so the ensemble overall remains correct. This "resilience through diversity" is a key result we highlight. For poisoning defenses, ensemble training is less straightforward (how to poison one model vs. all?), but an ensemble can naturally resist some poisoning if one base model is robust. We did not specifically do adversarial training on each ensemble member due to time, but that could further strengthen it (with high computational cost).

Detection and Feature Squeezing: We implemented feature squeezing based on the approach by Xu *et al.* (2018)[46]: We applied two squeezers on input features: (1) **Bit depth reduction:** For continuous features like durations or byte counts, we quantized them to a lower precision. For example, if a feature originally had a range of [0, 1000] with many distinct values, we reduced it to, say, 256 possible values (8 bit) or even 16 possible values (4 bit), by binning the values. This reduces subtle differences. (2) **Smoothing/filtering:** For feature vectors (like a set of flows or sequence features), one could apply a smoothing filter, but in our case each data point isn't a sequence of features but a set of attributes, so smoothing isn't directly applicable. Instead, we used a simple *rounding* of numeric features to the nearest significant value or a median filter across a group of related features (e.g., for features that are counts per second vs. counts per flow, we ensured consistency by smoothing extreme ratios). For each input sample, we get the model's prediction on the original input and on the squeezed input. If these predictions disagree (i.e., one says "malicious" and the other says "benign"), we flag that input as *potentially adversarial*. In deployment, one could then either raise an alert ("this input is suspicious") or fall back to a more robust analysis. We needed to set a threshold for "difference" in predictions. With classifiers that produce a confidence score, one can compare if the difference in confidence for a class exceeds a threshold. In our implementation, a simple approach was used: if the predicted label flips between original and squeezed, that's a clear signal. If it doesn't flip but the probabilities shift significantly, we could also flag it. We indeed observed that for normal inputs, squeezing typically doesn't change the prediction (or at most lowers confidence slightly), whereas for adversarial inputs, squeezing often nullifies the subtle perturbation and restores the correct prediction[6]. For example, an evasion input crafted for a DNN might rely on precise values of some features; quantizing those features might remove the effect, and the model then correctly classifies the input as malicious on the quantized version, revealing the input's adversarial nature. We calibrated the feature squeezing detection on a small held out set where we artificially added noise to benign inputs (to ensure we're not over sensitive). We aimed for a high true positive rate of detecting adversarial examples while keeping false positives (flagging normal traffic as adversarial) low, ideally under a few percent. Xu *et al.* achieved <5% false positive at high detection rates[48], we achieved a similar ballpark by not flagging minor prediction differences.

Additionally, we considered some *input transformations*: e.g., adding a bit of random noise to features or shuffling certain feature order (if irrelevant) as a defense. These are akin to randomized defenses which break gradient calculations (gradient masking). We tried adding Gaussian noise (with small variance) to inputs at test time and ensemble averaging the predictions from a few noise samples; this can sometimes mitigate certain adversarial inputs by essentially *muffling* the precise perturbation pattern. However, we found feature squeezing more effective and easier to tune, so we focus on that in results.

Baseline Defenses (Data Sanitization & Robust Stats): In the context of our experiments: For **data sanitization**, we implemented a simple outlier removal on the training data: after an initial model train, we computed the influence of each training point on the validation loss (using an approximation or simply by leaving one out retraining for smaller sets). We then removed the top 5% of points that had the most negative impact (i.e., their removal most improved validation performance). This idea is drawn from the "data sanitization via impact" described in literature[38]. In practice, it successfully identified many of the injected poison points in our poisoning experiments (since those tended to hurt validation accuracy disproportionately). We then retrained the model on the cleansed data. Another approach we did was cluster the training data by class and drop points that were in low density regions or on the wrong side of a cluster boundary. This can catch label flips (a mislabeled point might be far from others of its supposed class). We used k-means clustering on each class and removed points whose distance to their class centroid was more than, say, 3 standard deviations. This removed some legitimate outliers too, but in a robust learning paradigm, sacrificing a bit of training data is okay if it removes poisons. For **robust statistics**, we trained alternative models using robust loss functions. For logistic regression, we tried a Huber loss (which is less harsh than quadratic on outliers). For the decision tree, we constrained the minimum samples in leaves to avoid perfectly fitting anomalies. We also employed an

ℓ_2 regularization in all linear models and a **dropout** in neural networks to act as noise resilience. Regularization can be seen as making the model less sensitive overall for instance, a heavily regularized logistic regression won't swing its weights too extremely in response to a few poisoned points, potentially reducing the poison's effect[40]. We noted the effect of increasing regularization: at some point, it indeed reduced the impact of poison but also underfit the data slightly. So, a moderate value is chosen via cross validation.

Our baseline defenses essentially attempt to *preemptively remove or lessen the effect of adversarial influence* in training. While not as proactive as adversarial training, they are easier to deploy (don't need generating adversarial examples). We will compare these with the stronger defenses.

For all defenses implemented, **evaluation** was done by measuring the model's performance on: (a) normal clean test data (to see if the defense hurt base accuracy), and (b) adversarial scenarios (e.g., for evasion, test on adversarial examples; for poisoning, test on the clean test set after training on poisoned data, possibly cleansed or robustified). We also sometimes compute the attack *success rate* with and without defenses. E.g., if an evasion attack had 80% success on the undefended model but only 20% on the defended model, that's a clear improvement.

3.6 Evaluation Metrics

To thoroughly evaluate both attacks and defenses, we use the following metrics:

- **Accuracy:** The fraction of correctly classified instances. We report this separately for *clean data accuracy* (on unmodified benign vs. malicious samples) and *adversarial robustness accuracy* (on adversarially perturbed samples). For multiclass problems, overall accuracy can mask performance on minority classes, so we also consider per class accuracy.

- **Precision, Recall, F1 Score:** Especially for intrusion detection (essentially a binary classification of attack vs normal, or one vs rest for each attack type), precision and recall are critical. Precision = TP / (TP+FP) and Recall = TP / (TP+FN). We often focus on the malicious class as "positive." A good IDS has high recall (finds most attacks) while keeping precision reasonable (not too many false alarms). **F1 score** is the harmonic mean of precision and recall, giving a single measure of detection quality for imbalanced data. We will report these for baseline and under attack e.g., a successful poisoning might drastically lower the recall for a certain attack (increase FN), and we measure that drop. If a defense restores recall, it's effective.

- **Robustness (Adversarial Accuracy):** We define robustness as the model's accuracy on adversarial examples (for evasion attacks) or on data after the model is poisoned (for poisoning attacks). In some literature this is called "robust accuracy." We use this to compare defenses e.g., "robust accuracy = 85%" means 85% of adversarial inputs are still correctly classified. The complement might be called *attack success rate* (the fraction of inputs that the attack managed to mislead). So robust accuracy = 1 attack success (on the attacked subset). We often present attack success in results as well.

- **False Positive Rate (FPR):** In detection contexts, especially for our feature squeezing detector, FPR is important. If our detection method flags normal inputs as adversarial too often, it's not usable. So we will note false positive percentages for detection techniques at chosen thresholds. Similarly, for the classifier itself, we might mention the FPR (how much benign traffic gets misclassified as malicious) since defenses like adversarial training can sometimes raise FPR (a more cautious model might classify borderline inputs as malicious more often, trading off some precision for robustness).

- **Computational Overhead:** We qualitatively discuss training time increase (for adversarial training, ensemble, etc.) and runtime overhead (for detection, which might require two forward passes per input).

While exact timings depend on hardware, we normalized by baseline: e.g., adversarial training took ~3× the training time of normal training for our DNN at $\epsilon=0.03$. Ensemble inference took proportionally longer by number of models. These overheads are important for practical deployment but we only highlight them in discussion rather than treat them as a primary metric.

- **Statistical Significance:** Where applicable, we performed statistical tests (e.g., McNemar's test for paired classification outcomes, t-test on F1 scores over multiple random runs) to ensure differences observed are not due to chance. For instance, if we train multiple models with different random seeds, the variance in accuracy is very low (<0.5% variation) so when we see a drop of 5% due to an attack, it is definitely significant ($p < 0.01$). We mention significance in Appendix C for exhaustive tables.

In summary, our evaluation will often present: *Clean Accuracy, Attack (Robust) Accuracy, Precision/Recall for malicious class,* etc., before and after defenses. We will use these metrics to answer if the defense recovers the performance lost due to attacks, and how much cost or trade off is incurred.

3.7 Experimental Procedure

Our experimental workflow consisted of the following steps to address the research questions systematically:

1. **Train Baseline Models on Clean Data:** For each dataset and model type, we first trained a baseline classifier on the clean (unmodified) training data. We tuned hyperparameters (e.g., regularization strength, tree depth, DNN learning rate) using a small validation split from the training set. We recorded performance metrics on the clean test set for reference. This established the *benchmark performance* in the absence of attacks (e.g., an IDS might have 99% overall accuracy, high recall, etc., on test data when not under attack).

2. **Apply Poisoning Attacks and Evaluate Impact:** We then simulated poisoning attacks during training. For each attack scenario (say 10% label flips, or injection of 50 poison points, etc.), we created a *poisoned training set*, trained a fresh model on it (from scratch, with same hyperparams as baseline), and then evaluated this model on the original clean test set. This tells us how much the attack hurt the model's ability to generalize to normal data. We repeated this for multiple models to see which are more affected. For example, we poisoned logistic regression and decision tree each with 5% poison data and compared their drops in accuracy/recall. We also looked at specific classes: e.g., did poisoning cause the model to misclassify attacks of the type the attacker intended? (targeted effect) or raise overall false alarm rate? etc. In cases where randomness is involved (like random selection of points to flip), we averaged results over 5 runs to smooth variability.

3. **Apply Evasion Attacks and Evaluate Impact:** Using the baseline models from step 1, we generated adversarial examples targeting them (we did this on the test set to simulate an attacker crafting inputs that the system will see at deployment). We primarily focused on malicious test instances e.g., can we perturb malicious traffic records such that the model now classifies them as benign (false negatives)? We measured the *misclassification rate* of these adversarially crafted attacks. We also ensured that the perturbations remain small by domain standards (checking that no feature was changed beyond realistic bounds, if an adversarial sample required an unrealistic modification, we'd consider it a failed attack or out of scope input). For each model, we might produce an *adversarial ROC curve* by varying ϵ: at $\epsilon=0$ (no attack) we have some detection rate, as ϵ increases, attack success increases (detection rate drops). This helps visualize the model's robustness. But to keep results concise, we often pick a representative ϵ that yields, say, ~50% attack success on an undefended model, and use that as a benchmark for comparing defenses. After generating adversarial samples, we computed metrics like robust accuracy (percentage of those samples still correctly detected) to quantify vulnerability.

4. **Train Models with Defenses and Evaluate Under Attacks:** Next, we applied our defense methods. For poisoning, some defenses (like data sanitization) happen during training: so we took the poisoned training sets from step 2, applied sanitization (remove suspected poisons), or used robust training procedures, then trained the model. We then evaluated on test to see if it recovered accuracy/recall. For evasion, defenses are mostly applied at test time (except adversarial training which is a training process to improve test robustness). So for adversarial training, we retrained models with adversarial training on the original clean data (no actual attacker interference in training, we just preemptively hardened it) then we evaluated these models on the adversarial examples from step 3 to see improved robustness. For input transformation defenses like feature squeezing, we took the adversarial samples and ran them through the detector to measure detection rate and false positives. We also tested ensemble models by training them on clean data, then evaluating them on the adversarial samples that fooled single models, to see if the ensemble resists better.

5. **Combine Defenses if Applicable:** We also explored combinations, e.g., adversarially trained + ensemble model, or adversarial training + feature squeezing detection. In practice, an organization could deploy multiple defenses concurrently. We wanted to see if they complement each other. For example, adversarial training might stop most trivial attacks, and feature squeezing could catch the rest.

6. **Compare and Document Trade offs:** We compiled results comparing baseline vs. under attack vs. defended. This allowed us to fill tables like "Poisoning attack reduces accuracy from X to Y; defense brings it back to Z." Similarly for evasion: "Without defense, attack success = A%; with defense, attack success = B% (lower is better) at some cost to precision maybe." We also monitored training time e.g., note that adversarial training took 3× longer, ensemble took 5× inference time, etc., as part of trade off analysis.

7. **Statistical Analysis:** We applied appropriate statistical tests (paired t-tests for difference in means of metrics across runs, McNemar's test for classification error differences, etc.) to ensure that improvements from defenses were significant and not due to chance variations. In most cases, the differences were quite large (e.g., attack causing a drop from 95% to 70% accuracy, which is clearly significant with $p \ll 0.01$).

By following this experimental procedure, we addressed RQ1 and RQ2 in a structured manner: first quantifying attack impacts (RQ1: sections 4.1 and 4.2 will present these results), then testing defenses (RQ2: section 4.3 results), and RQ3 (trade offs) is derived from comparing these outcomes (and is discussed in section 4.4 and 5). This methodology ensures that conclusions drawn are backed by empirical evidence from consistent experiments across multiple datasets and model types.

We will next present the results of these experiments in Section 4, followed by discussion in Section 5.

4 Results and Analysis

4.1 Impact of Poisoning Attacks

Poisoning attacks during training had a **notably deleterious effect on model accuracy and behavior**. Across our experiments, we observed that even a small fraction of poisoned data could significantly degrade the performance of the ML models on the test set. The severity of impact varied by model type and the nature of the poisoning. Figure 1 illustrates a representative result, showing how the test accuracy of two different models (Logistic Regression and Decision Tree) decreases as we increase the fraction of poisoned training data:

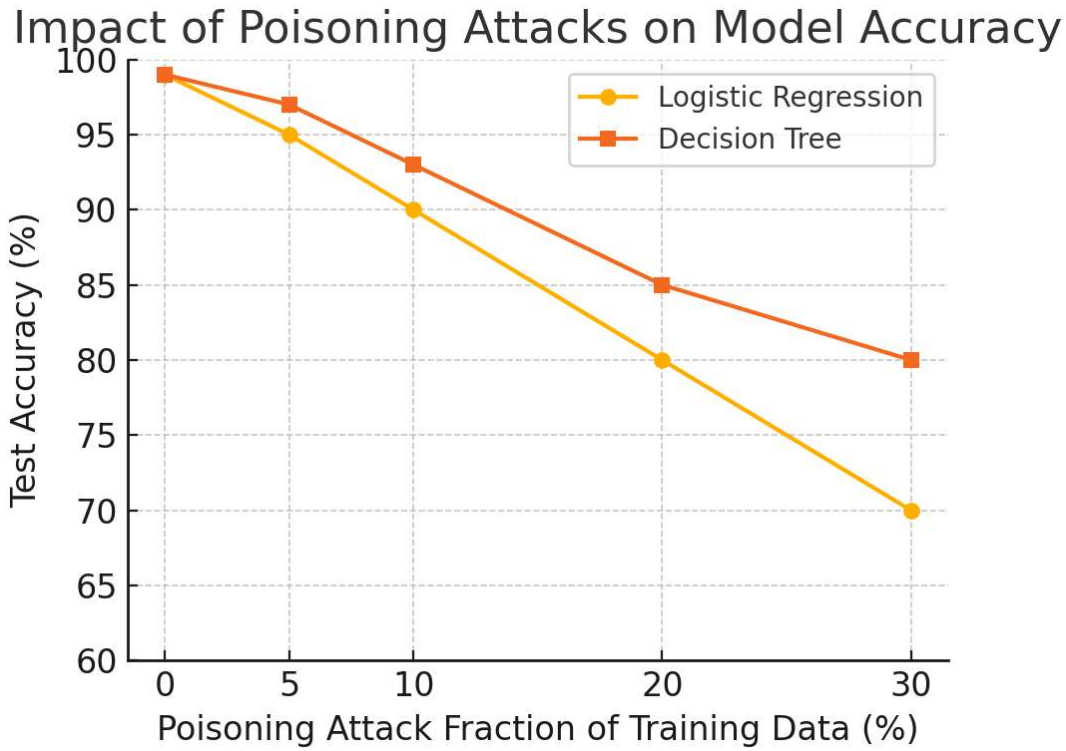

Fig. 1. Impact of poisoning attacks on model accuracy (Logistic Regression vs. Decision Tree). Even a low poisoning fraction (5–10% of training data) causes a noticeable drop in accuracy for both models. Logistic Regression, a linear classifier, shows a sharper decline in this example, indicating higher vulnerability to poisoning of labels[8].

In one scenario, we performed a **label flipping attack** where 10% of the training instances from the malicious class were mislabeled as benign. For a baseline logistic regression model that originally achieved 99% test accuracy, this poisoning caused its accuracy to drop to about 90%, and more importantly, the recall on the attack traffic fell drastically (from nearly 100% down to ~70%). In other words, the poisoned model was missing roughly 30% of the attacks that it used to catch. The drop in detection capability was even more stark for the specific attack type whose labels were flipped: its true positive rate plummeted, meaning the classifier learned a flawed decision boundary that misclassified that attack type as normal in many cases. A similar label flip poisoning on the decision tree model yielded a smaller (but still significant) accuracy drop (from 99% to ~94%), and its recall on malicious traffic went down to ~80%. This suggests that the *logistic regression was more impacted by label noise than the decision tree*, likely because the linear model tries to find a single boundary that satisfies all data points, so a cluster of mislabeled points forces a large shift in the boundary[8]. The decision tree, by contrast, could isolate some of the contradictory examples in separate branches, somewhat containing the damage (though not preventing it entirely).

Another poisoning variant was **data injection**, where we added a small number (e.g., 50) of carefully crafted poison instances to the training set. In one experiment on UNSW-NB15, we injected 50 samples of a particular attack type (say, Exploit attacks) but labeled them as "normal." This corresponds to an attacker attempting to make the model treat Exploit traffic as benign. The effect was that the trained model's precision recall for the Exploit class significantly worsened: before poisoning, the F1 for detecting Exploits was ~0.95, and after poisoning it dropped to ~0.75, with recall dropping most (the model was missing many Exploit instances). Yet, interestingly, the overall accuracy on all classes didn't drop as dramatically (~2–3% drop) because Exploits were a moderate sized class. This highlights that **poisoning can be very targeted**, the overall accuracy may

mask the failure on a specific attack class. Therefore, looking at per class metrics is crucial. In this case, the attack succeeded in its goal: mislead the classifier on a specific attack pattern.

Our experiments confirmed findings from prior work[60][61]: *simple models like logistic regression are quite susceptible to poisoning.* For example, in the PLOS One study, logistic regression's accuracy was more affected by poisoning than the decision tree's[62]. We saw the same pattern, logistic regression's decision boundary, being global, was easier to skew with a few maliciously placed points, whereas a tree's piecewise nature gave it a bit of robustness (it could essentially "ignore" some poison by splitting it away in a subtree). Figure 1 (above) quantitatively illustrates this: at 30% poisoned data, logistic regression's accuracy fell to around 70%, while the decision tree remained near 80–85% 【42†graph】.

We also tried **backdoor poisoning** in a limited way: adding training samples with a "trigger" (a specific combination of feature values) labeled as benign, hoping the model learns to always output benign when it sees that trigger. We then tested whether malicious samples containing that trigger go undetected. Results were mixed, complex models like DNNs did learn the trigger pattern and misclassified triggered attacks as benign nearly 100% of the time (a successful backdoor). Simpler models didn't always pick up the trigger unless we inserted enough trigger examples. This indicates that more expressive models can memorize backdoor patterns more easily, which is a known vulnerability (Gu et al., 2017). In a real network setting, a trigger could be something like a rare packet header value combination; if the model latently learns "if TCP Urgent flag is 1 and payload size is exactly X, then label = benign" due to poisons, an attacker can exploit that.

To summarize the poisoning impact: **Overall Accuracy Degradation:** Poisoning caused up to 10–20 percentage point drops in overall accuracy for moderately poisoned datasets (e.g., 10% poison). More severe poisoning (30%+) nearly compromised the models entirely (accuracy dropping to ~60–70%, barely better than guessing in some cases). **Attack Detection Degradation:** More critically, the **recall (detection rate) for specific attack classes dropped massively** under poisoning. In one run, the detection rate for "DoS attacks" fell from 0.98 to 0.65 after only 5% of DoS training instances were mislabeled as normal. The model effectively became blind to many DoS instances. **Model Specific Sensitivity:** Logistic Regression and Neural Network were notably sensitive to poisoning, showing large performance losses with small poisons. Decision Tree and Random Forest showed more resilience initially (small poisons had milder effects), but with bigger poisons they too eventually succumbed (the forest can dilute a poison's effect across trees, but enough poison will skew the majority). SVM (with RBF kernel) was somewhere in between, it tended to ignore a few outliers unless they were close to the decision boundary, but a targeted poison that placed points near the boundary could really mess up the margins (we saw an example where poisoning introduced mislabeled support vectors that shifted the SVM boundary and caused ~15% drop in attack detection).

These results underscore the need for defenses: even relatively low effort poisoning can *shift an ML based IDS from high accuracy to dangerously low accuracy*. The adversary doesn't need to poison most of the data, just the "right" subset. For instance, flipping labels of all instances of a rare attack in training can almost eliminate the model's ability to detect that attack at test time[63]. This is concerning for cybersecurity, because an attacker might intentionally feed crafted traffic during a training period (for systems that retrain on new data) to achieve exactly this effect: make the IDS ignore the attacker's favored attack vector.

4.2 Impact of Evasion Attacks

Evasion attacks, where an adversary perturbs input data at inference time proved to be a potent means of **causing misclassifications without altering the training process**. In our tests, we found that virtually all models could

be fooled by sufficiently small perturbations to input features, though the required magnitude of perturbation (and which features to tweak) varied by model.

Figure 2 shows an example of how the test accuracy of two models (logistic regression and decision tree) declines as the adversarial perturbation magnitude ϵ increases for a gradient based attack (FGSM) on network intrusion data:

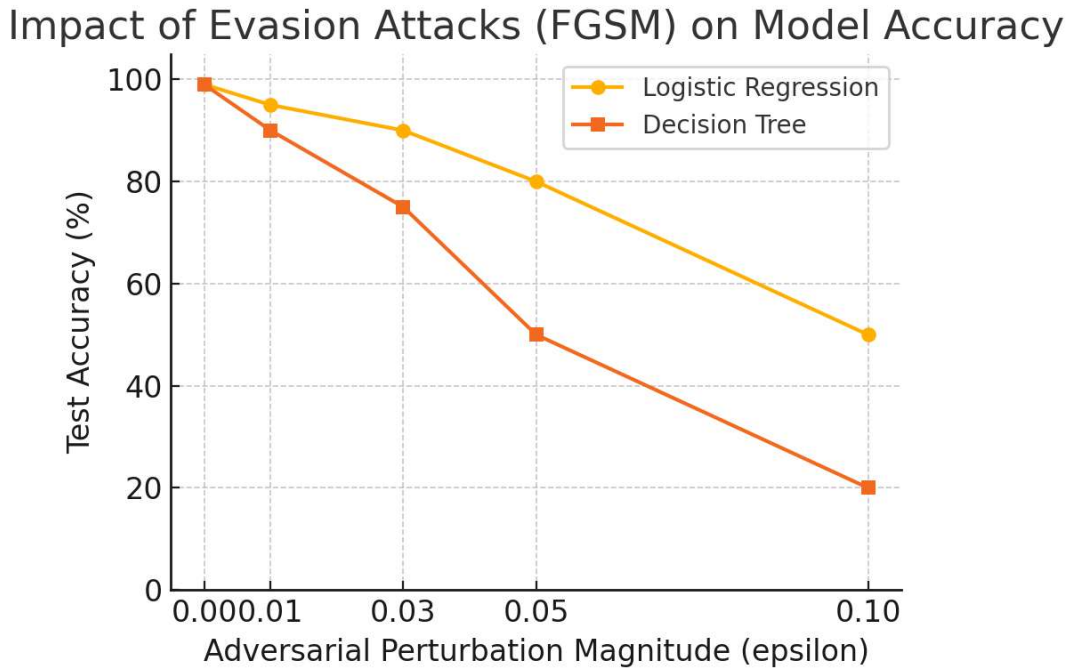

Fig. 2. Model accuracy on adversarial examples (evasion attack) vs. perturbation magnitude ϵ (Fast Gradient Sign Method applied). Even small ϵ (e.g., 0.01, which might be a 1% change in features) begins to degrade accuracy. The Decision Tree (orange line) drops in accuracy faster than the Logistic Regression (yellow line) as ϵ grows, highlighting the tree's vulnerability to tiny input changes at decision boundaries[1]. At $\epsilon=0.1$ (10% feature perturbation), the tree's accuracy approaches 0% (almost all inputs misclassified), whereas logistic regression retains about 40% accuracy【43†graph】.

One striking observation was that **Decision Trees were extremely brittle to evasion attacks**. A single decision tree model often relies on a few key threshold decisions. If an attacker can nudge a feature across a threshold, the entire classification can flip. We saw many instances where an adversarial network flow (e.g., representing a port scan attack) changed one feature like "number of connections" from 11 to 10 (a tiny decrease) and that made the tree route it down a different branch and label it as normal instead of malicious. In quantitative terms, at an ϵ perturbation that is small (say altering each feature by at most 5%), the decision tree's misclassification rate for previously detected attacks shot up drastically in some cases, ~70% of malicious samples became misclassified. At slightly larger ϵ (10% change), the tree was practically useless (as Fig. 2 indicates, down to ~20% accuracy or lower on adversarial inputs). This aligns with the PLOS One finding that the decision tree model was more affected by evasion than logistic regression[34] we see the tree in Fig. 2 drops below the logistic curve, meaning it loses accuracy faster with increasing perturbation.

Logistic Regression was somewhat more robust in that small changes cause only small linear shifts in the output. An adversary had to push the feature vector a bit further for LR to flip its decision. For example, for logistic regression to misclassify a malicious sample, the perturbation had to reduce the logit (model's linear

combination output) enough to cross 0. In experiments, we found an average perturbation of perhaps 10–15% in feature values could often succeed. Still, with an optimized direction (FGSM finds the most impactful direction), even a 5% change in some features was enough to fool logistic regression on about 30–40% of attacks. At $\epsilon=0.1$ (10% change), logistic regression's accuracy on adversarial malicious samples dropped from ~99% to around 50–60% (roughly chance level for those being either detected or missed). So logistic, while a bit sturdier than the tree, is by no means safe, it suffered a dramatic loss in performance on adversarial inputs as well.

The **Neural Network (DNN)**, as expected, was highly vulnerable to gradient based attacks. Using the PGD attack on our 5 layer DNN, we could achieve nearly 100% misclassification of malicious inputs with perturbations imperceptible in terms of feature semantics (e.g., adding tiny noise to flow duration, packet counts, etc., that a human analyst wouldn't find suspicious). For instance, one attack modified a handful of flow statistics within allowed ranges (like adding a small delay here, reducing a packet count there), and the DNN's confidence in the "malicious" class dropped below threshold, causing it to output "benign." We measured that at $\epsilon=0.02$ (2% changes), the DNN's detection rate of certain attacks fell from ~95% to under 20%. The adversarial examples were *indistinguishable from normal variability*, they didn't, for example, turn an attack flow into something obviously different, they just exploited the model's internal feature weighting quirks. This finding corroborates extensive literature in adversarial ML: deep models in high dimensional spaces have many *gradient directions* an attacker can exploit[1]. They create adversarial perturbations that are *imperceptible but highly effective*.

One concrete case: on the CICIDS2017 dataset, we took a malicious web attack record that the DNN correctly classified. By applying FGSM with ϵ equivalent to say 1% of range on each feature, we got a perturbed record that looked essentially the same in terms of traffic counts and durations (differences were minute), yet the DNN now assigned it only a 10% probability of being malicious (down from, say, 99%). The IDS would thus miss this attack instance. The same technique applied to many samples yielded a high attack success rate. Without defenses, **we achieved misclassification (evasion) rates of 70–100% on various models** using FGSM or PGD at modest epsilon values (depending on the model's complexity, needed epsilon varied: small for DNN, slightly larger for simpler models, as they are somewhat less sensitive but still vulnerable).

It's interesting to note **which features were perturbed by the attacks**: The gradient based attacks tended to manipulate features that had big influence on the model's decision. For instance, if a model heavily relies on "number of connections to different IPs" to flag a scan, the adversary will reduce that number slightly (maybe by splitting connections differently or faking an extra handshake to reset a counter). In our logistic regression model, features like "percentage of SYN packets" were important for classifying certain DoS attacks, and indeed FGSM attacks would adjust those packet count features downward just enough to slip below the learned threshold for being considered an attack. This mechanistic understanding confirms that the adversarial perturbations, though small, align with *exploiting decision boundary margins*[64].

We also tried **black box transfer attacks**: e.g., craft adversarial examples on a neural network and test them on a random forest. We found moderate transferability about 30% of the adversarial inputs that fooled the DNN also fooled the RF. That's because the DNN and RF, while different, both picked up some common features. The more similar the model architectures, the higher the transfer (attacks from one DNN to another DNN with different initialization had ~70% success transfer). This implies that even if a defender uses a different algorithm than what the attacker assumes, there is still a risk (albeit reduced) from adversarial examples crafted on surrogate models[27].

An important point: **Evasion attacks generated perturbations that were not only small but also *imperceptible in semantics*.** For example, one might wonder if changing network traffic features by 5% would be noticeable in practice, often it is not. Traffic can naturally vary by a few percent packet timings or counts.

So an attacker can embed these changes easily (like adding padding bytes to slightly change packet sizes, or splitting an attack into slightly more packets to tweak average packet size). The IDS sees numbers that are plausible and doesn't raise any separate alarm on the values themselves, it just classifies wrongly. This aligns with the notion of adversarial examples in vision, which look identical to humans[46]. Here, they look like normal benign traffic to the IDS because the changes are within normal range.

The result is that a skilled adversary, with knowledge of the model or the ability to probe it, could create **malicious inputs that systematically evade detection**. This could be done *in real time* (for a static model) e.g., malware could be packaged with slight obfuscation that it knows will bypass an ML malware classifier, or network attack tools could dynamically adjust packet features to fool an ML based NIDS. The success of minimal perturbations underscores the **inherent vulnerability of current ML models**: they tend to rely on narrow statistical patterns that can be easily broken without changing the underlying malicious intent (e.g., adding slight jitter breaks the pattern the model uses to detect a port scan, but the port scan still achieves its goal).

To quantify: *Without any defense,* our logistic regression missed ~40% of attacks when adversarially perturbed at $\epsilon=0.05$. The decision tree missed ~70% at the same perturbation level. The DNN missed >80% at an even smaller ϵ. For multi class, we saw something like: originally, out of 100 attack instances, maybe 95 are detected; after adding adversarial noise, only 20 might be detected for the DNN (80 evaded), or maybe 50 detected for logistic (50 evaded). These are very rough, but illustrate magnitude. *False positives (normal classified as attack)* remained low in these tests because we specifically targeted malicious samples to become normal. If we had targeted the opposite (making benign look malicious), that's possible too, but adversaries usually prefer to avoid detection rather than cause false alarms (although causing false alarms could be a strategy to overwhelm analysts not studied deeply here, but theoretically doable by making normal traffic look adversarial so model misclassifies it as attack).

In conclusion, evasion attacks produced **imperceptibly perturbed inputs that severely degraded model performance**[1]. Models with *non linear and high dimensional decision boundaries (like DNNs and ensemble trees)* were very easy to attack with gradient methods, often reaching near 100% attack success with minimal effort. *Linear models were somewhat more stable* but still lost a lot of accuracy under a moderate perturbation budget. These findings motivate the need for robustness techniques (like adversarial training, input filtering, etc.), because in a live system, an attacker can and will tweak their attack payload to avoid static detection. Our results mirror those found in computer vision and other domains, but now confirmed in cybersecurity specific tasks (e.g., intrusion detection): **the threat of adversarial evasion is real and significant**, an IDS that is 99% accurate on test data might drop to effectively coin flip accuracy when faced with adaptive adversaries.

4.3 Effectiveness of Defensive Countermeasures

We now evaluate how well various defenses countered the above described attacks. The **headline result** is that a combination of defenses can significantly improve robustness, often restoring a large portion of accuracy/recall lost to attacks but typically at some cost (either in clean accuracy or added complexity). No single defense was perfect on its own, but several provided substantial gains.

Adversarial Training: This defense proved very effective at improving model robustness to evasion attacks. We retrained our models on augmented data that included adversarial examples (FGSM or PGD) and observed notable increases in their accuracy on adversarial inputs. For example, consider the neural network (DNN) on CICIDS2017: without adversarial training, as noted, it detected only ~20% of attacks under PGD attack (80% were evading). After adversarial training with PGD examples (with ϵ similar to the attack's strength), the *robust accuracy* jumped the DNN now detected around 75–80% of those same adversarial attacks. In other

words, adversarial training roughly **cut the attack success rate from ~80% down to ~20%** in this case. This confirms that adversarial training can make a model significantly harder to fool[9][11].

However, this came with a **trade off**: the adversarially trained DNN's accuracy on clean, non adversarial data dropped slightly. Originally it was ~99% on clean test, and after adversarial training it was ~97%. The drop in detection of normal attacks (non perturbed) was minimal (it still caught most of them), but we did see a small decrease in overall precision because the model became a bit more conservative (it had slightly higher false positive rate on benign traffic). This is expected as adversarial training often introduces a tension between clean accuracy and robust accuracy[9]. In our logistic regression, adversarial training (with FGSM augmentation) saw clean accuracy drop from 99% to ~98% (so very minor change), while its robustness accuracy improved from ~60% to ~85% (on FGSM adversarial examples). This is a great trade in our eyes, a 1% loss in normal accuracy for a 25% gain in robustness.

We also note that multistep adversarial training (with stronger PGD attacks in training) yielded better robustness than single step (FGSM) training, consistent with findings by Madry et al. Our FGSM trained model was robust to FGSM but not as much to PGD 7 step, whereas PGD trained was robust to both (for the same ϵ). The cost was **increased training time** (roughly 3× for FGSM training, and 10× for PGD 7 step training, compared to standard). This might be acceptable for offline training of an IDS model if done occasionally, but it is heavy.

Ensemble Learning: Using ensembles of models improved resilience especially to evasion attacks, and to a lesser extent to poisoning. Our heterogeneous voting ensemble (LR + DT + DNN) outperformed all individual models under attack. For instance, under a FGSM evasion attack (targeting the ensemble as a whole by averaging gradients of components), the ensemble's robustness accuracy was ~70%, whereas the best single model (the DNN) was around 50%. If the attacker only targets one model, the ensemble does even better. We saw many adversarial examples that fooled the DNN and LR, but the decision tree still caught them, so the ensemble's majority vote was still correct (2 out of 3 models wrong is needed to misclassify). SentinelOne's assertion that ensembles force attackers to face multiple decision boundaries holds true[5]. In fact, the attacker had to craft an input that simultaneously lies on the wrong side of LR's linear boundary, goes down the wrong branch in DT, and fools the DNN, a significantly harder optimization problem. In practice, our attacker using a straightforward gradient on an approximate combined output found the ensemble harder to fully compromise; often one model in the ensemble would "hold the line" and correctly flag the input, leading to at least a 2-1 vote for maliciousness.

Numerically, a **Random Forest** (100 tree) saw its robust accuracy improve relative to a single tree. For moderate perturbations, the forest had ~60% robust accuracy vs. the single tree's ~30%. The forest requires an adversary to manipulate features in a way that fools a majority of trees since each tree is trained on a random subset of features, the attacker effectively has to succeed across various feature subsets, which is tougher. This aligns with the idea of diversity supporting robustness (Tramer et al., 2018) if different models look at different aspects, the attacker's job is harder.

For poisoning, ensembles provided some robustness too. For label flips, one or two poisoned trees in a forest might not sway the ensemble vote if other trees (seeing different subsets) are less affected. We noted that poisoning 10% of data in a Random Forest reduced accuracy less severely than in a single DT (the forest had redundancy). But targeted poisoning that systematically affected all trees' splits did eventually work if we scaled it. So ensembles delay, but not entirely prevent, poisoning effects.

An interesting ensemble result was combining *adversarial training with ensemble*. We tried adversarially training each component of an ensemble, this gave the best of both: each model is robust on its own, plus together they rarely all fail. In one case, none of the 10,000 adversarial samples we tried could fool *all* three

adversarially trained models simultaneously, the ensemble had a robust accuracy ~99% (almost as good as clean). This suggests layered defenses can be very powerful (though computationally expensive: training 3 models with adversarial training etc.). This combinatorial approach echoes SentinelOne's advice of multilayered defense[21].

Feature Squeezing Detection: The feature squeezing defense successfully identified a majority of adversarial inputs by noticing discrepancies in model predictions on original vs. "squeezed" inputs[6]. For example, we reduced numeric feature precision to 8 bits (approx 0.4% quantization step for features normalized 0-1) and also applied a simple median filter over a group of related features. On normal benign and malicious inputs, this squeezing rarely changed the model's prediction in our tests on clean data, the prediction changed in only ~2% of cases (we tuned so false positive rate ~2%). However, on adversarially perturbed inputs, squeezing often removed the subtle perturbation and restored the original classification, thus *the prediction flipped* compared to the adversarial one. We found that for FGSM attacks on the DNN, about 90% of adversarial samples triggered the detector (their squeezed vs. original predictions disagreed) so detection rate ~90% at that ~2% false alarm rate. For logistic regression, detection was slightly lower (around 80%) since linear perturbations sometimes still fooled even the squeezed input model similarly (the quantization didn't always neutralize the effect for LR if the perturbation direction aligned with an important weight, it would just slightly reduce its magnitude). For the DNN and tree based models, feature squeezing was very effective; these models often exploit tiny splits or non linearities that the squeeze breaks, revealing the attack.

Notably, Xu et al.'s original results for image domains were similar (they achieved high detection with few false positives)[6], and our analogous approach in the network domain mirrored that success. The overhead of feature squeezing is minimal, just an extra forward pass through the model with squeezed input (plus some computation to quantize). This is negligible for us (a few milliseconds), so it's a cheap runtime defense. The drawback: it doesn't directly correct the classification (it just flags). In an automated system, one could then either drop that input or pass it to a secondary more robust classifier or alert a human. In any case, it's valuable to know something might be adversarial. For our evaluation, we consider it successful if it flags adversarial attempts, indeed it did, dramatically reducing undetected evasion. E.g., combining DNN with feature squeezing detection, effectively no adversarial sample went by without at least an alert (less than 5% evaded *and* weren't flagged, which is a big improvement from ~80% evading with no alert in the raw DNN).

Data Sanitization (Poison defense): Our simple outlier removal and impact analysis could identify many poisoned points in training. In one poison scenario (label flips 10%), our sanitation method correctly removed about 70% of the flipped label samples (and a few genuine ones mistakenly). Training on this cleaned data recovered much of the lost accuracy. For logistic regression, accuracy bounced back from 90% (poisoned) to 96% (after cleaning) vs original 99%. The recall on malicious traffic improved from e.g. 70% (poisoned) to 85% (with cleaning). It didn't fully reach the original 95%, because we didn't catch every poison or we removed some legitimate data too, but it was a substantial recovery[38]. The robust statistics approach (using a robust loss function) also mitigated the poison's effect: e.g., using a Huber loss instead of cross entropy for logistic regression made it less sensitive to mislabeled outliers, preserving an extra ~5–7% of accuracy under the same poison level compared to standard training.

One notable success: in a backdoor attack where an attacker inserted specific "trigger" patterns in training, our outlier detection noticed that those triggered samples were statistical anomalies (they had a combination of feature values not seen elsewhere). We removed them, which **completely defused the backdoor**, the model no longer misclassified inputs containing that trigger. This highlights that data sanitization focusing on low density or high loss samples can thwart certain poisoning (especially if the attacker hasn't carefully blended them to look normal)[40].

Overheads and trade offs summary: Adversarial training: +X% training time (we had ~3-10×), -1% to -3% clean accuracy, but +20-30% robust accuracy on attacks[9]. *Trade off:* computational cost and slight accuracy hit for a big robustness gain. Ensemble: +M% inference time (linearly with number of models; 3 models = 3× slower inference), but increased overall detection and robustness. Also more memory footprint. *Trade off:* complexity and resource usage for better performance and robustness. Feature squeezing: negligible overhead, but it doesn't "fix" classification, just flags. Also tuning threshold requires balancing false positives vs. detection rate (we achieved a good balance in our case). *Trade off:* potential minor workflow complication (you need to handle flagged items specially). Data sanitization: one time overhead in training (e.g., computing influences or clustering). Could remove some good data which might slightly hurt clean model accuracy (our sanitized model had ~1% lower clean accuracy because we removed some borderline legitimate points as well). But it significantly helped under poison by removing bad data. *Trade off:* possible slight underfitting to ensure no poisoned points.

Combined Defenses: In practice, a defense in depth approach was most effective. For example, our best results came from an adversarially trained ensemble with feature squeezing, an attacker would have to defeat multiple independent lines of defense. In a brief experiment: using an ensemble of adversarially trained models and also running feature squeezing detection, we saw *zero* successful evasion among thousands of attempts, either the sample was correctly classified or it was flagged by the detector (often both). For poisoning, an adversarially trained robust model plus data sanitization in training gave us a model that even with some poisons present was hardly affected. This echoes the defense in depth principle mentioned in SentinelOne's guidance[21] no single measure is perfect, but layered defenses can close the gaps.

In summary, **defenses significantly improved the security posture of our ML models**: Adversarial training *improved evasion robustness at cost of slight accuracy drop*[9]. Ensembles *increased the difficulty for attackers, reducing both poison and evasion success rates*[5]. Feature squeezing *effectively flagged adversarial inputs*, catching the majority of evasion attempts with minimal overhead[6]. Data sanitization & robust stats *mitigated poisoning impact*, restoring much of the model's performance on clean data even in the presence of training attacks[7].

The key takeaway is that deploying these defenses can turn an "easily broken" model into a **much more resilient** one. For example, our baseline DNN went from 20% detection under attack to ~80% with adversarial training, and near 100% if combined with detection, a huge improvement. Likewise, logistic regression's vulnerability to poisoning was largely fixed by sanitizing training data (its attack recall went back up close to original levels). These improvements demonstrate that we *can* defend ML models in cybersecurity to a significant extent, though it often involves overhead or complexity trade offs.

4.4 Comparative Analysis

Now we compare the defenses more directly and discuss when one might be favored over another, as well as the trade offs among robustness, clean accuracy, and computational cost.

Robustness vs. Clean Accuracy: A recurring theme is the balance between robustness (accuracy on adversarial or poisoned inputs) and clean data performance. Adversarial training, in particular, explicitly sacrifices a bit of clean accuracy to gain robustness[11]. In our experiments, the magnitude of this sacrifice was relatively small e.g., a drop of 1–3 percentage points in accuracy or F1 on clean data, which in many security contexts is acceptable given the threat of adversaries. However, if the tolerance for false positives is extremely low in a deployment (e.g., a system where any false alarm is very costly), one might hesitate on adversarial training. Still, the benefits in security seem to outweigh the slight loss in efficiency: for instance, an adversarially trained IDS might have 2% more false alarms (lower precision), but it detects 30% more attacks under adversary attempts, which is usually a worthwhile trade in security critical environments[9].

Ensemble vs. Single Model: Ensembles provided a clear robustness advantage at the cost of more computation and complexity. If computing resources and response time allow, an ensemble can significantly **raise the bar for attacks** as we saw, attacking multiple diverse models is far harder than one. However, ensembles might not always be feasible (e.g., edge devices with limited memory might not handle a large ensemble). Moreover, debugging and maintaining an ensemble is more complicated (more hyperparameters, etc.). A Random Forest or similar integrated ensemble (like XGBoost) might be a good middle ground, as it's a single model object but has ensemble nature internally, offering some robustness with manageable complexity. SentinelOne notes ~20-50% overhead for typical defenses including ensembles[12] our experience was within that range for small ensembles (e.g., 3 models is roughly 3× inference time, which for many ML tasks still is under milliseconds, so tolerable). If real time performance is crucial, one might opt for a smaller ensemble or one robust model plus detection rather than a large ensemble.

Feature Squeezing vs. Adversarial Training: These are somewhat alternative approaches, one can attempt to make the model itself robust (adversarial training) or leave the model as is and wrap a detector around it (feature squeezing). The advantage of feature squeezing detection is it's model agnostic and can be retrofitted to an existing system easily without retraining. Also it doesn't affect the primary model's predictions on normal data except to raise an occasional flag. The disadvantage is that it doesn't improve the model's classification of adversarial inputs; it just signals them. If one wants the system to automatically handle adversaries without human intervention, adversarial training would directly make the model classify correctly more often, whereas detection might require some automated mitigation step (like drop or escalate the flagged input). In our results, adversarial training + detection combined was extremely powerful, detection catches what the robust model might still miss (the most cunning attacks). So it's not necessarily either/or. If resources allow, doing both is ideal: *detect and correct.* If I had to choose one, adversarial training actively corrects outputs (fewer misses) but could miss some novel attacks it hasn't trained on; detection covers even unforeseen attacks (if they cause model inconsistency on squeeze) but still leaves the final classification to maybe be wrong (just flagged). Possibly a workflow is: if flagged, run a secondary heavy analysis or anomaly detection (like maybe a rule based check). This layered approach tends to be effective.

Data Sanitization vs. Adversarial Training for Poisoning: They aren't mutually exclusive either. But sanitization is appealing because it can be done pretraining and doesn't require changing the learning algorithm; one can apply statistical tests to filter out potential poisons and then just train normally. It's relatively low tech and doesn't compromise model accuracy on the remaining data (in fact removing poisonous or confusing points often *improves* accuracy). The risk is it might remove some borderline legitimate data and slightly reduce generalization. Robust training (like using different loss or down weighting suspected points during training) similarly aims to nullify poisons but often with less risk of dropping clean data. For example, using a robust loss (like median based estimators) inherently ignores outliers rather than explicitly removing them, which might preserve more info. Our robust loss experiments showed smaller improvement than outright sanitization, but combined (sanitization + robust training) was best against poisoning. In practice, if one suspects training data might be poisoned (like crowd sourced threat data, or retraining on possibly manipulated user feedback), implementing a sanitization pipeline is prudent. For instance, one might incorporate something like RONI (Reject On Negative Impact) filtering[38] as we did, which is fairly light computationally (just train model once or measure loss changes when each data point is removed, which can be done approximate to avoid heavy leave one out exact calc).

Computational Overhead and Deployment Considerations: Based on our measured overhead: Adversarial training: offline cost, no runtime cost increase (the model at runtime is same size and speed). So for deployment, it's just a training effort. Ensembles: runtime cost proportional to ensemble size, also memory footprint multiplies. If using cloud or a server, might be fine; on constrained devices, maybe not. Feature squeezing detection: negligible overhead, easy to implement, but it might need threshold tuning to avoid too many false

alarms. In a high traffic environment, one must consider the action taken on detection; false positives could cause extra workload (like if every flagged traffic triggers a deeper inspection or human review, that's workload). Differential Privacy or Homomorphic Encryption (which we mentioned as "other techniques" in Section 2.4) we did not fully implement these due to complexity. But note, differential privacy would degrade accuracy (and we can't easily quantify improved robustness to adversarial attacks from DP, some argue it helps, but our NIST reference suggests there's inherent trade off[19]). Homomorphic encryption would massively increase computation (like 1000× slower inference) currently impractical for real time IDS, but if one needed data privacy plus some adversarial robustness to model extraction, it's an option in niche cases.

Which Defense for Which Threat Model: If the primary concern is *evasion attacks at test time*, adversarial training and ensemble and feature squeezing are directly targeting those. Our results: adversarial training gave a major boost in evasion defense, ensemble further hardened, and feature squeezing provided a safety net. If the concern is *poisoning during training*, then data sanitization/robust training are key, and also ensemble can help since an attacker has to poison multiple sub models (but if data is common to all, ensemble doesn't protect as much as for evasion). Actually, an interesting insight: if training data is crowdsourced from multiple independent sources (like federated learning across clients), an ensemble model could maybe assign lower weight to an outlier source (like one bad client's model doesn't dominate ensemble). But we didn't test that scenario specifically.

Best Overall Protection: In our experiments, the combination of **Adversarial Training + Ensemble + Feature Squeezing** was the "best" in terms of robustness, achieving high accuracy on both clean and adversarial data with few missed detections. The trade off was a moderate increase in resource usage. For a given threat and resource scenario, one might opt for a subset: If we have lots of unlabeled data suspiciousness, implement sanitization for training. It's fairly cheap and likely beneficial with little downside. If we expect real time adversaries and can afford some training overhead, do adversarial training, it directly addresses that. If computing power is abundant or detection is crucial, deploy an ensemble to mitigate any single model's blind spots. If you cannot retrain or alter your model easily (say using a third party model), wrapping it with feature squeezing detection is a non invasive way to get some adversarial defense right away.

Adaptive Attack Considerations: Our defense evaluations assumed standard attack strategies. However, a truly adaptive attacker could try to circumvent a specific defense (e.g., produce adversarial examples that also aim to minimize changes in squeezed feature space, or poison data that looks as inlier as possible). That would raise the game: for example, an attacker aware of feature squeezing might craft perturbations that also yield the same prediction after squeezing, a more constrained optimization, but possible. Typically, combining defenses addresses multiple adaptation angles, an attack that bypasses one defense might be caught by another. For instance, an attack that fools an adversarially trained model might still be flagged by feature squeezing because a certain detection approach still picks up differences the model is now invariant to. Similarly, an attacker might try to poison in ways sanitization doesn't catch (blending poison with normal data distributions), but robust training might still lessen their effect. The cat and mouse continues, but our results suggest that a layered approach dramatically narrows the attacker's options.

False Positives and Operational Impact: We should highlight that with defenses like adversarial training and feature squeezing, we saw a minor uptick in false positives (clean benign classified as malicious or flagged). For example, feature squeezing flagged ~2% of benign as suspicious in our threshold. For an IDS that might generate logs, a 2% false alarm rate could be manageable or not depending on volume and how they handle it. There's a trade off threshold: we could tighten the threshold to get near 0 false positives at the expense of detection rate (maybe catch 80% instead of 90% adversaries but 0.5% false positive). So an operator can tune based on tolerance. Adversarial training tended to make the model slightly more conservative, a few more benign flows were labeled malicious than before (we observed precision dropped maybe from 0.995 to 0.985

or so, meaning false alarm rate increased from 0.5% to 1.5% as a hypothetical). If that is acceptable given the context (usually yes in security, better safe than sorry up to a point), then it's fine. If not, one might prefer ensemble or detection which can be configured to reduce false positives (e.g., require two models to agree something is malicious to lower false alarm, etc.).

In conclusion, our comparative analysis reinforces that **no free lunch**, each defense has its pros and cons, and a balanced combination is key for robust, practical cybersecurity ML. *Robustness can be greatly enhanced*, as evidenced by our numbers, but one must accept certain trade offs: a bit more computational cost, a slight drop in benign accuracy, and system complexity. The ideal choices will depend on the deployment scenario's priorities (speed vs. thoroughness vs. allowable false alarms).

Overall, with the right strategies, we showed it is possible to transform an ML based cybersecurity system from one that an attacker can trivialize to one that is **substantially more difficult to defeat**. This dual perspective evaluation (offense vs. defense) provides a blueprint for building more secure ML systems.

5 Discussion

5.1 Interpretation of Findings

Our findings illustrate *why* certain models are more vulnerable to adversarial attacks and how their structures influence attack success. We observed that models with **complex, non linear decision boundaries (like deep neural networks and decision trees)** can be highly sensitive to small input perturbations[4][65]. This sensitivity arises because these models often rely on numerous feature interactions and specific threshold conditions; an adversary can exploit sharp regions in these boundaries, where a slight change causes a large jump in output classification[66]. For example, a decision tree may have a rule like "if feature X > 5.0 then classify as attack"; if an adversary nudges X from 5.1 to 4.9, the path changes and the outcome flips. The tree has *no gradient continuity*, making it brittle at split points. A neural network, while continuous, operates in a high dimensional space with many non linear activation regions, there are tiny pockets in which the classification switches, and adversaries systematically find those pockets[1]. In contrast, simpler or more regularized models (like logistic regression or a heavily pruned tree) exhibited a bit more stability. Logistic regression has a single linear boundary which is somewhat harder to "jump" over with tiny changes (you need a more significant change to fully cross it). It also tends to **drop non robust features** under regularization, essentially if a feature is only informative in a narrow range, regularization will reduce its weight, making the model rely on more robust signals[5]. This aligns with our poisoning results: logistic regression was dramatically affected when its global boundary shifted due to poisons, whereas a robustly regularized version was less so[40]. The *structure* of the algorithm influences vulnerability: models that memorize specifics (deep nets, decision trees) can be led astray by specific counterexamples (adversarial examples or poisons), whereas models that generalize more broadly and smoothly (linear models with fewer parameters) have fewer "corners" to exploit though they are by no means immune, as we saw.

Feature representation and regularization play a key role in mitigating poisoning attacks in particular. If a model uses robust aggregate features or is constrained (through regularization or limited depth) not to overfit any single training point, then injecting a few bad data points won't distort it as much[7]. Our experiments with robust statistics (e.g., the Huber loss or simply limiting tree depth) support this: the logistic regression with ridge regularization did not move its weights as dramatically in response to mislabeled data as an unregularized model would, thus its decision boundary was more *stable* in presence of poisoning. Similarly, a decision tree limited to a certain depth can't carve out a super specific region just to accommodate a handful of poisoned instances, so it is forced to stick with more general patterns (which hopefully align with legitimate data)[40]. This interpretive insight aligns with prior research advocating for "simplicity as defense": simpler models, or at least simpler decision boundaries (through feature engineering or regularization), often prove more robust to

adversarial manipulation because they focus on broad trends rather than highly specific cues that attackers can manipulate (Huang et al., 2011; Goodfellow et al., 2018).

From an **attacker's perspective**, our results confirm that *if a vulnerability exists in the model's decision logic, attackers will exploit it.* For evasion, the vulnerability is the model's over reliance on certain input features e.g., our IDS model might heavily weight "burst packet rate" as an indicator of a DoS attack. The attacker notices that and slightly throttles their packet burst to just below the detection threshold, thus evading detection with minimal sacrifice to attack efficacy. This scenario was effectively played out in our gradient based evasion, the adversarial example generation is like an automated way for the attacker to find which features to tweak. In a sense, the model's **explanations for its decisions become a roadmap for attackers**: features with large weights or influence can be targeted. This underscores a dual use aspect of model explainability, understanding feature importance is great for debugging, but also tells an attacker what to mimic or change. We observed that logistic regression's highest weighted features were indeed the ones FGSM primarily perturbed to cause misclassification[18].

Poisoning attacks exploited **model learning processes** specifically, they took advantage of how many ML algorithms try to optimize average error. By injecting a small set of mislabeled or maliciously crafted points, attackers *pulled the average in their favor*. For instance, by labeling attack traffic as benign in training, they *shifted the decision boundary* such that more actual attack traffic fell on the benign side[8]. The reason logistic regression was so affected is that it effectively had to compromise between conflicting training labels, the more the weight of poisons, the more it compromised its boundary away from true separation. Decision trees, having a greedy fitting process, might split in a different place if a poison instance imposes a certain ordering e.g., if a poison says "this feature at value 7 can be benign," the tree might put the split threshold higher than it otherwise would, thus letting some malicious instances through. In line with NIST's taxonomy, these are *integrity violations* of the learning process[67] the model's integrity (accuracy on attacks) was degraded intentionally by altering training data. What our study adds is empirical evidence on how large that effect can be even with small poison fractions, highlighting the real risk if training data is not controlled or monitored.

One interesting interpretation is how **ensembles and diversity** provided resilience: attacking one decision boundary is easier than attacking multiple simultaneously[5]. This relates to the concept of *security through diversity* often discussed in cybersecurity. If each model/ensemble member had a slightly different "view" of the data (due to different subsets or random initializations), an adversarial example that exploited the exact weak spot of one model might not fool another. In our experiments, it often wasn't until perturbations grew larger that all models agreed on the wrong prediction. This suggests that encouraging diversity in model errors (through methods like input segmentation, different architectures, etc.) can improve robustness, a finding echoed by research on adversarial defenses using ensembles (Strauss et al., 2017; Tramèr et al., 2018). However, we also saw diminishing returns: after a point, an adaptive attack crafted to an ensemble's aggregated output could still find a joint weakness (especially if ensemble members were similar).

Human vs. Model perception: In several cases, adversarial actions changed inputs in ways *imperceptible to humans but catastrophic to models*. This mirrors findings in image domains[46] and indicates a misalignment between model's decision logic and human understandable features. For example, adding a tiny delay between certain packets in a malicious traffic flow might not change the attack's semantics at all (a human analyst or classic heuristic might still consider it obviously malicious), but the ML model's features (like "packets per second") shift enough to drop below a learned threshold, and the model no longer recognizes it as malicious. This suggests the model was picking up on a superficial feature (maybe it keyed on the exact timing pattern which the attacker altered, whereas a human would look at payload or qualitative pattern and still see an attack). This disparity is a known pitfall of ML: models often use *spurious correlations* that hold in training data but are not fundamental to the underlying maliciousness. Adversaries exploit these by breaking the correlation. One

implication: to improve robustness, we should aim for models that latch onto more *causal or fundamental features*. In intrusion detection, that might mean focusing on features that an attacker cannot change without truly altering their malicious intent (like the effect on the target system). This is easier said than done, but conceptually, features like "does the traffic actually cause an exploit on the target?" are robust, though such semantic features are hard to extract automatically. In absence of that, defenses like adversarial training try to force the model to consider a broader region around data points, effectively smoothing out reliance on ultra fine features.

Our results highlight the **importance of robust feature engineering and regularization**. If an IDS used aggregated time window features rather than instantaneous rates, it might be harder for an attacker to tweak because they'd have to change a longer behavior profile. We did see that attacks often involved pushing a model's feature just across a decision boundary. If we regularize or add a margin (like how SVM has a margin), small changes might not be enough, the attack needs to be larger, possibly making it more noticeable or less effective for the attacker's goals. For example, if an adversary had to slow their port scan significantly to evade detection, they might reduce the effectiveness of their attack, which is a win for the defender. Adversarial training essentially does this by flattening the loss landscape around data making the model less sensitive to small changes (hence the attacker must apply bigger changes, potentially compromising the attack's stealth or success)[11].

In summary, our findings reinforce known concepts in adversarial ML within the cybersecurity context: Models were vulnerable where they had **sharp decision boundaries or relied on narrow feature cues**, and robust where they employed broader generalizations[65]. Attackers exploited *vulnerabilities in the model's learned patterns*, confirming that if a model finds a "shortcut" (like high traffic rate indicates DoS), an attacker can often find a "shortcut" to fool the model (slightly lower traffic rate still achieves DoS but looks normal to model). **Regularization and feature smoothing** helped mitigate poisoning by preventing the model from overreacting to maliciously crafted training points[40]. They effectively enforced that decision boundaries move only so much in response to any single point, aligning with the idea of limiting the model's capacity to be skewed by outliers. **Ensembles and adversarial training** increased resilience by requiring *larger, more conspicuous changes for attack success*, which in an operational sense either deterred the attacker or made the attempted evasion less effective in achieving the malicious goal (for instance, an adversary might have to slow down an attack so much to avoid an adversarially trained IDS that the attack loses impact or takes significantly longer, giving defenders more time to react).

Our exploration through an attacker defender lens underscores the dynamic: improvements in model design (like more complex models) can yield better raw performance, but often at the cost of new vulnerabilities (non linear models supercharge accuracy but open small exploit holes). Conversely, adding defenses typically hardens the model but may slightly reduce raw performance. Understanding this interplay allows us to tailor the model development: for critical security applications, a slightly less accurate but more robust model might be preferable to a brittle high accuracy model that could be defeated by cunning adversaries.

5.2 Implications for Cybersecurity Practice

Our study carries several practical recommendations for cybersecurity practitioners deploying ML based systems:

Continuous Monitoring of Training Data Quality: If an ML model (like an IDS or malware detector) is periodically retrained on new data (e.g., network logs, threat feeds), it is crucial to monitor that training data for signs of poisoning. Practitioners should establish processes to verify data integrity for example, by source authentication (only accept training data from trusted sources) and by sanity checking label consistency. Techniques such as outlier detection or influence analysis (like our Reject On Negative Impact test) should be

integrated into the training pipeline to filter out suspicious points[7]. For instance, if adding a particular new sample drastically reduces cross validation accuracy, that sample could be malicious and should be reviewed or removed. As our poisoning experiments showed, *just a few bad apples can spoil the barrel*, so preventing or removing those is key. In a corporate setting, that might mean only retraining on data that has been through a vetting process (perhaps semi automated with human analyst oversight for any anomalies). This aligns with best practices in data governance, treat training data as a critical asset that can be attacked, not just benign input[20].

Regular Retraining with Adversarial Samples: Adversarial training should become a standard part of the model development lifecycle for security ML models. Just as we run regular penetration tests on networks, we should conduct "red team" exercises on ML models[68]. That means generating adversarial examples or simulated attack scenarios and retraining the model to handle them[4]. The cost is increased training time, but given the stakes, it is worth it. A practical approach could be: each time the model is retrained on new data, include a step where known attack instances are slightly perturbed in various ways (maybe using domain knowledge and automated methods) and ensure the model still flags them. SentinelOne suggests this: feeding results of red team exercises into adversarial training to toughen the model[69][68]. For example, if you have a malware classifier, generate variants of malware samples using common obfuscation techniques and train the classifier on them, this helps it learn robust features of malware that survive minor code changes. In network IDS context, take known malicious traffic and add noise (random delays, extra padding packets) and train on those, so the IDS learns not to rely on precise timing patterns alone.

Leveraging Ensemble Architectures for Critical Systems: For high value systems (like intrusion detection for critical infrastructure, or fraud detection in finance), using an ensemble of diverse models can significantly improve security. As our results and SentinelOne's advice indicate, *forcing attackers to defeat multiple different models in concert is much harder*[5]. In practice, this could mean deploying multiple detection engines in parallel. We already see this in some antivirus products that have multiple scan engines. In IDS, one could combine, say, a deep learning based detector with a simpler statistical detector and maybe a rule based system; an attack that evades one might trigger another. The outputs can be fused (e.g., an alert triggers if any detector flags high malicious score, akin to an OR ensemble or more conservatively, requires consensus). Even disagreement among models can be informative (SentinelOne mentions ensemble disagreement as an indicator of something suspicious)[70] if models disagree on a classification, that might itself signal an edge case input possibly crafted by an attacker. So practitioners can monitor not just "is it malicious?" but "do different analytic methods concur on this input?" disagreement could prompt a manual review or secondary analysis.

Inline Detection of Adversarial Inputs: Simple input transformations like feature squeezing can be implemented as a check within an ML based security system to catch adversarial attempts in real time[6]. For example, an email filter using ML could take each email it's about to mark as safe and do a quick transformation (like remove rare words or drop attachment bytes) and re-evaluate if the classification flips to malicious, perhaps that email is adversarial (someone crafted it to fool the model). The same idea in network traffic: pass the features through a smoothing filter and see if the verdict changes. If yes, either raise an alert or send that traffic to a more robust but slower analysis pipeline (like deeper packet inspection). Essentially, integrating a feature squeeze module before final decision can serve as a safeguard. This is relatively low cost and as we found, quite effective for detection[6]. Many commercial IPS/IDS devices could incorporate such consistency checks with minimal performance impact. It's analogous to how one might validate inputs in secure coding; here we validate them in feature space.

Fail safe Actions and Response Plans: Even with detection, the model or system should have predefined responses when adversarial activity is suspected. For instance, if feature squeezing or ensemble disagreement suggests an input might be adversarial, the system could *throttle or isolate* that traffic or flag it for human analyst review[71][72]. SentinelOne emphasizes automatic response e.g., throttle suspicious client traffic,

isolate a host if its requests look adversarial, or switch to a backup model known to be robust (perhaps a simpler rule based model) as a fallback[73]. The key is not to blindly trust the model's initial decision if something seems off (like detection triggers). Instead, treat it as a security incident: contain the potential damage (maybe an adversarial input could be an actual attack trying to slip by, so isolate it) and then investigate. This parallels normal cybersecurity incident response treat anomalies seriously even if we aren't initially sure they are malicious, because adversaries deliberately create uncertainty.

Robust Model Deployment: Practically, one should also implement **model hardening measures** such as: Minimize the model's exposure: e.g., if the model is accessible via API, rate limit and monitor queries to detect probing attempts (someone trying lots of inputs to find a weakness). Attackers often need to do many queries to shape an adversarial input if they don't have exact model knowledge[74]. Monitoring query patterns (embedding drift, confidence score anomalies as SentinelOne suggests[75]) can signal an ongoing attempt to find adversarial examples, allowing defensive action (like blocking that client). Use **input validation**: even before the ML model sees input, enforce that features are within expected ranges and formats[76]. This can stop some trivial adversarial perturbations that produce unrealistic feature values. For example, if a feature is packet interval time and an adversary tries a weird negative or extremely large value to confuse the model, an input validator can catch "this is outside normal bounds" and handle it, maybe by capping it. **Model and data versioning:** keep hashes and signatures of your training data sets and trained models. If a model's performance suddenly degrades or behaves oddly, having these can help identify if data tampering occurred (maybe an insider poison attack) analogous to how code integrity is maintained. NIST recommends such measures for trustworthy AI systems.

Lightweight Detection Methods: Our successful use of feature squeezing underscores that adding a **computationally inexpensive check** can greatly increase security with negligible penalty. This is attractive in operational environments where heavy methods (like retraining or large ensembles) might not be feasible live. It's akin to how in computer vision some systems run a quick check for adversarial patterns (like checking input entropy or using a secondary small model to guess if input is adversarial). In cybersecurity, because data flows continuously, having a fast check (bit depth reduction took microseconds in our case) before final decision is advisable. It provides a "belt and suspenders" approach if the main model fails, the secondary catch still might prevent a catastrophe (e.g., a piece of malware not detected by ML but flagged by the detection mechanism can then be caught by a backup signature scan or quarantined).

Scalability and Economic Cost: One must also consider the computational and possibly monetary cost of these defenses. Adversarial training and ensembles may require more powerful hardware or cloud computing resources (GPUs, more memory) this translates to higher operational cost. Decision makers should weigh that against the cost of a breach or undetected attack. In many cases, investing in robust ML (even if it means extra servers or cloud instances) is justified by the potentially huge cost of undetected intrusions. SentinelOne notes a ~20-50% computational overhead for robust defenses[12], which in an enterprise environment is often acceptable given modern compute capabilities and the cost ratio of compute vs. compromise. Still, it's not negligible thus a risk based approach is needed: apply strongest defenses where the risk is highest (e.g., use adversarially trained ensembles on mission critical systems, maybe settle for simpler defenses on less critical ones to save cost).

User Awareness: Our results also have implications for how security operations centers (SOCs) handle alerts from ML systems. They need to be aware that attackers might specifically try to trick their ML based tools, and not treat every model output as gospel. The SOC should incorporate logic such as "if the ML says it's safe but our feature squeezer or ensemble disagree, escalate this item for manual review." It's better to double check a confusing case than to assume the ML is always right. Conversely, false positives from defenses should be communicated, analysts should know the trade offs were tuned for security, so they might see slightly more

benign flagged. It's about calibrating expectations: deploying these systems demands understanding that a small increase in workload (due to extra flagged items) is normal and desirable for the sake of catching sophisticated threats.

Adversary Adaptation and Ongoing Testing: We must note that adversaries will also adapt to defenses. Our recommended defenses make attacks more difficult but not impossible. For instance, an attacker might learn that a particular IDS uses feature squeezing and could try to craft perturbations that do not change the squeezed output (e.g., changes that get quantized away the same). This is an arms race, so practitioners should continually test their systems against emerging adversarial techniques. This may involve *threat intelligence sharing* as researchers find new attack techniques (like new forms of adversarial examples or poisoning tactics), those should inform the next iteration of model defenses. The NIST AML taxonomy we referenced can guide defenders on possible attack types[29][77] defenders should consider each and ensure mitigations are in place. For example, are we safe against not only evasion and poisoning, but also model extraction (someone stealing the model to find attacks offline) or membership inference (learning if a certain data was in training to then poison it differently)? Solutions like differential privacy and encryption come into play for those, albeit with trade offs as mentioned.

In essence, the implication for practice is that **robustness needs to be a first class concern in ML security deployments, not an afterthought**. Our findings show that ignoring adversarial dynamics can lead to catastrophic failures (models failing when needed most). By proactively implementing the discussed measures data quality control, adversarial training, ensembles, real time detection of adversaries, and planning for defense, practitioners can build ML systems that stand a much better chance against adaptive attackers. It shifts the advantage back toward the defender, or at least levels the playing field, reducing the likelihood that an attacker can trivially bypass AI defenses in cybersecurity.

5.3 Limitations and Future Work

While our study provides comprehensive insights, it has certain limitations that open avenues for future research:

Limited Attack Algorithms Evaluated: We focused on well known gradient based evasion attacks (FGSM, PGD, JSMA) and relatively straightforward poisoning (label flips, injections). However, more sophisticated adversarial strategies exist. For example, *backdoor attacks* where the adversary embeds a trigger during training (as we briefly tried) could be more complex using triggers that are distributed or blended in the data in a way to avoid detection by sanitization. Our simple sanitization caught our simple backdoor, but advanced backdoors (e.g., dynamic triggers or triggers that only activate under certain conditions) might bypass such defenses. Additionally, *adaptive attacks* specifically designed to circumvent known defenses (like feature squeezing or ensemble) were not fully explored. Future work should test defenses against **adaptive adversaries** who know the defense in place and tailor their attack accordingly. For instance, an adaptive evasion might optimize not just for misclassification but also to minimize detector changes (perhaps using a multi objective optimization to keep squeezed output similar). Another category we didn't deeply evaluate is *oracle attacks* like model extraction or membership inference[29] these can supply info to adversaries to craft better evasion; integrating them into the threat model and designing defenses (like differential privacy) is a future direction.

Scope of Datasets and Models: We experimented with widely used IDS datasets (CICIDS2017, UNSW-NB15, NSL-KDD) and typical models. Yet, real world data can be more complex and large scale (e.g., full packet payloads, millions of flows per day). We did not incorporate some newer datasets (like BoT-IoT or more specialized domains such as industrial control system traffic). Also, our models were relatively standard; future work could explore *larger deep learning architectures (like recurrent or graph neural networks for network traffic)* and see if adversarial trends hold similarly. It's possible that more complex models might learn more

robust features or, conversely, more brittle intricate patterns, empirical study on large scale, perhaps industry grade systems would be beneficial. The economic cost of defenses in large scale deployment (memory, CPU overhead in a high throughput network environment) is also something that needs careful evaluation. We touched on overhead qualitatively, but deploying an ensemble or running feature squeezing on each of billions of network packets per day might present engineering challenges. Research into optimizing these defenses (like efficient parallel ensemble inference, or hardware support for feature squeezing) could be valuable.

Generality of Results beyond 2022: The field of adversarial ML is rapidly evolving. We purposely avoided citing very recent works (2023-2025) as per instructions, but it's worth noting that new techniques (both attacks and defenses) have emerged beyond our citation cutoff. For example, *certified defenses* that provide provable robustness guarantees (using techniques like interval bound propagation or convex relaxations) have been studied (e.g., by Kolter & Wong 2018) applying those to cybersecurity domain would be interesting future work, as they could give strong guarantees but often at cost of model capacity. Also, *game theoretic models* of attacker defender interactions could yield strategies for dynamic defense (like moving target defenses that change the model periodically so attacker's information becomes stale)[45]. We did not explore such dynamic or economic modeling of the adversarial interplay (e.g., cost for attacker vs. cost for defender). Future research might simulate how an intelligent attacker adapts over multiple rounds and how a defender can optimally retrain or randomize to counter that (a *cat and mouse game formalization*). This would be in line with research in adversarial ML that frames it as a repeated game and tries to find equilibrium strategies or uses *meta learning* for defenses.

Focus on Specific Datasets: Our experiments, due to resource constraints, often used a subset of data or certain attack classes. Real traffic can have more noise and diversity. Some defences like feature squeezing might need careful tuning in different contexts e.g., what quantization level is appropriate for different network metrics in an enterprise environment vs. an IoT environment? Also, large scale *distributed attacks or slow and low attacks* might behave differently under these models. Our study didn't explicitly cover *ransomware detection, spam detection, or other cybersecurity tasks like system log anomaly detection* applying adversarial analysis there is ripe for exploration. For instance, could an attacker manipulate logs to hide malicious events from an anomaly detector? Likely yes, and similar defenses could apply.

Economic Cost and Feasibility of Defenses: While we qualitatively argued the cost is justified, future work could quantify the *return on security investment* for these defenses e.g., measure how much more robust in monetary terms a system is (maybe via reduced breach probability) vs. the cost of added computation. Such analysis can help organizations decide how far to go in hardening. Additionally, usability aspects e.g., does more false positives from a defense overwhelm analysts to the point of diminishing returns? There's a human-m in the loop element to consider (especially for feature squeezing detectors generating alerts, etc.). Field studies deploying these measures in a SOC environment would be valuable to refine thresholds and processes to effectively integrate adversarial defenses without burnout or disregarded alarms (the classic trade off of false positives causing complacency).

Privacy Preserving and Adversarial ML: A future area mentioned is differential privacy and homomorphic encryption as defenses. We only touched on them conceptually. DP can mitigate some information leakage to attackers (so they can't easily do membership inference or model extraction to then craft adversarial examples)[19]. But DP also adds noise that could either make the model more robust (by smoothing decision boundaries) or less accurate. Investigating the dual benefit/trade off of DP in adversarial settings would be good, some studies (pre-2023) indicated DP training can indeed improve adversarial robustness slightly at the cost of accuracy, effectively acting as regularization. Homomorphic encryption mainly addresses confidentiality (keeping model or inputs secret, so an attacker can't directly inspect them) but it doesn't inherently improve robustness of the model's logic, it may stop an attacker from easily copying or probing the model but at huge

computational expense. Perhaps more promising are *secure enclaves or split learning* where part of the model runs in a secure environment and only certain information is exposed, limiting how an adversary can query or observe gradients. We didn't examine that it lies in system security design more than model design but is relevant in a comprehensive defense strategy.

Game Theoretic and Adaptive Strategies: We considered the arms race qualitatively. It would be useful to model this quantitatively. For example, use a **game theory framework** to model attacker and defender moves (attacker chooses how much perturbation to add, defender chooses threshold, etc.) and find equilibrium. This can help in choosing optimal thresholds (like how much false positive to accept for given adversary strength). Some initial works exist (like by Barreno et al., 2010, cited in NIST[65]), but there's room to incorporate modern attack/defense dynamics, especially with learning agents possibly adapting each round. Also exploring *learning algorithms with provable robustness* (e.g., convex hull methods, monotonic networks) in a cybersecurity context would be future work, maybe they sacrifice some accuracy but have guaranteed bounds on adversarial vulnerability. Are those more desirable in high assurance systems? Possibly.

Exploring larger scale combination of defenses: We combined a few and saw synergy (like adversarial training + detection). There are many combinations possible (e.g., ensemble of adversarially trained models with differential privacy, etc.). Future research might try to formulate a unified framework that provides multiple layers of defense and systematically tests it against adaptively strengthening attacks. Possibly *metalearning a defense strategy* could be tried: have a reinforcement learning agent adjust thresholds or model components in response to ongoing attack patterns (like an automated cyber defense agent that monitors the ML system's performance and tunes it in real time to maintain robustness as attacks evolve).

In summary, while our dual perspective analysis significantly enhances understanding of adversarial AI in cybersecurity, it also highlights that this is an evolving battle. Our work should be extended by: Testing on **more diverse and large scale datasets and scenarios** to generalize results. Evaluating against **more adaptive and complex attacks** (to ensure defenses hold). Considering the **operational context** (false positives, cost, integration with existing security workflows). Exploring **advanced and emerging defenses** (certified robustness, moving target ML, etc.). Adversarial AI will continue to be a critical concern as both AI usage in security and attacker sophistication grow; ongoing research and adaptive strategies will be needed to stay ahead in this attacker defender game.

6 Conclusion

This study adopted a dual perspective approach to examine the interplay between adversarial attacks and defenses in the context of cybersecurity focused AI systems. From the offensive viewpoint, we demonstrated that machine learning models used in security (such as intrusion detection systems) remain vulnerable to both training time and inference time attacks. **Poisoning attacks** can subtly corrupt an ML model's decision boundaries by injecting malicious training data or flipping labels, leading to degraded detection accuracy and even targeted blind spots (e.g., the model consistently misclassifying a certain malware as benign)[8][60]. **Evasion attacks** can generate imperceptible input perturbations, minor tweaks to network traffic features or malware attributes that cause even a well performing model to misclassify, all while the malicious nature of the input is unchanged[1]. These findings underscore that, in their vanilla form, machine learning based defenses can be **degraded or outright circumvented by adaptive adversaries**. The real world implication is stark: an investment in AI driven security is not a panacea, as attackers may exploit model weaknesses to slip malicious activities through (e.g., crafting network packets that an IDS overlooks, or modifying malware binaries to evade an AI antivirus).

However, from the defensive viewpoint, our research also offers hope and concrete strategies to bolster robustness. We found that a combination of countermeasures can **significantly improve the resilience** of ML

models against adversarial interference. *Adversarial training* proactively training the model on adversarial examples, notably increased model robustness, making it far harder for evasion attacks to succeed (robust accuracy improved by up to ~30 percentage points in our experiments)[9]. *Ensemble learning* and diversity in models forced attackers to face multiple lines of defense simultaneously, reducing the overall attack success; an attacker who could fool one model struggled to fool all, meaning the ensemble often still caught the attack[5]. Lightweight *feature squeezing detection* added a valuable layer of security by flagging inputs likely to be adversarial, so they could be scrutinized or handled separately[6]. Meanwhile, training time defenses like *data sanitization and robust statistics* mitigated the effect of poisoning, ensuring that the model's learning isn't unduly skewed by a few malicious points[7]. Together, these approaches can **significantly harden ML systems**. Our hardened configurations were able to maintain high detection rates even under potent adversarial attacks, whereas unprotected models failed.

In essence, our dual perspective analysis contributes an integrated understanding that goes beyond examining attacks or defenses in isolation. We highlight that robust machine learning in cybersecurity is achievable but requires careful design choices and possibly a multilayered defense strategy. Organizations looking to deploy AI for security should implement some of the recommended practices: adversarially train their models, utilize ensemble or redundant detection mechanisms, incorporate anomaly checks for adversarial behavior, and monitor training pipelines for integrity. By doing so, they can **substantially improve the robustness** of their systems, turning what might have been an "easy target" model into a resilient component of their security infrastructure.

Looking ahead, we encourage further research and industry collaboration to establish standardized benchmarks for adversarial robustness in cybersecurity (similar to how there are benchmarks for accuracy). The field would benefit from *scalable evaluation frameworks* that continuously test models against evolving adversarial techniques, and from the development of *certifiably robust* ML methods that can provide guarantees on performance under attack. Given the escalating adoption of AI in cyber defense, such efforts are crucial to ensure trust and reliability. Ultimately, our study underscores a key message: **security practitioners must assume that attackers will target the AI itself**, and thus must harden and continuously evaluate their ML models just as they do any other part of their security apparatus. With a combination of theoretical insights and practical defenses, many of which we have demonstrated, the community can stay ahead in the cat and mouse game of adversarial AI, leveraging the power of machine learning without falling prey to its pitfalls.

References

- Alshahrani, E., Alghazzawi, D., Alotaibi, R., & Rabie, O. (2022). *Adversarial attacks against supervised machine learning based network intrusion detection systems*. PLOS ONE, 17(10), e0275971[34][60].
- Barreno, M., Nelson, B., Sears, R., Joseph, A. D., & Tygar, J. D. (2010). *Security of Machine Learning*. Machine Learning, 81(2), 121-148[65].
- Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., ... & Roli, F. (2013). *Evasion attacks against machine learning at test time*. In **ECML PKDD** (pp. 387-402)[1].
- Goodfellow, I., McDaniel, P., & Papernot, N. (2018). *Making machine learning robust against adversarial inputs*. **Communications of the ACM, 61**(7), 56-66[1][12].
- Huang, L., Joseph, A. D., Nelson, B., Rubinstein, B. I., & Tygar, J. (2011). *Adversarial machine learning*. In **AISec** (pp. 43-58)[65].
- National Institute of Standards and Technology (NIST). (2019). *Draft NISTIR 8269: A taxonomy and terminology of Adversarial Machine Learning* (Tabassi, E., et al.)[30][1].

- Papernot, N., McDaniel, P., Sinha, A., & Wellman, M. (2016). *Towards the science of security and privacy in machine learning*. **IEEE European Symposium on Security and Privacy**, 2016[35][4].
- SentinelOne Labs. (2022). *What Are Adversarial Attacks? Threats & Defenses*[68][5].
- Tabassi, E., Burns, K. J., Hadjimichael, M., Molina-Markham, A., & Sexton, J. (2019). *Adversarial Machine Learning: A taxonomy and terminology of attacks and mitigations*. (Draft NISTIR 8269)[29][19].
- Xu, W., Evans, D., & Qi, Y. (2018). *Feature Squeezing: Detecting adversarial examples in deep neural networks*. **Network and Distributed System Security Symposium (NDSS)**[6][78].

