Formally Verifying Robustness and Generalisation of Network Intrusion Detection Models

Robert Flood
University of Edinburgh
Edinburgh, United Kingdom
r.flood@ed.ac.uk

Marco Casadio
Heriot-Watt University
Edinburgh, United Kingdom
mc248@hw.ac.uk

David Aspinall
University of Edinburgh
Edinburgh, United Kingdom
da@ed.ac.uk

Ekaterina Komendantskaya
University of Southampton
Southampton, United Kingdom
e.komendantskaya@soton.ac.uk

ABSTRACT

We introduce a new approach for robustness and generalisation of neural network models used in Network Intrusion Detection Systems (NIDS). Models for NIDS must be robust against both natural perturbations (accounting for typical network variations) and adversarial attacks (designed to conceal malicious traffic). The standard approach to robustness is a cycle of training to recognise existing attacks followed by generating new attack variations to defeat detection. Besides robustness, another problem with research NIDS models trained on limited datasets is the tendency to over-fit to the dataset chosen; this highlights the need for cross-dataset generalisation. We address both problems by incorporating recent formal verification tools for neural networks. These frameworks allow us to characterise the input space and we also use verification outputs to generate constrained counterexamples to generate new malicious and benign data. Then adversarial training improves both generalisation and adversarial robustness. We demonstrate these ideas with novel specifications for network traffic, training simple, verifiable networks. We show that cross-dataset and cross-attack generalisation of our models is good and can outperform more complex, state-of-the-art models, unable to be verified similarly.

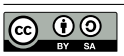
CCS CONCEPTS

• **Security and privacy** → **Logic and verification**; **Intrusion detection systems**; • **Computing methodologies** → *Machine learning approaches.*

KEYWORDS

IDS, formal verification, neural networks, robustness, generalisation, network security

ACM Reference Format:
Robert Flood, Marco Casadio, David Aspinall, and Ekaterina Komendantskaya. 2025. Formally Verifying Robustness and Generalisation of Network Intrusion Detection Models. In *The 40th ACM/SIGAPP Symposium on Applied Computing (SAC '25), March 31-April 4, 2025, Catania, Italy.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3672608.3707927



SAC '25, March 31-April 4, 2025, Catania, Italy

ACM ISBN 979-8-4007-0629-5/25/03
https://doi.org/10.1145/3672608.3707927

1 INTRODUCTION

Given the adversarial nature of intrusion detection, machine leaning-based network intrusion detection systems (NIDS) must be robust to evasive behaviours. Security is often likened to a cat-and-mouse game and this is seen in neural network robustness: adversarial attacks are proposed, defences are designed, before new attacks are created, restarting the cycle. In contrast to other defences, *neural network verification* is a formal approach which guarantees robustness within specified parameters. Recent work on methods like Reluplex [28, 42] use logical specifications to impose constraints on neural networks, restricting their outputs for given inputs and providing mathematical robustness guarantees.

But adversarial robustness is not the only form of robustness needed: NIDS should also be robust to inherent problem-space diversity, such as changes to network bandwidth or to minor variations in malicious traffic. In research NIDS pipelines, these differences are often not evaluated. One reason is the scarcity of sufficiently diverse public NIDS data; models are often trained and tested on the same dataset, without testing for robustness to concept drift. Furthermore, without sufficient data, it is difficult to validate that models do not overfit to arbitrary features (unrelated to attack classes being recognised). The marked reduction in the performance of typical models when trained and tested using disparate datasets is well-known, even for datasets containing similar malicious traffic [2]. In other words, *cross-dataset generalisation* is often poor. This is likely also true in real-world scenarios when IDS developers have limited access to sufficiently varied data.

Our work addresses these twin issues of verifiable robustness and generalisation by ensuring that the models we train verifiably adhere to certain *global constraints*. In contrast to other fields, such as computer vision where specifications are defined *locally* via ℓ_p-balls, discussed in Section 2.3, network data is structured enough that we can specify expected properties **globally**. For example, we can specify the structure of well-formed TCP handshakes, while our models learn other relationships independently. An analogy to image classification might help: we might verify that an image model classifying animals biases black and white pixels when identifying pandas, while allowing the network to infer the general shape of the animal. Importantly, our approach is also distinct from standard signature-based NIDS, as our specification provide only high-level details about expected malicious behaviour.

Our global constraints define known regions of benign and malicious traffic, allowing us to enforce model behaviour for corresponding network flows. We do this by generating additional malicious and benign data via *specification-driven adversarial training* [19], finding counter-examples via PGD [31] that lie within our benign/malicious constraints and adding them to the training set. This process expands the training data and also helps the model satisfy our specifications. By strengthening expected model behaviour in this manner, the models prioritise features reflecting attack behaviour over less relevant features, improving generalisation. We show a high-level overview of this process in Figure 1, with more detail in Section 3.

We define our properties using *Vehicle* [14], a specification language for writing and testing logical constraints for neural networks. Vehicle offers a high-level, readable DSL which acts as a front-end for the Reluplex-based verifier Marabou [42]. Note that both Vehicle and Marabou are pre-existing, emerging tools and not contributions of our paper. However, our work demonstrates their non-trivial application to a new domain area. Using Vehicle, we can better reason about model behaviour by contrasting satisfiable and unsatisfiable specifications. For instance, we automatically produce evasive flows according to threat models written in Vehicle and rank their relative effectiveness.

To summarise, our contributions include:

- **Specifications**: We write and test many specifications for model verification. In doing so, we eschew ℓ_p metrics. Instead, we embed expert knowledge in a *global* manner, specifying non-trivial, novel and performant properties. We provide the code outlining this process for public use[1].
- **Generalisation**: We explicitly measure the generalisation performance of our networks, considering both *cross-dataset* and *cross-attack* generalisation. Alongside standard benchmark datasets, we generate bespoke attack data.
- **Verification**: We produce NIDS models that are verifiably robust against adversarial and natural perturbations in certain regions of feature space. We also use counterexamples to examine model weaknesses, ranking strategies for producing evasive traffic. Via this process, we show that delaying packets is the most effective evasive strategy for our models.
- **Performance**: Our verified models outperform a comparable state-of-the-art model, improving generalisation by 40%.

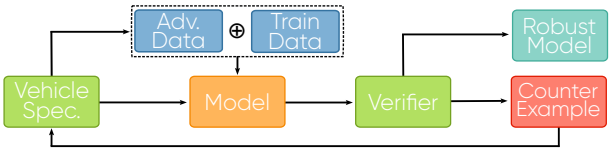


Figure 1: The specification-driven training pipeline.

Structure of the paper. Section 2 covers some basic background on NIDs, adversarial robustness and recent work on neural network verification. Section 3 describes the methodology for our new verified NIDS pipeline, in particular, explaining the format and meaning of specifications. Then Section 4 describes eight example applications of verification in NIDS, including cross-dataset generalisation (robustness), cross-attack generalisation and using the verification process to generate realisable evasive traffic. This last application can help understand model failures using the specification language, suggesting fixes or pre-processing. Finally, Sections 5, 6 and 7 discuss limitations, related work and summarise the paper.

[1] https://github.com/glo-fi/NIDS-Verify

2 BACKGROUND

2.1 Network Intrusion Detection

Benchmark NIDS datasets often have two parts, raw PCAP data, and tabular flow features. These public datasets have bolstered the development of myriad architectures and pipelines.

ML-based NIDS ability to generalise to new data scenarios is highlighted as a key advantage over signature-based methods, a driving motivation for the field. Whilst some works explicitly test for generalisation [1], the rarity of cross-dataset evaluation, or explicit generation of an alternative test data, fails to interrogate this assumption. Furthermore, when such results are reported, they suggest that cross-dataset generalisation is a major hurdle in NIDS research: in an investigation of NIDS cross-generalisation, Apruzzese et al. [2] find that NIDS F1 scores can collapse from near-perfection to less than 0.5. This is despite the fact that some benchmark datasets are highly similar, such as CIC IDS 2017 and 2018 [34], containing conceptually similar benign traffic and attacks.

2.2 Neural Network Verification

Recent advancements have seen the development of multiple neural network verification frameworks, which help validate the functionality and safety of neural networks under specific conditions [3, 4]. These tools are either complete verifiers, delivering definite true or false decisions, or incomplete verifiers that provide true or unknown outcomes. Complete verifiers employ methods like Satisfiability Modulo Theories (SMT), Mixed-Integer Linear Programming (MILP), or Branch-and-Bound (BaB).

SMT-based verifiers [28, 42] and MILP-based strategies [12, 36] translate specifications into linear inequalities, offering precise constraint representation at the expense of scalability. Conversely, BaB verifiers [18, 21] approximate constraints to improve scalability at the expense of precision. In this work, we use the SMT-based Marabou [42] for complete verification.

Marabou processes network properties by modifying the simplex method for networks with piece-wise linear functions like ReLUs, known as ReLUplex [28]. It computes node bounds and checks the satisfaction of linear constraints, identifying counter-examples when conditions are not met. We interface with Marabou using Vehicle, a high-level functional language designed for precise specification writing in neural networks [14].

To satisfy non-trivial constraints, networks are trained for *robustness*, using techniques such as adversarial training. After robust training, models often achieve higher verification success and are more likely to satisfy the desired properties.

2.3 Adversarial Robustness in NIDS

Given their security-centric role, adversarially robust NIDS are vital. Despite this, there is a disconnect between existing NIDS evasion

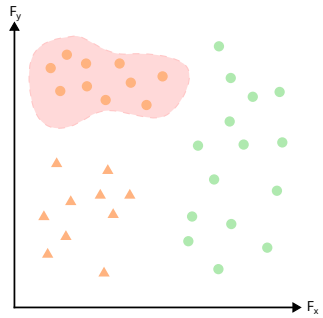


Figure 2: Initial boundary - Fails to generalise

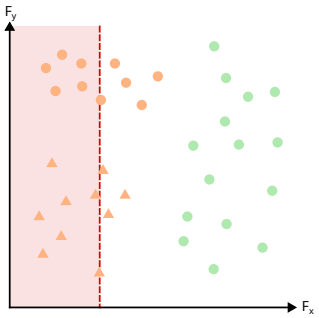


Figure 3: Global Spec. - Constrains all entries together

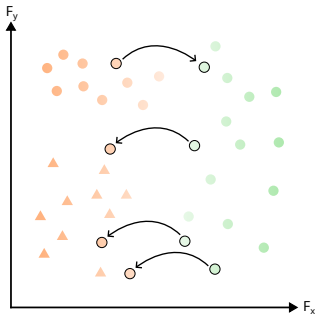


Figure 4: Local Spec. - Constrains entries individually

●: Malicious Data - Training, ▲: Malicious Data - Test, ●: Benign Data, F_x: A *relevant* feature, F_y: An arbitrary feature

methodologies and those often used to determine model robustness. In the latter case, 'closeness' can be defined as a region of perturbed inputs alongside some norm [29]. *Classical robustness*, intuitively, states that small variations to model input should result in small changes in its output. Mathematically, given model N with input x, is said to be robust iff for all ϵ there exists δ such that:

$$\forall x, x_0.\ ||x - x_0||_p < \delta \Rightarrow ||N(x) - N(x_0)||_p < \epsilon \tag{1}$$

where $||.||_p$ is the standard ℓ_p norm.

However, in NIDS, attacks that are identical in purpose can have wildly varying flow statistics and existing IDS evasion techniques do not map neatly onto the above notion of robustness. Instead, several papers craft adversarial examples with domain constraints, aiming to maintain flow semantics[24, 35, 39].

3 METHODOLOGY

Rather than simply apply tools to a new setting in the most straightforward way, we want to explore how verification can spawn new ideas in NIDS research. Vehicle [14] is ideal for this, allowing us to pinpoint and interpret details of model behaviour.

To develop this methodology, we have to confront limitations of current verification frameworks whilst writing non-trivial specifications. First, the Vehicle language is limited to simple quantifiers, boolean operands, conditionals and arithmetic. Thus, our feature set needs to express concrete properties of the traffic, avoiding features with high inter-feature dependency such as 'Forward Packet Size Mean' and 'Total Packet Size Deviation'. Second, for our properties to correctly bound benign/malicious traffic, we rely on global constraints that reflect domain knowledge. Third, we have to write constraints that do not cause exponential blow-up of Marabou. We note that rules for signature-based NIDS have similar complexity and expertise requirements, but would not allow for the generalisation that our NIDS builds in. We detail our approach to these issues in Sections 3.3, 3.1 and 3.2 respectively.

Our verification examples in Section 4 have multiple specifications with many unique constants. For readability, we replace these constants, writing lower bounds for a feature F as α_F, upper bounds as β_F and constants as γ_F. Thus, a simple, arbitrary specification for model N and input x that constrains N to output class C_0 when the first five features lie within certain bounds takes the form:

$$\texttt{Bound5Feats} : \forall i \in [1,5].\ \alpha_i \leq x_i \leq \beta_i \implies N(x) = C_0 \tag{2}$$

We can also use specifications to check for correct usage. For instance, all models we develop adhere to `ValidInput`, which ensures that all features lie between 0 and 1:

$$\texttt{ValidInput} : \forall i, x.\ 0 \leq x_i \leq 1 \tag{3}$$

As large feature sets can cause overfitting to arbitrary features [25], we use a restricted feature set. Given a flow of size N, we extract packet-level features from the first m packets: packet sizes, inter-arrival times (IATs), TCP flags and packet directions. We supplement these with two features that are constant across a flow: transport protocol — TCP or UDP — and time since last flow with identical Source/Destination IPs — a feature we call *TimeElapsed*. Otherwise, when $N \leq m$, we zero pad for 'missing' packets. For a given flow x, we write the packet features of the i-th packet as: $x_{FeatureName\text{-}i}$.

In Sections 3.1– 3.2, we introduce our methodology with a straightforward experiment to examine whether, given a simple attack and weak generalisation conditions, it is possible satisfy a global robustness specification. We train a model to detect nmap traffic [30] and verify its correctness properties.

3.1 Global vs. Local Specifications

Our aim is to write specifications that improve model robustness and cross-dataset generalisation, by encoding expert knowledge. We do this *globally*. In other words, each property is defined in a fixed manner for the entire input space. We contrast this with *local* specifications, where each property is dependant on specific inputs, such as *classical robustness* defined in Section 2.3. Both formulations aim to enforce model robustness. We demonstrate the difference visually in Figures 3–4.

We divide features into three categories depending on how an expert may interpret them:

(1) ***Related** features* which are directly relevant for classification.
(2) ***Bounded** features* whose behaviour is fixed in certain intervals.
(3) ***Unknown** features* whose relationship is unknown.

Consider detecting a DoS attacks flooding a HTTP server. A defining feature is that the webpage is repeatedly accessed in quick succession. Thus, *TimeElapsed* is likely a *related* feature. In contrast, small changes to the size of the HTTP GET request packet — which can naturally vary — is likely irrelevant for flow classification. As such, we would consider this feature to be *bounded*. Importantly, we do **not** want to exclude this feature entirely, as large changes could still be indicative of abnormal behaviour, such as a large payload encoded as a URL parameter. Instead, we aim to verify model behaviour in a bounded, specified interval. Finally, the relationship between, for example, packet flag features and classification may be unclear. We would consider these to be *unknown* features. For other attacks, this categorisation would almost certainly change.

We apply a similar logic to our nmap example. As standard nmap flows are short and highly predictable, consisting of quick three-way handshakes with fixed sizes, we write specifications that describe these properties as being *related* or *bounded*, detailed below.

Related Features. For global specifications, we treat *related* and *bounded* features in a similar manner, specifying model behaviour explicitly within certain regions. For *related* feature $\hat{F}$, we specify an interval where the model must make fixed classification decisions. Formally, given flow x which takes on value $x_{\hat{F}}$ for feature $\hat{F}$ and model N that performs binary classification $N(x) \rightarrow C_i, i \in [0, 1]$, we fix i and specify an interval $[\alpha, \beta]$ such that:

$$\forall x. \, x_{\hat{F}} \in [\alpha, \beta] \implies N(x) = C_i \tag{4}$$

In words, all **arbitrary** x with $x_{\hat{F}} \in [\alpha, \beta]$ must be classified as C_i.

For our nmap verification, we highlight related features by specifying that flows with sufficiently 'quick' *TimeElapsed* values and packet sizes that are 'close' to those of a standard nmap connection must be malicious (writing $N(x) = C_{mal}$ as *mal*):

$$\texttt{MalElapsed} : x_{timeElapsed} = 0.0 \vee x_{timeElapsed} \leq \beta_{timeElapsed} \tag{5}$$

$$\texttt{MalPktSize} : \forall i \in [1, 3]. \; \alpha_{pktSz-i} \leq x_{pktSz\text{-}i} \leq \beta_{pktSz-i} \tag{6}$$

$$\texttt{nmap} : \forall x. \; \texttt{MalElapsed}(x) \wedge \texttt{MalPktSize}(x) \implies mal \tag{7}$$

Bounded Features. In contrast, for *bounded* feature $\hat{F}$, we wish to specify an interval where altering $x_{\hat{F}} \in [\alpha, \beta]$ does not change model behaviour for **fixed** x. Formally, given two flows x and $\bar{x}$ which differ only on feature $\hat{F} \in [\alpha, \beta]$:

$$N(x) = C_i \implies N(\bar{x}) = C_i \tag{8}$$

In other words, altering $x_{\hat{F}}$ within $[\alpha, \beta]$ should not affect the model's output.

Unknown Features. We stress that we do not attempt to completely encapsulate attack definitions — or that this is even possible to do by hand. Thus, we include features where we do not specify their relevancy for classification, leveraging the model's ability to learn the relationship for us.

Importantly, a feature's category can be defined **implicitly** across multiple rules. Given some property P_1 which bounds feature $F_1 \in [\alpha_1, \beta_1]$ and property P_2 which bounds $F_2 \in [\alpha_2, \beta_2]$, the following rules imply F_1 is *bounded* and F_2 is *related*:

$$S_1 : P_1 \wedge P_2 \implies N(x) = C_0$$
$$S_2 : P_1 \wedge \neg P_2 \implies N(x) = C_1 \tag{9}$$

3.2 Principles for Global Specifications

When writing specifications, we follow several principles to ensure their utility whilst adhering to the technical limitations of the verification framework. We present more specifications in Section 4.

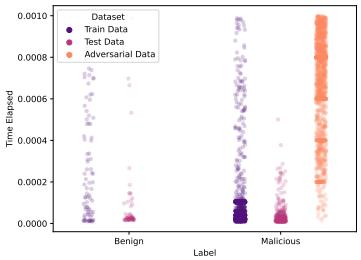


Figure 5: Expansion of related feature, *TimeElapsed*.

Input Expansion. By generating additional data, our specifications expand the space of model inputs. This is good: NIDS datasets often have limited diversity [20] and expanding input could improve generalisation. The adversarial training procedure 'fills' loose bounds around certain features, as seen in Figure 5, which shows that adversarial data is more dense in regions without training data. Thus, we *underspecify* the decision boundary as wider bounds may lead to contradictions between properties or cause the model to incorrectly learn benign/malicious behaviours.

Input Reduction. Although we want an exhaustive specification that captures all malicious flows in the training dataset (and all realisable malicious flows not in the data), in reality, this is impossible. One limiting factor is that disjunctions cause exponential blow-up of the Marabou backend. For instance, we can write a property that must hold regardless of packet directions:

$$\texttt{allDirections} : \forall i \in [1, 10]. \; x_{pktDir-i} = \text{in} \vee x_{pktDir-i} = \text{out} \implies y \tag{10}$$

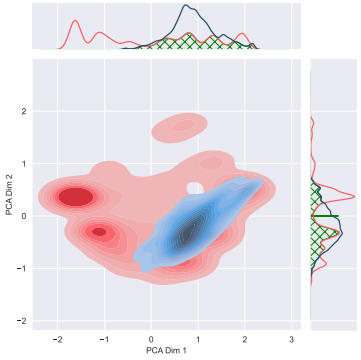


Figure 6: Projection showing the limited overlap between training and adversarial data.

However, this requires 2^{10} checks to verify. Poorly written specifications can easily require costly checks. Thus, as well as generating additional input, we must also ***restrict*** the verifiable input space to prevent blow-up. For instance, we can consider only flows that conform to expected protocol behaviour (determined via packet sizes and flags). Similarly, we can limit properties by only considering the most common combinations in the training data. Together, these allow us to write specifications representing realistic traffic without onerous resource usage. For instance, in Section 4.1, we consider four combinations of packet directions, encapsulating approximately 60% of malicious training data whilst reducing the number of checks from 1024 to 16. This limited subspace can be visualised as the space occupied by the adversarially generated data versus the training data. In Figure 6, we demonstrate this reduced overlap via PCA along the bounded features.

Counterexample-Guided Specifications. In our experience, writing immediately satisfiable specifications often stems from inadvertently trivial conditions. Due to the feature/problem space gap in NIDS, it is difficult for specifications to cover all edge cases, which then fail in some unexpected way. For instance, when specifying the behaviour of benign traffic, one might not bound a flow's duration. However, this will likely be unsatisfiable as, say, the model may classify negative durations as malicious. However, this flow is not actually realisable, suggesting a disconnect between the specification and what the model understands to be benign.

Thus, we write specifications in a counterexample-guided manner. Given a counterexample x, we check whether x conforms to our understanding of the malicious data. Keeping a human-in-the-loop, if we identify x as unrealisable, due to, say, impossible packet sizes, inter-arrival times or flag combinations, we rewrite the specification to eliminate this mistaken counterexample. The specifications in Section 4 vary in complexity and sensitivity, with some requiring 40+ counterexamples and refinements to be verifiable.

Nmap Results. As can be seen in Table 1, we achieve a perfect F1 score classifying basic nmap traffic. Due to the attack's homogeneity, this is not surprising. However, the model is mathematically guaranteed to be robust against minor perturbations. Importantly, the specification is only satisfied *after* adversarial training. Before, no properties were verifiable, even after reducing the global bounds. This highlights how ML-based NIDS can easily overfit, justifying using verification to guarantee robustness.

3.3 Feature Set & Data

We test the model's cross-dataset generalisation performance with an even split between the benign and malicious classes. We consider *DoS Hulk*, *DoS Slowloris*, *FTP/SSH Bruteforce* and *SQL injection* traffic as the malicious classes. We sample training data from CIC IDS 2017 and consider two test datasets. For the first, we extract benign and attack data from CIC IDS 2018. For the second, we use bespoke attack data to evaluate model performance when generalising to different network conditions and flow lengths. We generate this traffic via DetGen [13], a deterministic network generation framework. When applicable, we alter the traffic's *temporal* characteristics by limiting attacker bandwidth to 10mb, 25mb and 50mb, altering both inter- and intra-arrival times i.e., *TimeElapsed*. To alter the traffic's *spatial* characteristics, we assume that an attacker is able to attack different webpages/databases or use evasive packet-padding. We generate attack traffic for five scenarios, corresponding to different webpage/database sizes or padding lengths and combine this with benign traffic from CIC IDS 2018. We call this the '*DetGen*' dataset. We preprocess data and select models via reference to the training data only to prevent data leakage and to simulate detecting unknown attacks. This process follows the recommendations of Flood et al. [20] to alleviate issues with benchmark NIDS datasets.

4 APPLICATIONS OF VERIFICATION

All models we train are of fixed architecture, consisting of a feedforward network with shape $(256, 128, 2)$ with approximately 44000 parameters. For these networks, Vehicle takes roughly 8 minutes to verify all global specifications.

4.1 Cross-dataset Generalisation

Experiment. Following the nmap verification, we consider more complicated attacks with more complex specifications. First, we train a model to detect all volumetric denial of service attacks in the CIC datasets, aiming to generalise to arbitrary network conditions and page sizes. To ensure that the model learns the salient features of each attack, we write properties that reinforce these generalisation aims. The DetGen data consists of HTTP flood traffic generated in the manner outlined in Section 3.3. As a benchmark, we compare our results to LUCID [16], a state-of-the-art DoS classifier, as well as our simple model without adversarial training.

Specifications. All DoS attacks in the CIC datasets function via asymmetric resource usage. Some attacks increase the target load by delaying responses, leading to artificially high inter-arrival times and flow durations, or deviate from the expected behaviour of a TCP/HTTP connection. We aim to capture these defining qualities in our properties. In addition to `ValidInput` and `MalTimeElapsed` from Section 3, we write three base properties:

$$\begin{aligned}&\texttt{ValidTCPHandShake} : x_{pktFlag\text{-}1} = \text{SYN} \wedge x_{pktSz\text{-}1} = 52 \\ &\wedge x_{pktDir\text{-}1} = \text{out} \ \wedge x_{pktFlag\text{-}2} = \text{SYN + ACK} \wedge x_{pktSz\text{-}2} = 52 \\ &\wedge x_{pktDir\text{-}2} = \text{in} \ \wedge x_{pktFlag\text{-}3} = \text{ACK} \wedge x_{pktSz\text{-}3} = 40 \\ &\wedge x_{pktDir\text{-}3} = \text{out} \ \wedge x_{protocol} = \text{TCP} \end{aligned} \tag{11}$$

$$\begin{aligned}&\texttt{ValidHTTPConn} : x_{pktFlag\text{-}4} = \text{ACK + PSH} \wedge \alpha_{pktSz-4} \leq \\ &x_{pktSz\text{-}4} \leq \beta_{pktSz-4} \wedge x_{pktDir\text{-}4} = \text{out} \ \wedge x_{pktFlag\text{-}5} = \text{ACK} \\ &\wedge x_{pktSz\text{-}5} = 40 \wedge x_{pktDir\text{-}5} = \text{in} \ \wedge x_{protocol} = \text{TCP} \end{aligned} \tag{12}$$

$$\begin{aligned}&\texttt{ValidIATs} : \forall i \in [2, 10].\ 0.000001 \leq x_{pktIATs-i} \leq 0.05 \vee \\ &\sum_i x_{pktIATs-i} \leq 0.2 \end{aligned} \tag{13}$$

$$\texttt{validSizes} : \forall i \ (x_{pktSz-i} >= 40) \wedge (\sum_5^{10} x_{pktSz-i} > 400) \tag{14}$$

These form the building blocks of the four main properties which specify malformed TCP connections, infrequent (presumably benign) HTTP traffic, volumetric HTTP traffic and volumetric HTTP traffic with unusual IATs: respectively, `invalidHTTP`, `GoodHTTP`, `HulkAttacks` and `SlowIATsAttacks`.

$$\texttt{invalidHTTP} : \forall x.\ \texttt{validInput}(x) \wedge (\neg\texttt{validTCPHandshake}(x) \vee \neg\texttt{validHTTPConn}(x)) \implies \mathit{mal} \quad (15)$$

$$\texttt{GoodHTTP} : \forall x.\ \texttt{validInput}(x) \wedge \texttt{validTCPHandshake}(x) \wedge \texttt{validHTTPConn}(x) \wedge \texttt{validTimeElapsed}(x) \wedge \texttt{validIATs}(x) \wedge \texttt{validSizes}(x) \implies \mathit{ben} \quad (16)$$

$$\texttt{HulkAttacks} : \forall x.\ \texttt{validInput}(x) \wedge \texttt{validTCPHandshake}(x) \wedge \texttt{validHTTPConn}(x) \wedge \neg\texttt{validTimeElapsed}(x) \wedge \texttt{validIATs}(x) \wedge \texttt{validSizes}(x) \implies \mathit{ben} \quad (17)$$

$$\texttt{SlowIATsAttacks} : \forall x.\ \texttt{ValidInput}(x) \wedge \texttt{ValidTCPHandShake} \wedge \neg\texttt{ValidIATs}(x) \implies \mathit{mal} \quad (18)$$

Table 1: F1 scores of tested networks. Bold indicates models that satisfy relevant specifications.

Classifier	Test Data	Acc.
Nmap Verification		
NN	**nmap (DetGen)**	**1.0000**
Cross-dataset Generalisation		
NN	DoS (CIC IDS 2018)	0.5583
NN	DoS (DetGen)	0.5622
LUCID	DoS (CIC IDS 2018)	0.5421
LUCID	DoS (DetGen)	0.5468
NN	**DoS (CIC IDS 2018)**	**0.9111**
NN	**DoS (DetGen)**	**0.9521**
Cross-attack Generalisation		
NN	SSH (CIC IDS 2018)	0.4456
NN	FTP (DetGen)	0.4914
NN	**SSH (CIC IDS 2018)**	**0.8871**
NN	**SSH (DetGen)**	**0.8059**
NN	**FTP (DetGen)**	**0.8938**

Results. Table 1 summarises performance of the models alongside comparative benchmarks. Since the specifications are global rather than local, we do not express verifiability as a percentage of verified subspaces. Instead, the success of the global specifications is presented as a binary outcome: either verified or not verified.

We verify a series of complex specifications whilst enhancing cross-dataset generalisation. Training with adversarial loss, we improve cross-dataset generalisation accuracy by approximately 0.35–0.4 on both the CIC IDS 2018 and DetGen datasets. We also note that more complicated models do not produce similar improvements. Contrasted with LUCID [16], a state-of-the-art DoS classifier which cannot be verified with Marabou due to its architecture, the model is highly performant whilst being verifiable.

Importantly, our specifications allow us to investigate complex feature interactions in a human-readable manner, ensuring that the model conforms to expectations. For instance, the `HulkAttacks` and `SlowIATsAttacks` specifications both require the *TimeElapsed* feature to be less than some bound, β_{Hulk} and $\beta_{SlowIAT}$, respectively, whilst the `SlowIATsAttacks` specification additionally constrains the IATs of a flow to be large. Maximising these β bounds, we find that this extra IAT constraint allows us to verify `SlowIATsAttacks` specifications when $\beta_{Hulk} << \beta_{SlowIAT}$, demonstrating that unusually high IATs contribute towards classifying flows as malicious.

4.2 Cross-attack Generalisation

Experiment. We test the impact of our verification pipeline on the model's *cross-**attack** generalisation* performance. Specifically, we evaluate the model's ability to generalise to distinct but conceptually similar attacks, not present in the training data. Due to the volumetric nature of both the *SSH-BruteForce* and *FTP-BruteForce* attacks in CIC IDS 2017 and 2018[2], we consider these attacks to have some high-level similarity to DoS data.

We train the model **solely** on DoS flows from CIC IDS 2017 and on specification-based adversarial DoS flows, which we generate without reference to the malicious SSH or FTP traffic. Specifically, we do **not** generate adversarial traffic based on the FTP or SSH specifications. However, we still try to verify these specifications for the model.

Specifications. Similar to the `ValidHTTPConn` property in Section 4.1, we restrict the specifications to SSH/FTP login handshakes, identified via sequences of packet sizes, and enforce that malicious FTP/SSH flows have low inter-flow arrival time:

$$\texttt{ValidLoginFTP/SSH} : \forall i \in [1, 10].\ x_{pktSz\text{-}i} = \gamma_{pktSz-i} \quad (19)$$

$$\texttt{NegTimeElapsed} : x_{timeElapsed} \leq \beta_{timeElapsed} \quad (20)$$

$$\texttt{MalLoginFTP/SSH} : \forall x.\ \texttt{ValidLoginFTP/SSH}(x) \wedge \texttt{NegTimeElapsed}(x) \implies \mathit{mal} \quad (21)$$

Results. Our approach also improves cross-attack generalisation. The model achieves a score of 0.8871 and 0.8938 on SSH and FTP BruteForce traffic respectively, when trained exclusively on DoS traffic. As we generate traffic that adheres to our specification during the adversarial training process, we expand the training data with synthetic malicious data with low *TimeElapsed* and synthetic benign data with high *TimeElapsed*. Subsequently, the model is far less likely to overfit to the DoS traffic in CIC IDS 2017, which has little diversity. As FTP and SSH BruteForce traffic can also be identified via low inter-flow arrival times, the models successfully generalise to these distinct attack classes. Furthermore, we manage to verify the specifications outlined in above **without** additional adversarial training, providing strong guarantees about detecting malicious SSH and FTP traffic.

4.3 Generating Realisable Evasive Traffic via Counterexamples

Experiment. Next, we aim to explore model failures constructively, using them to generate interpretable examples of verifiably evasive and realisable traffic flows for particular attacks automatically. We do this by training an initial NIDS to detect SQL injection traffic via a standard train/test pipeline — without adversarial training. We then write specifications based on random malicious samples in the training data, ensuring that they are verifiable by

[2]Whilst data has been labelled as FTP-BruteForce in CIC IDS 2018, we note that the attack was launched against a closed port so is uninteresting. We omit this part of the data and rely on the *DetGen* data instead.

tightly restricting feature bounds. By repeatedly querying the verifier whilst enlarging features bounds, we can produce counter-examples akin to a white-box, feature-space adversarial attack [23]. We target a model that was trained to detect SQL injection-based dumping of a mySQL database.

If performed naively, we have few guarantees about these flows, such as whether they correspond to actual traffic. To ameliorate this, we undertake this process in a systematic manner, aiming to produce counter-examples that reflect realisable evasive traffic. Our threat model assumes that an attacker can only modify their attack traffic, thus, we only change forward packet features. Furthermore, we assume that features can only be increased — via packet delays, random padding or turning inconsequential TCP flags on. With these restrictions, we can write hundreds of exploratory specifications to test whether any suitable evasive traffic conforms to these standards.

Specifications. Given an exemplar malicious flow, we write a specification that corresponds to the model classifying that particular flow as malicious, setting all features to constant values:

$$\texttt{Initial} : \forall i \in [1, m].\ x_{feature\text{-}i} = \gamma_{feature-i} \implies mal \tag{22}$$

As SQL injection traffic is more complicated than DoS traffic, we parse the first 26 packets of each flow. Approximately 12 of these are attacker controlled and, since we modify packet sizes, flags and IATs, this results in roughly 36 attacker controlled features. Due to the infeasibility of performing an exhaustive grid search over all of these features, we randomly select a subset a, re-writing the specifications such that $x_{feature-i} \in a$ is bounded from below by $\gamma_{feature-i}$ and above by $\beta_{feature-i}$.

$$\begin{aligned}\texttt{Evasive} : \forall x_{feature-i} \in a.\ \gamma_{feature-i} <= x_{feature\text{-}i} \\ <= \beta_{feature-i} \implies mal\end{aligned} \tag{23}$$

We consider any counterexamples produced via this process to be realisable evasive traffic. We automate this process 50 times, randomly selecting a for each iteration, tracking the ratio of satisifiable to unsatisfiable specifications. We consider values of $|a|$ up to 16, and of upper-bounds $\beta_{feature-i}$ in increments of 0.1. To gain better insight into the effectiveness of, say, flipping TCP flags compared with padding packet sizes as evasion strategies for this specific network, we then repeat the search, increasing the bounds of only a specific class of feature at a time.

Results. The entire verification process takes roughly 12 hours per flow on consumer hardware[3]. The model initially separated SQL injection traffic from benign flows with an F1 score of 0.81 and our analysis only holds for this specific model. However, we note that this verification-driven approach is generally applicable and provides unique insight into failure modes of a model. We successfully begin producing realisable evasive traffic for perturbations greater than 0.2, as seen in Figure 7.

In Figure 8, we show the effectiveness of our three attack strategies — padding packets, delaying packets and flipping TCP flags — at different perturbation levels. Here, we see that the success of

[3]A laptop with an 11th Gen Intel i5-1135G7 and 8GB of RAM

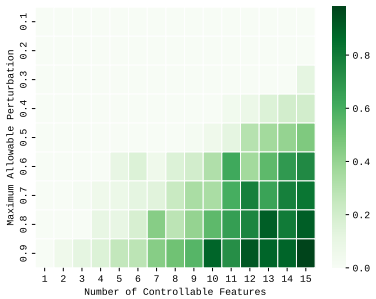


Figure 7: Heatmap showing the percentage of evasive flows for a given bound, when n features are attacker-controlled.

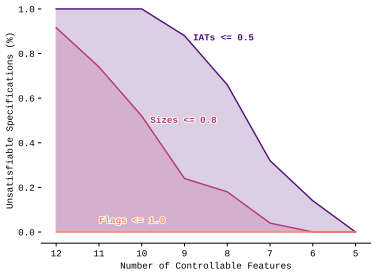


Figure 8: Effectiveness of the three feature-specific strategies at various levels of maximum allowed perturbation.

verification may vary greatly which is why the described heuristics are important. Delaying packets is the most effective strategy by far; both delaying several packets by small amounts, or a few packets by large amounts result in realisable counter-examples. In contrast, when padding packets, the specifications require far larger bounds in order to produce counter-examples whilst exclusively flipping TCP flags fails to produce any evasive traffic whatsoever.

Although targetting specific classes of features is more effective than random features, we note that there are cases where, depending on attacker constraints, the initial general search is worthwhile. For instance, our process does not find a single evasive flow when only three IAT features are bounded from above by 0.5. However, our more general process does find such counterexamples, provided other features are also allowed to be perturbed.

Importantly, we see that the effectiveness of the evasion strategies varies widely depending on the initial seed flow, allowing us to infer model failure modes with high granularity. For certain flows, delaying packets is the *only* strategy that produces counter-examples whereas, for others, alternative strategies also work.

4.4 Effectiveness of Local Robustness

Experiment. Adversarial training is a state-of-the-art technique, typically used to strengthen local robustness and is a direct alternative to enforcing global constraints. However, there is little reason for local robustness to attenuate concept drift as our approach does. Instead, the cross-dataset generalisation performance is a consequence of enforcing **global** robustness constraints that accurately reflect the underlying structure of malicious traffic, rather than

robustness in general. To demonstrate the difference, we train the NIDS to be locally robust about each input point. We use PGD with $\epsilon = 0.1$ to train the model adversarially and verify that the model is indeed locally robust using Vehicle before evaluating its cross-dataset generalisation.

Results. We successfully train a model that is verifiably locally robust about each input point. However, this provides absolutely no benefit to cross-dataset generalisation, with the locally robust model achieving an F1 score of only **0.5792** on the test dataset. As local perturbations do not adhere to the underlying structure of network traffic, the local adversarial examples contain little information about out-of-distribution inputs, unlike our global bounds.

4.5 Comparison with BARS

Experiment. As far as we are aware, we are the first to apply deterministic neural network verification to NIDS. As a comparison, we contrast this verification approach to BARS [40], a probabilistic certified robustness methodology for NIDS based on randomised smoothing. For a given model N and input x, BARS produces targeted noise generator a G_N and robustness region r^x about x such that smoothed model $\hat{N}$ is certifiably robust when sufficient points are sampled from within r^x.

Although BARS is situated in a similar domain, its goals and approach differs from our work significantly as it focuses on local adversarial perturbations. Following from Section 4.4, we note that both global specifications and domain constraints are difficult to represent via $\epsilon - \delta$ balls. Rather than supercede BARS, we intend to show how verification differs from certified robustness.

We apply BARS to a DoS NIDS, with train and test data generated according to our cross-dataset generalisation procedure in Section 4.1, producing certified radii for the malicious class.

Results. For the globally verifiable model, we see that BARS does not correctly calculate robustness regions. As a result, the BARS robustness curve is constant for all values of ϵ. Investigating the noise produced by G_N, we see that features are overwhelmingly perturbed to be less than 0 and fail to change model output when clipped between 0 and 1. As a result, each r_i^x is significantly greater than 1, providing little insight into the actual robustness region of x. Whilst BARS works as intended in more general settings, in this instance, the disconnect between BARS robustness regions and the realisable values of network traffic — which are easily expressed in Vehicle specifications — impedes its performance.

4.6 Specification Transferability

Experiment. As our counter-example driven approach requires verifying models repeatedly, we use a small model to make this feasible: each counter-example takes approximately 30 seconds. However, verifying larger models is useful. Thus, after defining the robustness regions and specifications using the small base model, we train seven additional networks of increasing size, up to approximately 2.8 million parameters. This is a similar scale to models in verification competitions [6]. As increasing model depth leads to exponentially greater verification time, we limit models to four hidden layers. We train these models according to the original robustness regions before attempting to verify the `GoodHTTP` specification, determining whether specifications and robustness regions are transferable between models of different sizes.

Table 2: Neural Network Parameters Versus Verification Time

Parameters	Time (s)	Verifiable?
43776	41	✓
186498	47	✓
252290	53	✓
449154	239	✓
733314	910	✓
963330	1023	✓
1750274	5661	✓
2842882	19708	✓

Results. We find that the `GoodHTTP` specification transfers perfectly across all models tested. For each model architecture, we achieve perfect adversarial accuracy during training, resulting in models that adhere to `GoodHTTP` without any additional modification of robustness regions or specifications. Whilst we only have to perform this final process once, verifying larger models is more onerous; Table 2 demonstrates the tradeoff between model size and verification time.

4.7 Robustness of Global Constraints

Experiment. When we train models adversarially, we generate adversarial samples based on bounds defined in the specifications, constraining features to various degrees. It is possible that the bounds are overly restrictive and that local-but-unconstrained perturbations can still degrade model performance. If model performance degrades when naive perturbations are applied to features that we would expect to vary naturally, such as inter-arrival times or packet sizes, this suggests that the models are still overfitting to the training data, despite our attempts otherwise.

To verify that this isn't happening and to investigate the relationship between our global bounds and unconstrained, local perturbations, we randomly select a subset of performant models from Section 4.1 (each achieving an accuracy of 0.85+ on the DoS test data) and perturb data via the Fast Gradient Sign [22] and the Momentum Iterative [15] Method.

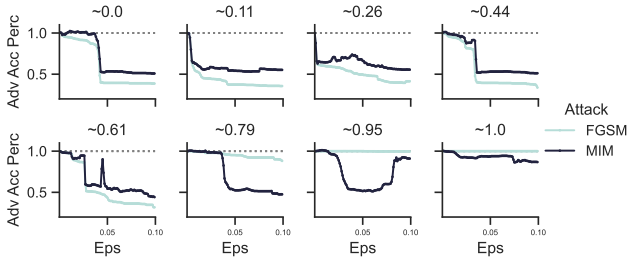


Figure 9: Robustness of the globally constrained models versus unconstrained, local adversarial attacks. Facet titles reflect the approximate adversarial training accuracy.

Results. We present our results in Figure 9. Training on adversarial samples within the specification clearly improves adversarial robustness against the fast gradient sign method, with models scoring highly on adversarially generated data maintaining near perfect accuracy. Whilst the effect is less pronounced for the momentum iterative method, we still see benefits for models with high accuracy on the adversarial training data. [4]

4.8 Coverage Metrics

Experiment. Specifications for DNN verification are usually assumed to be correct. In contrast, in this work, since the specifications are manually engineered, we go a step further and validate them via coverage metrics. Specifically, we check the percentage of flows in the training/test data that satisfy specification bounds.

Table 3: Generalisability of the specifications. Note that 'Malicious*' is calculated on verified attack specifications.

Attack	Property	Dataset	Coverage (#)	Coverage (%)
DoS	Benign	Train	13488/189796	7.11
DoS	Benign	Test	4696/211174	2.22
DoS	Malicious	Train	190507/190663	99.92
DoS	Malicious	Test	221801/221801	100.00
DoS	Malicious*	Train	125087/190663	65.61
DoS	Malicious*	Test	183727/221801	82.83
SSH	Benign	Train	748/6876	10.88
SSH	Benign	Test	123/102624	0.12
SSH	Malicious	Train	6964/6964	100.00
SSH	Malicious	Test	108639/108639	100.00
SSH	Malicious*	Train	2851/6964	40.94
SSH	Malicious*	Test	62402/108639	57.44

Results. Table 3 reports the results of the validation checks. Notably, the complete specifications for malicious traffic cover ≈100% of the malicious traffic from the datasets. Furthermore, while decreasing, the verifiable specifications still cover a high percentage of the datasets (from 40.94% to 82.83%). Lastly, while the coverage of the benign traffic is much lower (from 0.12% to 10.88%), it is an encouraging result as the specifications target only subsets of the benign data, such as SSH traffic, which make up a comparatively small percentage of the dataset. These results confirm the validity of the specifications and that they reflect actual traffic in our datasets.

5 LIMITATIONS

Our approach has several limitations. Current verification frameworks cannot verify deep networks or complex architectures; models in state-of-the-art verification competitions have between approximately 0.5 and 100 million parameters [6]. Our process also requires repeatedly querying models to produce counterexamples, limiting model parameters and depth. Future improvements should improve this situation [42].

Section 3 mentioned that increasing detail of specifications by adding more disjunctions can lead to long verification times, so specifications that carve up the input space too finely become impractical. To account for this, we write specification reflecting commonly occurring feature combinations in the training data, as discussed in Section 3.2. In any case, the human-in-the-loop effort also limits how complex we would want specifications to become; there is a similarity with the use of human-written signatures for IDS which have limits on scope and complexity. Ultimately, because there can be no complete specification of malicious traffic, there is a trade-off between what we want the model to learn and the global constraints we want to specify.

6 RELATED WORK

To the best of our knowledge, while research exists on security applications of DNN verification [5] and on probabilistic verification for DNN-based NIDS [40], no prior work has explored the use of verification frameworks with machine learning-based NIDS.

ML-based NIDS are a well-trodden topic with many competing architectures. Often, classifiers make significant modifications to 'off-the-shelf' models, including CNN-based [16, 41], autoencoder-based [32] and graph-based [38] approaches. These architectures are too complex for current complete verification methods, making behavioural guarantees difficult.

Much work exists on robustness and adversarial examples for NIDS. Zhang et al. [43] provide a thorough overview of attacks and defenses applied to NIDS, including the HopSkipJump attack [10] and ensemble adversarial training [37].

NIDS models are often trained and tested on the same public dataset, but unfortunately many commonly used datasets have flaws [17, 20]. Even simple models with restricted feature sets can be highly performant on common datasets, as demonstrated by Jacobs et al. [25]. But these performant models fail to generalise to other datasets or attacks when withheld as test sets [2, 9]. We aim to improve generalisation via a specification-driven approach.

Existing research on DNN verification primarily focuses on local robustness [8] in computer vision [33] . In contrast, global properties are defined over regions of space not parameterised by inputs [27], making them more general and challenging to prove. Altogether, global properties are less commonly considered in research, with some exceptions. Katz et al. [28] define global robustness specifications for ACAS Xu [26]. Chen et al. [11] also verify global properties for security classifiers. However, unlike our work, their global properties are unrelated to the underlying mechanism of attacks and only bound model outputs, rather than inputs.

7 CONCLUSION

To our knowledge, we are the first to investigate the applications of neural network verification in the network intrusion domain. We do this non-trivially, aiming to improving adversarial robustness, cross-dataset generalisation as well as uncover model weaknesses and failure modes. We design our models, feature sets and verification criteria from the ground up to minimise the effects of low data diversity via targeted restriction and expansion of our input data.

[4] We note that Figure 9 violates some of aspects of the Carlini et al.'s evaluation checklist [7], such as Facet 7 increasing in accuracy for larger ϵ. However, similar to Section 4.5 this is an artefact of clipping the input data. Otherwise, the adversarial attacks behave as expected.

We developed models with considerably improved cross-dataset and cross-attack generalisation compared to standard approaches, whilst gaining strong mathematical guarantees about our models' behaviour in regions of the input space. Our specifications allow us to reason about the complex feature interactions of our models, thanks to choosing a tractable feature set and using the high-level specification language Vehicle. Our work provides insight into how verification frameworks alongside data generation techniques can improve model behaviour. Neural network verification is an emerging area of research and there are many possible avenues for future work, using other tools, as well as exploring local specifications, considering more complex attacks and generalisation conditions.

ACKNOWLEDGEMENTS

We thank our reviewers for their comments which helped improve this paper. This work was partially funded by EPSRC grant EP/T027037/1.

REFERENCES

[1] Giuseppina Andresini, Feargus Pendlebury, Fabio Pierazzi, et al. 2021. Insomnia: Towards concept-drift robustness in network intrusion detection. In *Proceedings of the 14th ACM workshop on artificial intelligence and security*.

[2] Giovanni Apruzzese, Luca Pajola, and Mauro Conti. 2022. The cross-evaluation of machine learning-based network intrusion detection systems. *IEEE Transactions on Network and Service Management* 19, 4 (2022), 5152–5169.

[3] Stanley Bak, Changliu Liu, and Taylor Johnson. 2021. The second international verification of neural networks competition: Summary and results. *arXiv preprint arXiv:2109.00498* (2021).

[4] Teodora Baluta, Zheng Leong Chua, Kuldeep S Meel, et al. 2021. Scalable quantitative verification for deep neural networks. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 312–323.

[5] Teodora Baluta, Shiqi Shen, Shweta Shinde, et al. 2019. Quantitative verification of neural networks and its security applications. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 1249–1264.

[6] Christopher Brix, mark Niklas Müller, Stanley Bak, et al. 2023. First three years of the international verification of neural networks competition. *International Journal on Software Tools for Technology Transfer* 25, 3 (2023), 329–339.

[7] Nicholas Carlini, Anish Athalye, Nicolas Papernot, et al. 2019. On evaluating adversarial robustness. *arXiv preprint arXiv:1902.06705* (2019).

[8] Marco Casadio, Ekaterina Komendantskaya, Matthew L Daggitt, et al. 2022. Neural network robustness as a verification property: a principled case study. In *International Conference on Computer Aided Verification*. Springer, 219–231.

[9] Marta Catillo, Andrea Del Vecchio, Antonio Pecchia, et al. 2021. A Critique on the Use of Machine Learning on Public Datasets for Intrusion Detection. In *International Conference on the Quality of Information and Communications Technology*. Springer, 253–266.

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

[11] Yizheng Chen, Shiqi Wang, Yue Qin, et al. 2021. Learning security classifiers with verified global robustness properties. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 477–494.

[12] Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. 2017. Maximum resilience of artificial neural networks. In *Automated Technology for Verification and Analysis: 15th International Symposium, ATVA 2017, Pune, India, October 3–6, 2017, Proceedings 15*. Springer, 251–268.

[13] Henry Clausen, Robert Flood, and David Aspinall. 2019. Traffic generation using containerization for machine learning. In *Proceedings of the 2019 Workshop on DYnamic and Novel Advances in Machine Learning and Intelligent Cyber Security*.

[14] Matthew L Daggitt, Wen Kokke, Robert Atkey, et al. 2024. Vehicle: Bridging the Embedding Gap in the Verification of Neuro-Symbolic Programs. *arXiv preprint arXiv:2401.06379* (2024).

[15] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, et al. 2018. Boosting adversarial attacks with momentum. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 9185–9193.

[16] Roberto Doriguzzi-Corin, Stuart Millar, Sandra Scott-Hayward, et al. 2020. LUCID: A practical, lightweight deep learning solution for DDoS attack detection. *IEEE Transactions on Network and Service Management* 17, 2 (2020), 876–889.

[17] Gints Engelen, Vera Rimmer, and Wouter Joosen. 2021. Troubleshooting an Intrusion Detection Dataset: the CICIDS2017 Case Study. In *2021 IEEE Security and Privacy Workshops (SPW)*. IEEE, 7–12.

[18] Claudio Ferrari, Mark Niklas Mueller, Nikola Jovanović, et al. 2022. Complete Verification via Multi-Neuron Relaxation Guided Branch-and-Bound. In *International Conference on Learning Representations*.

[19] Thomas Flinkow, Barak A. Pearlmutter, and Rosemary Monahan. 2024. Comparing Differentiable Logics for Learning with Logical Constraints. arXiv:2407.03847 [cs.LO]

[20] Robert Flood, Gints Engelen, David Aspinall, et al. 2024. Bad Design Smells in Benchmark NIDS Datasets. In *2024 IEEE 9th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 658–675.

[21] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, et al. 2018. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE symposium on security and privacy (SP)*. IEEE, 3–18.

[22] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. arXiv:1412.6572 [stat.ML]

[23] Dongqi Han, Zhiliang Wang, Ying Zhong, et al. 2021. Evaluating and improving adversarial robustness of machine learning-based network intrusion detectors. *IEEE Journal on Selected Areas in Communications* 39, 8 (2021), 2632–2647.

[24] Soumyadeep Hore, Jalal Ghadermazi, Diwas Paudel, et al. 2023. Deep packgen: A deep reinforcement learning framework for adversarial network packet generation. *arXiv preprint arXiv:2305.11039* (2023).

[25] Arthur S Jacobs, Roman Beltiukov, Walter Willinger, et al. 2022. AI/ML for Network Security: The Emperor has no Clothes. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 1537–1551.

[26] Kyle D Julian, Jessica Lopez, Jeffrey S Brush, et al. 2016. Policy compression for aircraft collision avoidance systems. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*. IEEE, 1–10.

[27] Anan Kabaha and Dana Drachsler Cohen. 2024. Verification of Neural Networks' Global Robustness. *Proceedings of the ACM on Programming Languages* 8, OOPSLA1 (2024), 1010–1039.

[28] Guy Katz, Clark Barrett, David L Dill, et al. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International conference on computer aided verification*. Springer, 97–117.

[29] Linyi Li, Tao Xie, and Bo Li. 2023. Sok: Certified robustness for deep neural networks. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1289–1310.

[30] Gordon Fyodor Lyon. 2009. *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure.

[31] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, et al. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *International Conference on Learning Representations*.

[32] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, et al. 2018. Kitsune: an ensemble of autoencoders for online network intrusion detection. *arXiv preprint arXiv:1802.09089* (2018).

[33] Kexin Pei, Linjie Zhu, Yinzhi Cao, et al. 2017. Towards practical verification of machine learning: The case of computer vision systems. *arXiv preprint arXiv:1712.01785* (2017).

[34] Iman Sharafaldin, Arash Habibi Lashkari, Ali A Ghorbani, et al. 2018. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp* 1 (2018), 108–116.

[35] Ryan Sheatsley, Blaine Hoak, Eric Pauley, et al. 2021. On the robustness of domain constraints. In *Proceedings of the 2021 ACM SIGSAC conference on computer and communications security*. 495–515.

[36] Vincent Tjeng, Kai Xiao, and Russ Tedrake. 2019. Evaluating robustness of neural networks with mixed integer programming. *n International Conference on Learning Representations*, (2019).

[37] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, et al. 2017. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204* (2017).

[38] Andrea Venturi, Matteo Ferrari, Mirco Marchetti, et al. 2023. ARGANIDS: a novel network intrusion detection system based on adversarially regularized graph autoencoder. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*. 1540–1548.

[39] Junnan Wang, Liu Qixu, Wu Di, et al. 2021. Crafting adversarial example to bypass flow-&ML-based botnet detector via RL. In *Proceedings of the 24th International Symposium on Research in Attacks, Intrusions and Defenses*.

[40] Kai Wang, Zhiliang Wang, Dongqi Han, et al. [n. d.]. BARS: Local Robustness Certification for Deep Learning based Traffic Analysis Systems.

[41] Wei Wang, Yiqiang Sheng, Jinlin Wang, et al. 2017. HAST-IDS: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection. *IEEE access* 6 (2017), 1792–1806.

[42] Haoze Wu, Omri Isac, Aleksandar Zeljić, et al. 2024. Marabou 2.0: A Versatile Formal Analyzer of Neural Networks. arXiv:2401.14461 [cs.AI]

[43] Chaoyun Zhang, Xavier Costa-Perez, and Paul Patras. 2022. Adversarial attacks against deep learning-based network intrusion detection systems and defense mechanisms. *IEEE/ACM Transactions on Networking* 30, 3 (2022), 1294–1311.