Genos: General In-Network Unsupervised Intrusion Detection by Rule Extraction

Ruoyu Li[§†], Qing Li[†], Yu Zhang[§], Dan Zhao[†], Xi Xiao[♮], Yong Jiang[♮†]
[§]Tsinghua University, China; [†]Peng Cheng Laboratory, China
[♮]Tsinghua Shenzhen International Graduate School, China
{liry19,yu-zhang23}@mails.tsinghua.edu.cn;
{liq,zhaod01}@pcl.ac.cn; {jiangy,xiaox}@sz.tsinghua.edu.cn

***Abstract*—Anomaly-based network intrusion detection systems (A-NIDS) use unsupervised models to detect unforeseen attacks. However, existing A-NIDS solutions suffer from low throughput, lack of interpretability, and high maintenance costs. Recent in-network intelligence (INI) exploits programmable switches to offer line-rate deployment of NIDS. Nevertheless, current in-network NIDS are either model-specific or only apply to supervised models. In this paper, we propose Genos, a general in-network framework for unsupervised A-NIDS by rule extraction, which consists of a Model Compiler, a Model Interpreter, and a Model Debugger. Specifically, observing benign data are multi-modal and usually located in multiple subspaces in the feature space, we utilize a divide-and-conquer approach for model-agnostic rule extraction. In the Model Compiler, we first propose a tree-based clustering algorithm to partition the feature space into subspaces, then design a decision boundary estimation mechanism to approximate the source model in each subspace. The Model Interpreter interprets predictions by important attributes to aid network operators in understanding the predictions. The Model Debugger conducts incremental updating to rectify errors by only fine-tuning rules on affected subspaces, thus reducing maintenance costs. We implement a prototype using physical hardware, and experiments demonstrate its superior performance of 100 Gbps throughput, great interpretability, and trivial updating overhead.**

***Index Terms*—intrusion detection, P4 switch, rule extraction**

I. Introduction

The network intrusion detection system (NIDS) has been a crucial network security infrastructure for decades. Among its various categories, anomaly-based NIDS (A-NIDS) that works in an unsupervised manner is drawing more attention. Compared to supervised methods, this type of methods is more promising because 1) it eliminates the need of attack data for training; 2) it does not rely on predefined threat models, improving the detection of unforeseen anomalies. With the advances of various unsupervised machine learning (ML) and deep learning (DL) models, many A-NIDS approaches [1]–[8] have been proposed. While these approaches have demonstrated considerable malicious traffic detection capability, they suffer from a few limitations that hinder their practical use.

Low throughput and high delay. The inference speed of ML/DL models can hardly catch up with the soaring speed of today's high-throughput networks (e.g., 100 Gbps). As a consequence, most of these methods can only work in an off-path deployment fashion (e.g., deploy on a GPU server on control plane), causing prolonged response time.

Limited interpretability. Due to the black-box nature of many ML/DL models, network operators are often reluctant to trust the high-stake decisions made by these models, whose output are typically scores/labels unintuitive for humans.

High updating overhead. Most methods require retraining to update models (e.g., to solve false positives), which is time-consuming and induces extra overhead.

Meanwhile, the recent advance in programmable data plane empowers in-network intelligence (INI), and opens up the possibility of fully deploying ML/DL-based NIDS on switching ASICs for line-speed processing. Prior works have realized the deployment of a series of learning models for intrusion detection, such as neural networks [9]–[11], SVM [12], [13], decision trees [12], [14], and ensemble models [15], [16]. However, existing INI solutions have two problems: 1) the vast majority are only applicable to specific models; 2) the only one general INI method (Mousika [17]) conducts model-agnostic translation to decision trees for deployment, while it only supports supervised models. Currently, there is no work proposing general INI methods for unsupervised A-NIDS.

Designing a general INI framework of A-NIDS faces several challenges. *First*, most existing rule extraction methods (e.g., [17]–[20]) are supervised, mainly due to their heavy reliance on rule extraction models that inherently require labeled data for each class (e.g., CART decision trees). *Second*, existing methods often lack support for incremental updates, as their rules are obtained from the overall distribution of data, which needs to be re-estimated even for small updates. Yet retraining models can lead to significant changes in the extracted rule set, requiring reinstallation of a large number of rules on the switch. *Third*, A-NIDS typically requires more complex features, especially flow-level statistics (e.g., packet size mean/-variance), to achieve sufficient accuracy without using labels. However, current switching ASICs usually have arithmetic constraints (e.g., do not support division) that restrict the support for complex features. As such, most prior methods (e.g., [12], [13], [17]) only consider packet-level features.

In this paper, we present Genos, a general in-network deployment framework for various A-NIDS models. We adopt

Corresponding author: Qing Li (liq@pcl.ac.cn).

a systematic approach, referring to the tools of program development, and design three modules: Model Compiler, Model Interpreter, and Model Debugger. The Model Compiler treats an A-NIDS as the "source model", converts it into a rule set as the "intermediate representation", and forms a set of P4 tables as the "object representation", enabling the efficient deployment of A-NIDS within programmable switches' data plane. The Model Interpreter comes into play when an anomaly is detected. It utilizes the decision logic of the extracted rules and provides explanations for the identified anomalies, which assists network operators in understanding the underlying reasons for the predictions. To address false positives, the Model Debugger can identify the rules responsible for incorrect decisions and incrementally generate a limited number of new rules to rectify the errors.

Due to the multimodal nature of benign data (e.g., a server often supports multiple services), benign samples often locate in multiple disjoint subspaces within the feature space. Following this intuition, Genos resolves the aforementioned challenges with three key designs. First, we design a*divide-and-conquer rule extraction* method. Specifically, we propose a *Score Clustering Tree*, which partitions the feature space according to scores generated by the unsupervised source model into subspaces, each containing samples of similar normality. Then, for each subspace, we design a *Decision Boundary Estimation* approach to obtain axis-aligned rules that accurately approximate the decision boundaries of the source model. Second, we realize incremental updating to rectify errors, which locates errors to the granularity of subspaces, and directly fine-tunes the extracted rules only on the affected subspaces. In this way, we avoid retraining the source model and reduce the number of rules to be reinstalled. Third, we realize a feature extractor on the data plane that supports the acquisition of bidirectional flow-level features, especially finding a workaround to handle the comparison of complex features that cannot be easily computed by switching ASICs.

We implement a prototype on a commodity programmable switch and evaluate the deployment of four distinct A-NIDS models. Compared to four prior works (including Mousika), our rule extraction method achieves better results of 97.79% fidelity and 98.80% detection accuracy. In general, Genos can achieve about 100 Gbps throughput, interpretable detection results, and trivial model updating overhead.

We summarize our contributions as follows:

- A general in-network A-NIDS framework achieving better throughput, interpretability, and incremental update.
- To the best of our knowledge, it is the first work to realize the model-agnostic rule extraction of A-NIDS models to P4 tables in a fully unsupervised manner.
- A prototype on hardware, realizing flow-level feature extraction within switching ASICs' limited operations.

II. Background and Motivation

A. Anomaly-based Network Intrusion Detection (A-NIDS)

A-NIDS is a promising category of NIDS as it requires no attack data and meanwhile can better detect unseen attacks. A-NIDS approaches are typically built upon unsupervised learning models. For example, Mirsky et al. [1] and Li et al. [7] both use autoencoders for intrusion detection. Binbusayyis et al. propose an unsupervised NIDS combining convolutional autoencoder and one-class SVM [21]. In [22], the authors propose an A-NIDS approach based on isolation forest.

Formally, for a d-dimensional feature space $\mathcal{X}$, given a stationary distribution $\mathcal{D}$ of normal traffic, an A-NIDS can be abstracted as a function f that estimates the probability density function of benign data distribution, i.e., $f(\boldsymbol{x}) \approx \mathrm{P}_{\mathcal{X}\sim\mathcal{D}}(\boldsymbol{x})$, and detects anomalies via a threshold $f(\boldsymbol{x}) < \varphi$. In practice, this probability can be translated to different criteria of *anomaly scores*, such as reconstruction error of autoencoders and average traversing path length of isolation forests.

Though A-NIDS has many advantages, there are also some limitations that hinder their practical use: 1) low throughput: on control plane, even extremely efficient approaches like [5] can only achieve 10 Gbps-level throughput; 2) limited interpretability: most approaches only output a score (e.g., mean squared error [3], [6]) or a label (e.g., one-hot label [4]) that cannot explain anomalies in terms of important attributes; 3) high overhead of updating: cumbersome retraining process is required to update models.

B. P4 Switch and In-Network Intelligence

P4 switches [23] are equipped with programmable switching ASICs. They allow customized processing logic inside match-action pipelines through P4 programs, while reaching extraordinary processing speed and throughput (e.g., Tbps). Such advantages prompt the research on deploying ML/DL models directly on P4 switches, which is referred to as *in-network intelligence* (INI).

To realize INI, several computation constraints of P4 need to be considered, such as lack of support for division, float operations and loop operations. Recent works have realized the deployment of different ML models. For example, Xiong et al. transform four ML models into match-action tables to realize INI [12]. Some other works focus on tree-based models for INI as they naturally fit the match-action logic [13]–[16]. As the ML community flourishes, ML/DL models will continue to evolve, and the race of deploying new models will never end. Instead of designing model-specific INI solutions, Mousika [17] first proposes the concept of *general INI*. It adopts the technique of knowledge distillation [18] to translate the knowledge of well-trained models into binary decision trees, enabling indirect deployment of complex models and better flexibility over model-specific methods. However, Mousika can only work for supervised models and cannot be applied to A-NIDS approaches that are unsupervised.

C. Challenges of General INI for A-NIDS

To the best of our knowledge, research on general INI for unsupervised A-NIDS still remains blank. We attribute this vacancy to the following unresolved challenges:

Unsupervised rule extraction. The key technique used by Mousika, the general framework for supervised models, is

(a) Autoencoder (b) Variational autoencoder

Fig. 1: Rules that extract A-NIDS using knowledge distillation (KD) and only benign data suffer huge accuracy loss.

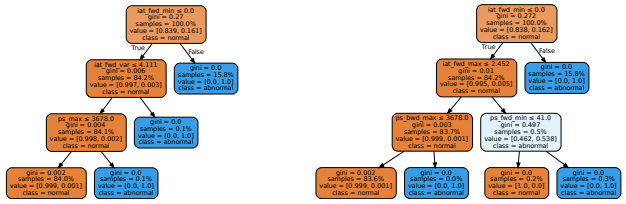
knowledge distillation. In a larger scope, it belongs to model-agnostic *rule extraction* aiming to translate black-box models into interpretable rules. Most of existing methods use CART decision trees as the translation target for rule extraction [19], [20], [24]. They require labeled data for each class to precisely determine the decision logic of the models, which cannot meet A-NIDS' requirement of only using unlabeled benign data. Yet forcing supervised rule extraction methods in a one-class scenario will cause severe accuracy loss. For example, in Fig. 1, compared to the original models, decision trees obtained by knowledge distillation show drastic performance degradation in detecting attacks (i.e., low true positive rate).

High overhead of update. Deployed models sometimes need updates, e.g., when network operators find alarms are false positives and expect to reduce similar cases. Typically, we need to retrain the models and re-obtain the rules. However, many rule extraction methods [18]–[20], [25] adopt decision trees, which inherently do not support incremental updates. Retraining tree models, even using a small scale of data, can cause remarkable changes in tree structure and rules. Fig. 2 illustrates such an example. As such, a large number of deployed rules have to be deleted and reinstalled to fix a small number of errors. Worse, this process can cause certain downtime of a switch, increasing the risk of being infiltrated.

Acquisition of flow-level features. A-NIDS approaches usually need sophisticated features to precisely represent normal traffic, particularly flow-level features. Most methods (e.g., Planter [16], Mousika [17]) only support packet-level feature extraction on P4 switches. A more recent work [26] first realizes stateful flow-level features, including statistics like mean and variance, by using bit shift operations to replace division operations. However, this approach can only summarize a flow at a fixed inference length of a power of 2, which can be insufficient for detecting time-related attacks. For example, there might be little difference between the bytes of the first four packets of a normal HTTP request and a CC attack.

III. Overview

This paper proposes a novel framework, Genos, which implements a complete life cycle for general in-network deployment of A-NIDS. We observe that benign samples often locate in multiple disjoint subspaces within the feature space, due to the multimodal nature of benign data. Therefore, we adopt a divide-and-conquer approach, and extract rules on the granularity of subspace. This not only enables accurate approximation of the source model, but also allows incremental updating by only fine-tuning rules on the affected subspaces.



(a) Before update (b) After update

Fig. 2: Updating false positives with only 0.24% of data causes changes in tree structure and splitting criteria.

Fig. 3 shows the overview. We refer to the concepts in program development and design three modules running on control plane: 1) Model Compiler that translates an A-NIDS model into P4 tables by rule extraction; 2) Model Interpreter that explains important attributes for decisions; 3) Model Debugger that fixes wrong decisions by incremental updates.

Model Compiler. This module achieves the translation of a black-box A-NIDS model into a set of rules, which are subsequently transformed into match-action tables for efficient in-network deployment. We propose a model-agnostic rule extraction algorithm that addresses the challenge of unsupervised model extraction. The algorithm comprises a Score Clustering Tree to partition the feature space into subspaces, and a Decision Boundary Estimation approach to determine the decision rules of the source model on each subspace.

Model Interpreter. This module interprets the opaque output of A-NIDS models (e.g., anomaly scores) by feature importance. Considering that rule extraction inherently provides a global explanation for the decision logic of the source model, we realize an effective and efficient local explanation method, i.e., explaining the detection of one anomaly at a time, based on the feature deviations of extracted rules.

Model Debugger. Thanks to our divide-and-conquer rule extraction, we design this module that can pinpoint the rules producing errors and incrementally update them. We develop an *excluding* mode for the scenarios where data go into some subspace misidentified as anomalies inherently by the source model, and a *patching* mode for the scenarios where errors are attributed to insufficient generalization of the rule in a certain subspace. As a result, only a small number of rules from the affected subspaces need to be reinstalled on switches to increase accuracy, reducing the overhead of updates.

Besides, Genos realizes bidirectional flow-level feature extraction on the data plane. We manage to compare mean and variance values to rule thresholds without division. Particularly, we design an adaptive timeout mechanism for flow length determination, executing *active* timeout for long lived flows and *inactive* timeout for burst flows, promoting more timely and accurate representations of flows.

IV. Framework Design

A. Model Compiler

1) Design Goal: When designing the Model Compiler, we aim to derive an in-distribution rule set $\mathcal{C} = \{C_1, C_2, ...\}$ from

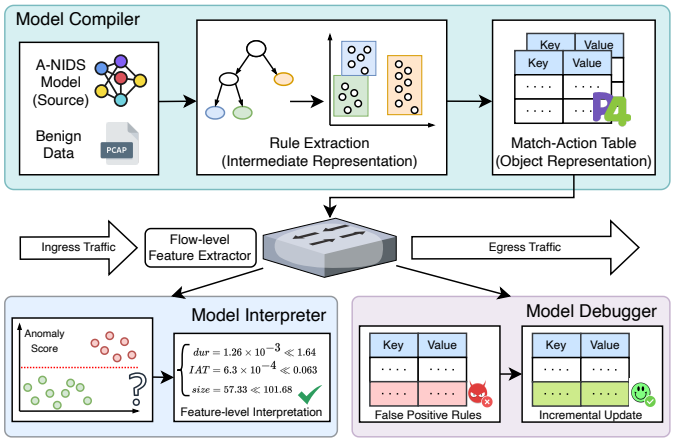


Fig. 3: Overview of Genos.

a trained A-NIDS model f, utilizing its anomaly threshold φ and benign training data $\boldsymbol{X}$. To ease the final deployment on P4 switches, we focus on axis-aligned rules that involve straightforward comparison and exclude rules of other formats to avoid additional calculations (e.g., linear models [27]). Each rule $C = ... \wedge (x_i \perp v_i) \wedge ... \wedge (x_j \perp v_j)$ represents a conjunction of feature constraints, where v_i denotes the bound for the i-th dimension; $\perp \in \{\leq, >\}$ denotes the comparison. We represent a data sample satisfying a rule as $\boldsymbol{x} \in C$. The anticipated outcome entails the derived rules to exhibit a substantial fidelity to the source model, implying a comparable coverage of benign data and a similar detection rate of anomalies:

$$\arg\min_{\mathcal{C}} \mathcal{L}_{\mathcal{X}\sim\mathcal{D}}(\mathcal{C}, f, \varphi) + \mathcal{L}_{\mathcal{X}\nsim\mathcal{D}}(\mathcal{C}, f, \varphi), \tag{1}$$

where $\mathcal{D}$ is the stationary distribution of benign data, and $\mathcal{L}$ is a loss function to measure fidelity in a certain space.

2) Rule Extraction: The biggest challenge in solving the problem in (1) is the minimization of the second term. Without labeled abnormal samples, this term is neither deterministic nor can be easily estimated by sampling points in a high-dimensional large space. The essence of the challenge comes from the multimodal nature of normal data. For example, a server supports multiple services such as web, email, and database, each represented by distinct features that are possibly located in separate regions within the feature space. The limited transition between these regions makes axis-aligned rules incompetent in accurately approximating the decision boundary of the source model.

Motivated by the above intuition, we design a divide-and-conquer strategy. The primary concept revolves around partitioning the space into subspaces that encapsulate normal data with more compact distributions and then using axis-aligned rules to approximate the source model on each subspace. To this end, we design a tree model for problem breakdown and an estimation approach for problem resolution.

Score Clustering Tree. We notice that, even in the absence of labeled data, the output of A-NIDS models (i.e., anomaly scores) can serve as a valuable indicator to guide the partitioning of the feature space into subspaces for compact distribution of normal data. Built upon the CART decision tree [28], Score Clustering Tree (SCT) introduces a new splitting criterion. Given the data $\boldsymbol{N}$ at a tree node, SCT first obtains the output of the source model $f(\boldsymbol{x})$ for each $\boldsymbol{x} \in \boldsymbol{N}$. SCT seeks a splitting point s for the node that maximizes the gain:

$$\arg\max_{s} I(\boldsymbol{N}) - \frac{|\boldsymbol{N}_l|}{|\boldsymbol{N}|} I(\boldsymbol{N}_l) - \frac{|\boldsymbol{N}_r|}{|\boldsymbol{N}|} I(\boldsymbol{N}_r), \tag{2}$$

where s includes the splitting feature and threshold, $\boldsymbol{N}_l$ and $\boldsymbol{N}_r$ are the data split to the left and right child nodes, respectively, and $|\boldsymbol{N}|$ denotes the number of data samples. I is the Gini index calculated by $1 - \sum p_j^2$, where p_j originally is the probability of each class on this node. Given that we only have unlabeled benign data, we modify p to be the average output scores of the source model:

$$p = \frac{1}{|\boldsymbol{N}|} \sum f(\boldsymbol{x}), \forall \boldsymbol{x} \in \boldsymbol{N}. \tag{3}$$

Unlike clustering techniques that rely solely on Euclidean distance (e.g., K-means), our approach takes the predictions of the source model to acquire a more profound understanding of benign data distribution, thus can more effectively categorize related benign data into one subspace. A tree splits nodes until it satisfies one of the conditions: i) the number of data samples at the node $|\boldsymbol{N}| = 1$; ii) output scores between any of the data samples at the node are below a limit, i.e., $\forall \boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)} \in \boldsymbol{N}$, $|f(\boldsymbol{x}^{(i)}) - f(\boldsymbol{x}^{(j)})| < \epsilon$; iii) it reaches the maximum depth τ.

While our tree primarily serves as a space-splitting mechanism, it can also directly identify certain subspaces as anomalies. We define two types of leaf nodes in our tree: anomalous leaf nodes and unlabeled leaf nodes. The former is assigned only when all the data samples at a leaf node possess output scores lower than the threshold set by the source model, i.e.,

$$y_{\boldsymbol{N}} = 1 \text{ iff } |\{\boldsymbol{x} \in \boldsymbol{N}; f(\boldsymbol{x}) < \varphi\}| = |\boldsymbol{N}|. \tag{4}$$

With benign training data, such cases may arise when outliers are present within the data or if the source model exhibits false positives due to inadequate generalization. For the latter scenario, we will discuss how to rectify errors in the Model Debugger. For unlabeled leaf nodes, we use the following method to further extract their rules of normality.

Decision Boundary Estimation. For each of the subspaces determined by a leaf node, we design an approach using axis-aligned rules to approximate the decision boundary of the source model. Let $\boldsymbol{X}_k$ represent the training data falling into the k-th leaf node. To begin with, we utilize the minimal hypercube H_k as a reference to bound each dimension of the data samples in $\boldsymbol{X}_k$, which are identified as normal by the source model, thus ensuring $\mathcal{L}_{\boldsymbol{x}\in\boldsymbol{X}_k}(H_k, f, \varphi) = 0$. The minimal hypercube H_k is encompassed by $2 \times d$ axis-aligned hyperplanes, and can be described using the following rule:

$$\begin{aligned} H_k &= (v_1^- \leq x_1 \leq v_1^+) \wedge ... \wedge (v_d^- \leq x_d \leq v_d^+), \\ v_i^- &= \min(x_i | f(\boldsymbol{x}) > \varphi, \boldsymbol{x} \in \boldsymbol{X}_k), \\ v_i^+ &= \max(x_i | f(\boldsymbol{x}) > \varphi, \boldsymbol{x} \in \boldsymbol{X}_k), \end{aligned} \tag{5}$$

where x_i denotes the i-th dimension of features in $\boldsymbol{x}$.

To explore the decision boundary, for the i-th dimension, we first conduct uniform sampling of N_e data points on each hyperplane of the hypercube, denoted as $\boldsymbol{e}^{(1)}, \ldots, \boldsymbol{e}^{(N_e)} \in H_k \wedge (x_i = v_i), v_i \in \{v_i^-, v_i^+\}$. These data points are referred to as *initial explorers*. For each initial explorer $\boldsymbol{e}$, we further generate N_s *auxiliary explorers* in its vicinity, drawn from a truncated multivariate Gaussian distribution denoted as $\mathcal{N}(\boldsymbol{e}, \boldsymbol{\Sigma}, i)$. The center of sampling is an initial explorer $\boldsymbol{e}$, and the sampling radius is determined by the covariance matrix:

$$\boldsymbol{\Sigma} = \text{diag}(\rho|v_1^+ - v_1^-|, \ldots, \rho|v_d^+ - v_d^-|), \tag{6}$$

where ρ is a hyperparameter to control the sampling radius. The sampling along the i-th dimension is half-truncated to ensure that only samples outside the hypercube are retained, in an attempt to extend the boundary. With $N_e \times N_s$ auxiliary explorers in total, we query the source model and utilize Beam Search to select N_e samples with the lowest output scores, indicating their proximity to the model boundary. To guarantee fast convergence towards the boundary, we refer to Fast Gradient Sign Method (FGSM) [29] for adversarial attacks and design an approximation approach for black-box scenarios. Its basic idea is to move towards the opposite direction to model training. As an initial explorer $\boldsymbol{e}$ and its auxiliary explorer $\hat{\boldsymbol{e}}$ are spatially close, we can assume the model score is monotonous between the two points, and approximate the score's gradient at their midpoint by calculating the slope between the two points. Similar to FGSM, we subtract the sign of gradient from the midpoint as the new initial explorer for the next iteration:

$$\begin{aligned} &\boldsymbol{e}_{next} = \boldsymbol{e}_{mid} - \eta \cdot sign(\nabla_{\boldsymbol{e}_{mid}}), \\ &\boldsymbol{e}_{mid} = \frac{\boldsymbol{e} + \hat{\boldsymbol{e}}}{2}, \nabla_{\boldsymbol{e}_{mid}} = \frac{\nabla f(\boldsymbol{e_{mid}})}{\nabla \boldsymbol{e_{mid}}} \approx \frac{f(\boldsymbol{e}) - f(\hat{\boldsymbol{e}})}{\boldsymbol{e} - \hat{\boldsymbol{e}}}, \end{aligned} \tag{7}$$

where $sign(\cdot)$ is the sign function, and η controls the stride. The iteration stops when it meets one of the two conditions:
i) an auxiliary explorer $\hat{\boldsymbol{e}}_{last}$ that satisfies $f(\hat{\boldsymbol{e}}_{last}) < \varphi$ is found. For the i-th dimension, we establish a constraint to extend the boundary of the hypercube, i.e., $c_i = (x_i \leq \hat{e}_{last,i})$ if $\hat{e}_{last,i} > v_i^+$, or $c_i = (x_i > \hat{e}_{last,i})$ if $\hat{e}_{last,i} < v_i^-$;
ii) it reaches the maximum iterations, suggesting the difficulty in moving towards the decision boundary by perturbing a particular feature dimension. We calculate the difference between the model output for the last auxiliary explorer $\hat{\boldsymbol{e}}_{last}$ and that of the first initial explorers $\hat{\boldsymbol{e}}$. If $|f(\hat{\boldsymbol{e}}) - f(\hat{\boldsymbol{e}}_{last})| < \delta$, we conclude that this dimension represents a *contour line* for the source model. In this case, we do not produce any constraints for this dimension. Otherwise, we generate constraints in the same manner as those produced under the first condition.

The final rule is obtained by taking the disjunction of the hypercube and the constraints on each dimension:

$$C_k = H_k \vee (c_1 \wedge c_2 \wedge ... \wedge c_d). \tag{8}$$

The rule extraction algorithm is presented in Algorithm 1. The result rule set comprises both the complementary rule for anomalous leaf nodes (lines 4∼6), and the conjunction of the top-down tree rules and the rules obtained through boundary estimation for each unlabeled leaf node (lines 8∼10).

Algorithm 1: Rule Extraction from A-NIDS

Input: A-NIDS model f and threshold φ; dataset $\boldsymbol{X}$
Output: axis-aligned rule set $\mathcal{C}$

```
1  Initialize an empty set C;
2  T ← ClusteringTree(X, f, φ);
3  for leaf node N in T do
4      C_N, y_N ← Root2LeafRule(T, N);
5      if y_N = 1 then
6          C.append(¬C_N);
7      else
8          C_be ← BoundEstimate(N, f, φ);
9          C ← C_N ∧ C_be;
10         C.append(C);
11 end for
12 return C;
```

3) Translation to P4 Table: Due to the axis-aligned nature, our extracted rules can be readily translated into P4 tables using range matching for each feature dimension, as shown in Listing 1. The majority of the table serves as an allowlist except for the rules derived from anomalous leaf nodes. Flows failing to match any rules are set as anomalies. We also adopt prior efforts [17], [26] to further encode range matching into ternary matching to enhance compatibility across switches.

```
table anids_rules {
    key = {
        meta.f_1: range;
        meta.f_2: range;
        ...
        meta.f_d: range;
    }
    actions = {set_benign, set_anomalous}
    default_action = {set_anomalous;}}
```

Listing 1: P4 table for extracted rules.

B. Model Interpreter

This module demystifies the opaque outputs of A-NIDS to improve human understanding and trust in model decisions. This process is known as *local interpretation*, typically outputting the most important features related to a decision. Existing works in this field [30]–[33] often involve fitting a local linear model (e.g., lasso) to uncover the feature weights, which can be slow and unsuitable for delay-sensitive detection tasks, e.g., real-time online security analysis. Our method can provide accurate and fast interpretations thanks to the inherent global interpretation provided by our rule extraction, which captures the decision logic of the source model and allows for the interpretation of individual decisions.

Considering a data sample $\boldsymbol{x}$, its prediction $y_{\boldsymbol{x}}$ and the extracted rule set $\mathcal{C}$, the goal of Module Interpreter is to output a feature importance vector $\boldsymbol{p}_{\boldsymbol{x}}$ to explain the prediction. The process is described in Algorithm 2. In line 2, we first pinpoint the rule corresponding to $\boldsymbol{x}$. As SCT splits the feature space into subspaces by leaf nodes, the rules obtained on each leaf node are disjoint, resulting in at most one rule matching a given data sample. For a normal prediction, its important

Algorithm 2: Interpretation by extracted rules

Input: data sample $(\boldsymbol{x}, y_{\boldsymbol{x}})$, extracted rule set $\mathcal{C}$
Output: feature importance vector $\boldsymbol{p}_{\boldsymbol{x}}$

1 Initialize a zero vector $\boldsymbol{p}_{\boldsymbol{x}}$;
2 $C \leftarrow \text{FindRule}(\boldsymbol{x}, \mathcal{C})$;
3 **for** i **in** $\boldsymbol{x}.size$ *and* $(v_i^- \leq x_i \leq v_i^+) \in C$ **do**
4 **if** $y_{\boldsymbol{x}} = normal$ **then**
5 $\boldsymbol{p}_{\boldsymbol{x}}[i] \leftarrow 1/(v_i^+ - v_i^-)$;
6 **else**
7 $\boldsymbol{p}_{\boldsymbol{x}}[i] \leftarrow \text{ReLU}(x_i - v_i^+) + \text{ReLU}(v_i^- - x_i)$;
8 **end for**
9 **return** $\boldsymbol{p}_{\boldsymbol{x}}$;

feature should be a distinct space that anomalies are unlikely to fall in. Thus, we use the reciprocal of each feature constraint range as the importance (line 5). For an abnormal prediction, as our rules describe the space of normality, the features that do not conform to the rules reveal the anomaly, and higher deviations indicate more anomalous. So we use the distance between the feature and its violated bound as the weight (line 7). If there is no constraint on a feature, we will not assign weights to this feature. All feature values and rule bounds are normalized so that the computed weights are comparable.

C. Model Debugger

This module updates the deployed model to address errors, especially false positives. Unlike the conventional updating pipeline which involves model retraining and redeployment, our method circumvents these steps by directly operating on the extracted rules. This is made possible by our rule extraction algorithm, which isolates the model extraction from the overall feature space into small subspaces.

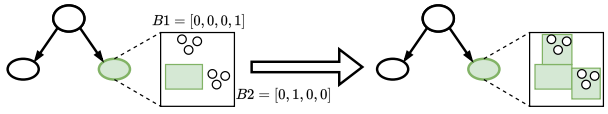
Suppose a network operator finds a batch of reported anomalies are false positives, denoted by $\boldsymbol{X}_{fp}$. Depending on the types of leaf nodes on which the samples are identified, our Model Debugger adopts two different modes (Fig. 4):

Patching Mode. False positives that happen on an unlabeled leaf node may result from our rule extraction not being sufficiently generalized to approximate the real decision boundary. Despite narrowing the overall feature space to small subspaces via our tree model, axis-aligned hyperplanes might still not perfectly fit the complex source model, especially when it is far from linear. In this case, our method incrementally generates a set of new rules, referred to as *patches*. False positive samples that arrive at the same leaf node may violate the rule of the leaf node with respect to different feature constraints. For each sample $\boldsymbol{x} \in \boldsymbol{X}_{fp}$, we use a $2 \times d$-length bitmap $B_{\boldsymbol{x}}$ to record the dimensions that the sample does not conform to:

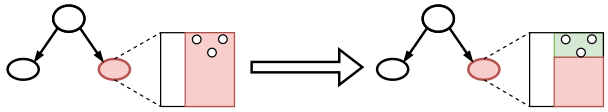
$$B_{\boldsymbol{x}}[i] = \begin{cases} 01, & x_i > v_i^+, \\ 10, & x_i < v_i^-, \end{cases} \tag{9}$$

where v_i^- and v_i^+ represent the upper and lower bounds of the rule's constraint on the i-th dimension.

Next, we employ our Decision Boundary Estimation method to generate a new rule for samples that have identical bitmaps.



(a) Patching Mode on unlabeled leaf node



(b) Excluding Mode on anomalous leaf node

Fig. 4: Illustration of Model Debugger to fix false positives.

These patch rules not only involve the false positives but also explore the boundary to a more reasonable space, improving the generalization of the rules and better mimicking complex and nonlinear decision boundaries of the source model.

Excluding Mode. This mode applies when false positives occur on an anomalous leaf node due to the source model's misidentification of samples in this subspace. As a result, our tree model also labels this leaf node as anomalous. Since this subspace already falls outside the source model's decision boundary, our decision boundary estimation method cannot generate a patching rule. Instead, we use a minimal hypercube to exclude the area containing the false positive samples from the subspace, effectively creating a rule that removes the influence of these misidentified samples from the leaf node.

D. Data Plane Implementation

We implement the data plane of Genos on a Tofino programmable switch (1400 lines of P4 code). We detail the implementation of the flow-level feature extractor, which realizes the 30 flow-level features list in Table I on the data plane.

Bidirectional Flow Record. Like many control plane flow analyzers (e.g., [34]), our implementation on the data plane also records bidirectional flows rather than unidirectional flows to better represent flow patterns. When a packet arrives, we parse its 5-tuple, packet size, and timestamp. Based on the packet direction (e.g., LAN to WAN), we use two hash tables to record the flow: a forward table and a backward table, where the keys are the forward and backward 5-tuples, respectively.

Stateful Storage. Each hash table entry points to a register array for stateful storage. Each register contains an incremental statistic of the flow. The first five statistics are space-related, including 1) packet counts M; 2) sum of packet sizes LS_s; 3) squared sum of packet sizes SS_s (data plane supports the approximate square calculation); 4) maximum packet size Max_s; 5) minimum packet size Min_s. In addition, we record the timestamp of the first packet t_1 and the timestamp of the last packet t_M in the flow. Using these timestamps, we can calculate the flow duration and inter-arrival time between packets, which enable us to preserve four time-related incremental statistics: 1) sum of inter-arrival times LS_t; 2) squared sum of inter-arrival times SS_t; 3) maximum inter-arrival time Max_t; 4) minimum inter-arrival time Min_t. The calculation of these time-related statistics can be implemented in two stages.

TABLE I: Flow-level features implemented on data plane.

Attribute	Statistics	Direction	Number
packet count	$1\sim m$(active timeout)	fwd, bwd, both	3
packet size	mean/max/min/var	fwd, bwd, both	12
inter-arrival time	mean/max/min/var	fwd, bwd, both	12
flow duration	microsecond	both	1
destination port	0∼65535	fwd	1
L4 protocol	TCP or UDP	-	1

Timeout Mechanism. Referring to the classic NetFlow [35], we design an adaptive timeout mechanism for flow length determination, which has two types of timeouts: 1) active timeout, which segments long flows to reduce persistent storage occupation, and is triggered when flow packet count reaches a threshold m; 2) inactive timeout, which identifies burst flows and is triggered when inter-arrival time exceeds a threshold value Δ. This mechanism offers dynamic flow length determination, enabling a more comprehensive understanding of flow patterns.

Feature Acquisition. When a timeout occurs, we retrieve the statistics of the flow from the register array to build a feature vector for model inference. Since a register can only be accessed once in the pipeline, we utilize the *resubmit* mechanism to obtain register values. For features like L4 protocol, port, packet count, and maximum/minimum statistics, we can directly obtain them from the packet header and registers. The most challenging features are the mean and variance values, which cannot be computed by switching ASICs. We notice that their comparison to rule thresholds (v_i), i.e.,

$$\mu_i = \frac{LS}{M} \perp v_i, \quad \sigma_i = \frac{SS}{M} - (\frac{LS}{M})^2 \perp v_i, \tag{10}$$

where $\perp \in \{\leq, >\}$, can be rearranged into:

$$LS \perp M \cdot v_i, \quad M \cdot SS - LS^2 \perp M^2 \cdot v_i, \tag{11}$$

where $M \cdot v_i$ and $M^2 \cdot v_i$ can be computed in advance. In this way, we achieve the comparison without division operation. We pre-encode $M \cdot v_i$ and $M^2 \cdot v_i$ in P4 tables as rule thresholds for every possible value of M (i.e., 1 to m at most due to active timeout). While this may increase the number of table entries, it remains acceptable as our extracted rules are efficient.

V. Evaluation

A. Experimental Setup

We use two benchmark datasets for network intrusion detection: CIC-IDS [36] and TON-IoT [37]. These datasets are in PCAP files, providing sufficient benign data (e.g., server traffic) and a wide range of realistic attack traffic (e.g., DDoS, scanning, botnet). The datasets are randomly split into training (40%), validation (30%), and testing (30%) sets by flows.

We adopt four types of unsupervised models commonly as A-NIDS source models, including autoencoder (AE), variational autoencoder (VAE), one-class SVM (OCSVM), and Isolation Forest (iForest). These models are trained as well as hyperparameter calibration using only benign data. The performance of these models is evaluated using the Area Under the ROC Curve (AUC) score. Table II provides a description of the datasets, along with the AUC scores achieved by the A-NIDS models. We use grid search to decide our hyperparameters.

TABLE II: Datasets and A-NIDS models.

Characteristics		CIC-IDS	TON-IoT
PCAP size		40.1GB	6.27GB
#Attack types		6	9
#Normal samples		687,565	309,086
#Attack samples		288,404	893,006
source model AUC	AE	0.9921	0.9998
	VAE	0.9901	0.9998
	OCSVM	0.9967	0.9993
	iForest	0.9879	0.9877

TABLE III: Performance on each type of attacks.

Attack	AE		VAE		OCSVM		iForest	
	TPR	TNR	TPR	TNR	TPR	TNR	TPR	TNR
backdoor	1.000	0.9772	1.000	0.9715	1.000	0.9857	1.000	0.9686
DDoS	1.000	0.9832	1.000	0.9832	1.000	0.9851	1.000	0.9767
DoS	1.000	0.9849	1.000	0.9849	1.000	0.9915	1.000	0.9831
injection	1.000	0.9888	1.000	0.986	1.000	0.9795	1.000	0.9701
MITM	1.000	0.9776	1.000	0.9669	1.000	0.9835	1.000	0.9582
brute force	1.000	0.9816	1.000	0.9903	1.000	0.9874	1.000	0.9709
ransomware	1.000	0.9962	1.000	0.9808	1.000	0.9789	1.000	0.9607
scanning	1.000	0.9904	1.000	0.9865	1.000	0.9846	1.000	0.9605
XSS	1.000	0.9862	1.000	0.9882	1.000	0.9774	1.000	0.9518
#Rules	29		27		17		17	

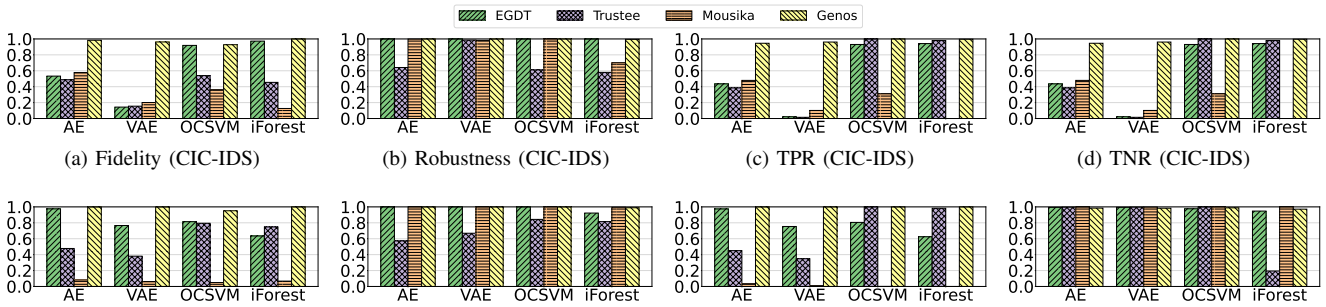
B. Model-Agnostic Rule Extraction

Metrics. We use four metrics to evaluate the performance of rule extraction: 1) fidelity, the ratio of data samples on which the predictions of extracted rules are identical to the predictions of the source model; 2) robustness, the ratio of data samples added with a perturbation, on which the predictions of extracted rules are identical to the predictions of the source model; 3) true positive rate (TPR) and 4) true negative rate (TNR), indicating accuracy in unbalanced data scenarios.

Baselines. We compare our method to three baselines:
1) *Estimated Greedy Decision Tree (EGDT)* [20]: It extracts a decision tree that actively samples new training points to mirror the computation performed by the source model.
2) *Trustee* [19]: It synthesizes high-fidelity and low-complexity tree models that can demystify black-box model decisions.
3) *Mousika* [17]: It uses knowledge distillation to translate the knowledge of a complex model into a binary decision tree.

All baselines are comparable to ours since they are model-agnostic tree-based models that generate axis-aligned rules and can be deployed on programmable switches. Extracting rules using these methods can only access benign datasets.

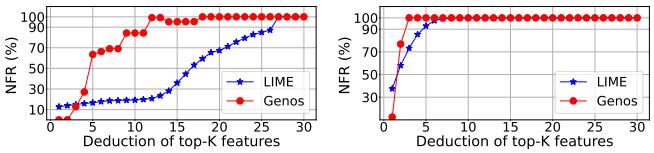
Fig. 5 shows Genos achieves superior performance in all metrics. First, it achieves the highest fidelity across all source models and datasets, with some scores even surpassing 0.999. It indicates that Genos can accurately capture the decision logic and precisely match the predictions of the source models. Second, Genos achieves the highest TPR across all the source models and datasets, with a TPR of 1.00 for all the models on the TON-IoT dataset. This exceptional result signifies that our rules can accurately detect various anomalous traffic. In contrast, most baselines fail to obtain adequate TPR since they rely on much anomalous data to determine decision boundaries, which may not be readily available in practice. Third, the robustness of Genos is also commendable, with scores ranging from 0.9890 to 1.00, demonstrating its ability to handle perturbations in practical environments. Moreover, the TNR is consistently high (0.9715 to 1.00).



(a) Fidelity (CIC-IDS) (b) Robustness (CIC-IDS) (c) TPR (CIC-IDS) (d) TNR (CIC-IDS)

(e) Fidelity (TON-IoT) (f) Robustness (TON-IoT) (g) TPR (TON-IoT) (h) TNR (TON-IoT)

Fig. 5: Comparison of rule extraction performance on four A-NIDS models using two traffic datasets.



(a) CIC-IDS (b) TON-IoT

Fig. 6: Feature deduction test on LIME and Genos.

We also study the performance of our extracted rules in detecting each type of attacks. Table III shows our rules can perfectly detect all types of attacks using any of the source models (TPR=1.000). The TNR also reaches a satisfying level (0.981 on average). Such a remarkable detection performance is owing to the high efficacy of our method, which extracts fewer than 30 rules, making it practical for deployment on resource-constrained network devices.

C. Interpreting Decisions

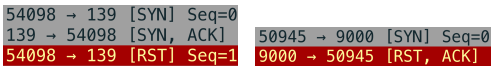
We compare the Model Interpreter of Genos to LIME [27], a state-of-the-art model-agnostic explanation method. Following prior works (e.g., [32]), we conduct a feature deduction test. Intuitively, if top-K features are selected as important for a decision, removing these features from this data sample would probably lead to misclassification by the source model. As we only access benign data for training, the selected features are essentially significant attributes of benign data. Thus, we use the Negative Flipping Rate (NFR) as the metric, which measures the ratio of the samples that are initially predicted as normal by the source model and are predicted as abnormal after nullifying the selected top-K features.

Fig. 6 depicts Genos reaching high NFR faster than LIME on both datasets. On TON-IoT dataset, Genos attains 100% NFR with only the top three features. In terms of efficiency, Genos is over 2000 times faster than LIME ($0.168\mu s$ v.s. $478.206\mu s$ on average) when interpreting a decision. We attribute the superiority to our rule extraction, which can well describe the feature ranges of normal data, and accurately and efficiently interpret decisions by deviations of feature values.

Taking two samples from TON-IoT datasets that scan attack attempts as examples, Table IV presents the interpretation of Genos. One sample successfully scans an open port while the other fails. As the normal traffic mainly consists of MQTT brokers with a limited range of ports, the wide scanning range of ports is a strong indicator for scanning attacks. Another important feature is the flow packet count: one sample has three packets, while the other has two. Manual data analysis, as depicted in Fig. 7, also reveals a clear distinction: a successful scanning attack receives an ACK from the victim, while scanning on a closed port is directly reset. Therefore, this feature captures the distinct patterns between the two attack scenarios. Overall, it demonstrates that the interpretation provided by Genos is aligned with expert knowledge.

TABLE IV: Interpretation of two scanning attack samples.

Successful scanning			Unsuccessful scanning		
Top Feature	Value	Rule	Top Feature	Value	Rule
dest_port	139	> 1882	dest_port	9000	≤ 1883
duration	0.0002	> 13.082	duration	0.663	> 13.082
count	3	> 116.48	count	2	> 116.48



(a) (b)

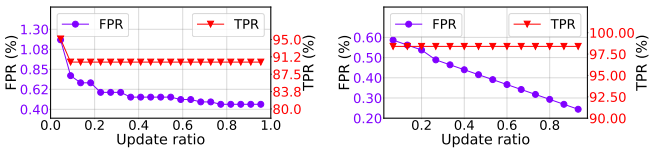
Fig. 7: Two scanning attack samples analyzed by Wireshark.

D. Updating Rules

We first use training sets for model training and rule extraction, and apply extracted rules to the detection of validation sets. We then employ the Model Debugger to update the extracted rules using these false positives, and finally evaluate the updated rules on testing sets.

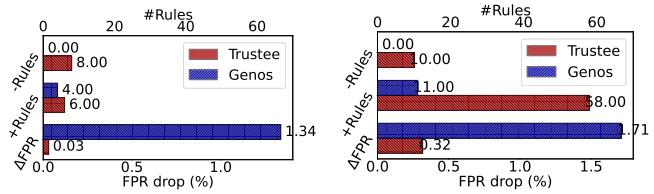
In Fig. 8, as we include more false positives in rule updates, the FPR consistently decreases, showing Genos can learn from mistakes to enhance its decision-making. Meanwhile, the TPR remains steady, indicating the persistent detection accuracy for anomalies even when rectifying false positives. This highlights the adaptability and resilience of Genos in evolving network environments, which is vital for the usability of NIDS.

To evaluate the overhead of updates, we employ Trustee as a baseline, which is based on CART decision trees for rule extraction. Since incremental update is not viable for Trustee, we update its rules by retraining the source model and re-extracting the rules. Fig. 9 shows Genos demonstrates a



(a) CIC-IDS (b) TON-IoT

Fig. 8: Updating with incremental ratios of false positives.



(a) CIC-IDS (b) TON-IoT

Fig. 9: Change of rule numbers and FPR after updates.

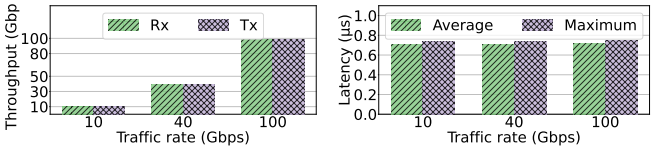
higher FPR drop than Trustee with fewer rule changes. Thanks to the incremental update mechanism supported by our rule extraction method, Genos can efficiently reduce the overhead of updates since only a few new rules need to be added. In contrast, the rule set of Trustee drastically changes since the updated model changes the fundamental tree structure.

E. Hardware Performance

We install the P4 tables obtained by the Model Compiler and the P4 program that realizes the flow-level feature extractor and rule matching on the hardware switch. We use a traffic generator (Keysight XGS12) to generate high-speed traffic and evaluate the hardware performance with respect to throughput, processing latency, and resource consumption.

We conduct experiments with loads of 10 Gbps, 40 Gbps, and 100 Gbps. As shown in Fig. 10, Genos exhibits exceptional performance, maintaining a perfect match between switch throughput and traffic rate. This ensures that the switch can efficiently process incoming traffic without bottlenecks or packet loss. Further, the processing latency is impressively low, at around 0.74 microseconds (1μs=10^{-6}s). In comparison to A-NIDS deployment on the control plane, even a highly efficient approach using DPDK [5] can only achieve 12.65 Gbps of throughput and 0.047 seconds of latency. The results show that Genos, through its general in-network approach, achieves high throughput and real-time online intrusion detection.

For memory resources, Genos occupies 16.9% SRAM and 0.3% TCAM. For computation resources, Genos utilizes 10 stages, 16.3% eMatch Xbar, 0.5% tMatch Xbar, 26.6% Hash bit, and 52.1% ALU. The relatively heavy utilization of the last two resources is attributed to the flow-level feature extraction mechanism. Note that the basic packet forwarding function of switches does not use ALU extensively, so it will not be significantly affected by Genos. Overall, Genos is efficient and practical for deployment on switch ASICs.



(a) Throughput (b) Latency

Fig. 10: Runtime performance of Genos.

VI. Related Work

A. Programmable Data Plane

There have been other works focusing on offloading various network tasks on programmable data planes, such as measurement [38]–[40], load balancing [41], RTT monitoring [42], failure detection [43], and DDoS mitigation [44], [45]. The primary distinction between these methods and ours lies in the approach they take to perform the tasks. While the aforementioned works mainly depend on threshold-driven logic, our framework utilizes ML/DL models to achieve in-network A-NIDS to reduce the reliance on hand-crafted thresholds.

B. Model Extraction and Explanation

There are other related works in the field of model extraction and explanation. However, these methods are either model/architecture-specific [46], [47], local explanations that generate a rule for a single data sample [30]–[33], or extracting rules that need additional calculation (e.g., linear equations) [48], [49]. Our method is different from theirs as we aim to generate model-agnostic, global, and axis-aligned simple rules in an unsupervised manner, which are easily translatable into P4 tables for efficient deployment on switching ASICs.

VII. Conclusion

This paper proposes Genos, a framework for the general in-network deployment of A-NIDS models, which can realize unsupervised model extraction and translation, produce explanations for predictions, and incrementally update deployed rules to handle false positives. Extensive experiments show that Genos can accurately detect various malicious traffic and reach line-rate throughput for online intrusion detection. One limitation of Genos is that it may not be suitable for handling raw data representations as input (e.g., raw packet bytes used in [50] that entail additional convolutional layers for latent feature extraction), as Genos treats every dimension as a semantic feature. However, such representations are criticized by [51] for the possible misalignment of header fields for different protocols, decreasing model performance. Nonetheless, we leave the exploration of interpreting models using other types of data representations as our future work to promote versatility. The authors have provided public access to their code and/or data at https://github.com/Ruoyu-Li/Genos-INFOCOM24.

Acknowledgement

This work is supported by National Key Research and Development Program of China under grant No. 2022YFB3105000, Major Key Project of PCL under grant No. PCL2023A06, and the Shenzhen Key Lab of Software Defined Networking under grant No. ZDSYS20140509172959989.

REFERENCES

[1] Y. Mirsky, T. Doitshman *et al.*, "Kitsune: An ensemble of autoencoders for online network intrusion detection," in *25th Annual Network and Distributed System Security Symposium (NDSS)*, 2018.

[2] M. Du, Z. Chen *et al.*, "Lifelong anomaly detection through unlearning," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2019.

[3] R. Tang, Z. Yang *et al.*, "Zerowall: Detecting zero-day web attacks through encoder-decoder recurrent neural networks," in *IEEE Conference on Computer Communications (INFOCOM)*, 2020.

[4] Y. Wan, K. Xu *et al.*, "Iotargos: A multi-layer security monitoring system for internet-of-things in smart homes," in *IEEE Conference on Computer Communications (INFOCOM)*, 2020.

[5] C. Fu, Q. Li *et al.*, "Realtime robust malicious traffic detection via frequency domain analysis," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2021.

[6] R. Li, Q. Li *et al.*, "Adriot: An edge-assisted anomaly detection framework against iot-based network attacks," *IEEE Internet of Things J.*, vol. 9, no. 13, pp. 10 576–10 587, 2022.

[7] R. Li, Q. Li, and Y. Huang, "Iotensemble: Detection of botnet attacks on internet of things," in *27th European Symposium on Research in Computer Security (ESORICS)*, 2022.

[8] C. Fu, Q. Li, and K. Xu, "Detecting unknown encrypted malicious traffic in real time via flow interaction graph analysis," in *30th Annual Network and Distributed System Security Symposium (NDSS)*, 2023.

[9] G. Siracusano and R. Bifulco, "In-network neural networks," *CoRR*, vol. abs/1801.05731, 2018.

[10] Q. Qin, K. Poularakis *et al.*, "Line-speed and scalable intrusion detection at the network edge via federated learning," in *2020 IFIP Networking Conference*, 2020.

[11] T. Dao and H. Lee, "Jointnids: Efficient joint traffic management for on-device network intrusion detection," *IEEE Trans. Veh. Technol.*, vol. 71, no. 12, pp. 13 254–13 265, 2022.

[12] Z. Xiong and N. Zilberman, "Do switches dream of machine learning?: Toward in-network classification," in *18th ACM Workshop on Hot Topics in Networks (HotNets)*, 2019.

[13] C. Zheng, M. Zang *et al.*, "Automating in-network machine learning," *CoRR*, vol. abs/2205.08824, 2022.

[14] B. M. Xavier, R. S. Guimaraes *et al.*, "Programmable switches for in-networking classification," in *IEEE Conference on Computer Communications (INFOCOM)*, 2021.

[15] C. Busse-Grawitz, R. Meier *et al.*, "pforest: In-network inference with random forests," *CoRR*, vol. abs/1909.05680, 2019.

[16] C. Zheng and N. Zilberman, "Planter: seeding trees within switches," in *ACM SIGCOMM Conference, Poster and Demo Sessions*, 2021.

[17] G. Xie, Q. Li *et al.*, "Mousika: Enable general in-network intelligence in programmable switches by knowledge distillation," in *IEEE Conference on Computer Communications (INFOCOM)*, 2022.

[18] Y. Li, J. Bai *et al.*, "Rectified decision trees: Exploring the landscape of interpretable and effective machine learning," *CoRR*, vol. abs/2008.09413, 2020.

[19] A. S. Jacobs, R. Beltiukov *et al.*, "Ai/ml for network security: The emperor has no clothes," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2022.

[20] O. Bastani, C. Kim, and H. Bastani, "Interpreting blackbox models via model extraction," *CoRR*, vol. abs/1705.08504, 2017.

[21] A. Binbusayyis and T. Vaiyapuri, "Unsupervised deep learning approach for network intrusion detection combining convolutional autoencoder and one-class SVM," *Appl. Intell.*, vol. 51, no. 10, pp. 7094–7108, 2021.

[22] Y. Dong, Q. Li *et al.*, "Horuseye: Realtime iot malicious traffic detection framework with programmable switches," in *32nd USENIX Security Symposium, USENIX Security*, 2023.

[23] P. Bosshart, D. Daly *et al.*, "P4: programming protocol-independent packet processors," *Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, 2014.

[24] B. Letham, C. Rudin *et al.*, "Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model," *CoRR*, vol. abs/1511.01644, 2015.

[25] N. Frosst and G. E. Hinton, "Distilling a neural network into a soft decision tree," *CoRR*, vol. abs/1711.09784, 2017.

[26] G. Zhou, Z. Liu *et al.*, "An efficient design of intelligent network data plane," in *32st USENIX Security Symposium (USENIX Security)*, 2023.

[27] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should i trust you?": Explaining the predictions of any classifier," in *22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2016.

[28] L. Breiman, J. H. Friedman *et al.*, *Classification and Regression Trees*. Wadsworth, 1984.

[29] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *3rd International Conference on Learning Representations (ICLR)*, 2015.

[30] S. M. Lundberg and S. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems*, 2017.

[31] M. T. Ribeiro, S. Singh, and C. Guestrin, "Anchors: High-precision model-agnostic explanations," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[32] W. Guo, D. Mu *et al.*, "Lemna: Explaining deep learning based security applications," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2018.

[33] D. Han, Z. Wang *et al.*, "Deepaid: Interpreting and improving deep learning-based anomaly detection in security applications," in *2021 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2021.

[34] "Cicflowmeter," https://github.com/ahlashkari/CICFlowMeter, Canadian Institute for Cybersecurity, 2016.

[35] "Netflow," https://cisco.com/c/dam/en/us/td/docs/security/stealthwatch/netflow/Cisco_NetFlow_Configuration.pdf, Cisco, 2023.

[36] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *4th International Conference on Information Systems Security and Privacy (ICISSP)*, 2018.

[37] T. M. Booij, I. Chiscop *et al.*, "Ton_iot: The role of heterogeneity and the need for standardization of features and attack types in iot network intrusion data sets," *IEEE Internet Things J.*, vol. 9, no. 1, pp. 485–496, 2022.

[38] S. Wang, C. Sun *et al.*, "Martini: Bridging the gap between network measurement and control using switching asics," in *28th IEEE International Conference on Network Protocols (ICNP)*, 2020.

[39] J. Xing, Q. Kang, and A. Chen, "Netwarden: Mitigating network covert channels while preserving performance," in *29th USENIX Security Symposium (USENIX Security)*, 2020.

[40] D. Barradas, N. Santos *et al.*, "Flowlens: Enabling efficient flow classification for ml-based network security applications," in *28th Annual Network and Distributed System Security Symposium (NDSS)*, 2021.

[41] R. Miao, H. Zeng *et al.*, "Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics," in *ACM SIGCOMM Conference (SIGCOMM)*, 2017.

[42] S. Sengupta, H. Kim, and J. Rexford, "Continuous in-network round-trip time monitoring," in *ACM SIGCOMM Conference (SIGCOMM)*, 2022.

[43] E. C. Molero, S. Vissicchio, and L. Vanbever, "Fast in-network *GraY* failure detection for isps," in *ACM SIGCOMM Conference (SIGCOMM)*, 2022.

[44] Z. Liu, H. Namkung *et al.*, "Jaqen: A high-performance switch-native approach for detecting and mitigating volumetric ddos attacks with programmable switches," in *30th USENIX Security Symposium (USENIX Security)*, 2021.

[45] M. Zhang, G. Li *et al.*, "Poseidon: Mitigating volumetric ddos attacks with programmable switches," in *27th Annual Network and Distributed System Security Symposium (NDSS)*, 2020.

[46] D. Kazhdan, B. Dimanov *et al.*, "MEME: generating RNN model explanations via model extraction," *CoRR*, vol. abs/2012.06954, 2020.

[47] P. Liznerski, L. Ruff *et al.*, "Explainable deep one-class classification," in *9th International Conference on Learning Representations (ICLR)*, 2021.

[48] M. W. Craven and J. W. Shavlik, "Using sampling and queries to extract rules from trained neural networks," in *International Conference on Machine Learning (ICML)*, 1994.

[49] Z. Zhou, Y. Jiang, and S. Chen, "Extracting symbolic rules from trained neural network ensembles," *AI Commun.*, vol. 16, no. 1, pp. 3–15, 2003.

[50] G. Marín, P. Casas, and G. Capdehourat, "Deep in the dark - deep learning-based malware traffic detection without expert knowledge," in *IEEE Security and Privacy Workshops (SPW)*, 2019.

[51] J. Holland, P. Schmitt *et al.*, "New directions in automated traffic analysis," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2021.